

Characterization of Silent Stores

Gordon B.Bell
Kevin M. Lepak
Mikko H. Lipasti
University of Wisconsin—Madison
http://www.ece.wisc.edu/~pharm

Background

- Lepak, Lipasti: *On the Value Locality of Store Instructions*: ISCA 2000
- Introduced Silent Stores
  - A memory write that does not change the system state
  - Silent stores are real and non-trivial
    - 20%-60% of all dynamic stores are silent in SPECINT-95 and MP benchmarks (32% average)

Why Do We Care?

- Reducing cache writebacks
  - Reducing writeback buffering
- Reducing true and false sharing
- Write operations are generally more expensive than reads

Code Size / Efficiency

```
R(I1,I2,I3) = V(I1,I2,I3) - A(0)*(U(I1,I2,I3)) - A(1)*(U(I1-1,I2,I3) + U(I1+1,I2,I3)
+ U(I1,I2-1,I3) + U(I1,I2+1,I3) + U(I1,I2,I3-1) + U(I1,I2,I3+1)) - A(2)*(U(I1-1,I2-1,I3)
+ U(I1+1,I2-1,I3) + U(I1-1,I2+1,I3) + U(I1+1,I2+1,I3) + U(I1,I2-1,I3-1)
+ U(I1,I2+1,I3-1) + U(I1,I2-1,I3+1) + U(I1,I2+1,I3+1) + U(I1-1,I2,I3-1)
+ U(I1-1,I2,I3+1) + U(I1+1,I2,I3-1) + U(I1+1,I2,I3+1)) - A(3)*(U(I1-1,I2-1,I3-1)
+ U(I1+1,I2-1,I3-1)  + U(I1-1,I2+1,I3-1) + U(I1+1,I2+1,I3-1) + U(I1-1,I2-1,I3+1)
+ U(I1+1,I2-1,I3+1) + U(I1-1,I2+1,I3+1) + U(I1+1,I2+1,I3+1))
```

Example from *mgrid* (SPECFP-95)

Eliminating this expression (when silent) removes over 100 static instructions (2.4% of the total dynamic instructions)

This Talk

- Characterize silent stores
  - Why do they occur?
  - Source code case studies
- Silent store statistics
- Critical silent stores
- Goal: provide insight into silent stores that can lead to novel innovations in detecting and exploiting them

Terminology

- ***Silent Store:*** A memory write that does not change the system state
- ***Store Verify:*** A load, compare, and conditional store (if non-silent) operation
- ***Store Squashing:*** Removal of a silent store from program execution

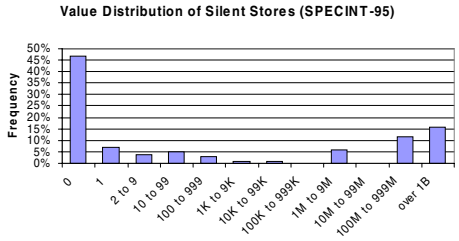
An Example

```
for (i = 0; i < 32; i++){
   time_left[i] -= MIN(time_left[i],time_to_kill);
}
```

Example from *m88ksim*

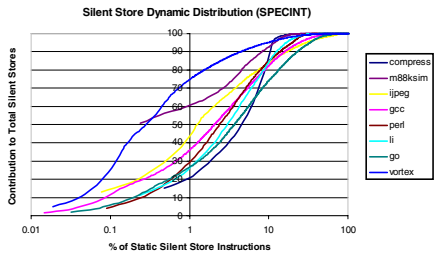
- This store is silent in over 95% of the dynamic executions of this loop
- Difficult for compiler to eliminate because how often the store is silent may depend on program inputs

Value Distribution



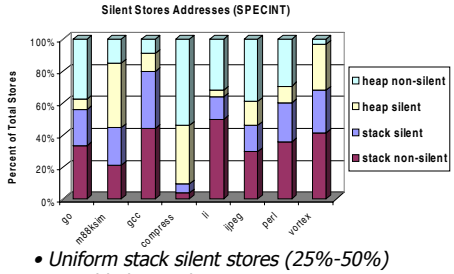
Both values and addresses are likely to be silent

Frequency of Execution



Few static instructions contribute to most silent stores

Stack / Heap



- *Uniform stack silent stores (25%-50%)*
- *Variable heap silent stores*

Stores Likely to be Silent

- 4 categories based on previous execution of that particular static store
- Same Location, Same Value
  - A silent store stores the same value to the same location as the last time it was executed
  - Common in loops

Same Location, Same Value

```
for (anum = 1; anum <= maxarg; anum++) {
    argflags = arg[anum].arg_flags;
```

Example from *perl*

- *argflags* is a stack-allocated temporary variable (same location)
- *arg_flags* is often zero (same value)
- Silent 71% of the time

Stores Likely to be Silent

- Different Location, Same Value
  - A silent store stores the same value to a different location as the last time it was executed
  - Common in instructions that store to an array indexed by a loop induction variable

Different Location, Same Value

```
for(x = xmin; x <= xmax; ++x)
      for(y = ymin; y <= ymax; ++y){
            s = y*boardsize+x;
            ...
            ltrscr -= ltr2[s];
            ltr2[s] = 0;
            ltr1[s] = 0;
            ltrgd[s] = FALSE;
      }
```

Example from *go*

- Clears game board array
- Board is likely to be mostly zero in subsequent clearings
- Silent 86%, 43%, 77% of the time, respectively

Stores Likely to be Silent

- Same Location, Different Value
  - A silent store stores a different value to the same location as the last time it was executed
  - Rare, but can be caused by:
    - Intervening static stores to the same address
    - Stack frame manipulations

Same Location, Different Value

```
for(x = xmin; x <= xmax; ++x)
      for(y = ymin; y <= ymax; ++y){
            s = y*boardsize+x;
            ...
            ltrscr -= ltr2[s];
            ltr2[s] = 0;
            ltr1[s] = 0;
            ltrgd[s] = FALSE;
      }
```

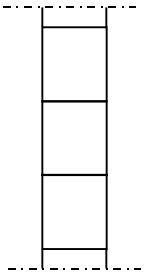
Example from *go*

- *ltrscr* is a global variable (same location)
- *ltr2* is indexed by loop induction variable (different value)
- Silent 86%, but of that 98% is Same Location, Same Value

Callee-Saved Registers

call foo()
call bar()
...
call foo()

```
*** void foo() ***
sw $17,28($fp)
...
*** void bar() ***
sw $17,28($fp)
...
```

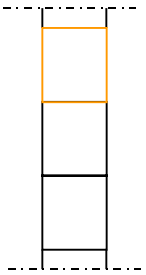


$17 is callee-saved

Callee-Saved Registers

call foo()
call bar()
...
call foo()

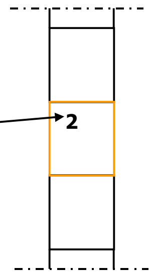
```
*** void foo() ***
sw $17,28($fp)
...
*** void bar() ***
sw $17,28($fp)
...
```



Callee-Saved Registers

call foo()
call bar()
...
call foo()

```
*** void foo() ***
sw $17,28($fp)
...
*** void bar() ***
sw $17,28($fp)
...
```

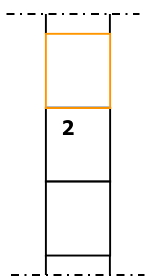


PACT 2000 G. Bell, K. Lepak and M. Lipasti

Callee-Saved Registers

call foo()
call bar()
...
call foo()

```
*** void foo() ***
sw $17,28($fp)
...
*** void bar() ***
sw $17,28($fp)
...
```

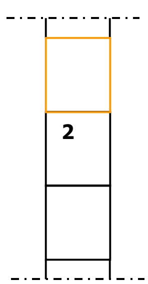


PACT 2000 G. Bell, K. Lepak and M. Lipasti

Callee-Saved Registers

call foo()
call bar()
...
call foo()

```
*** void foo() ***
sw $17,28($fp)
...
*** void bar() ***
sw $17,28($fp)
...
```

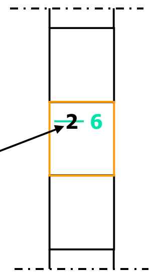


PACT 2000 G. Bell, K. Lepak and M. Lipasti

Callee-Saved Registers

call foo()
call bar()
...
call foo()

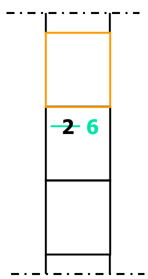
```
*** void foo() ***
sw $17,28($fp)
...
*** void bar() ***
sw $17,28($fp)
...
```



Callee-Saved Registers

call foo()
call bar()
...
call foo()

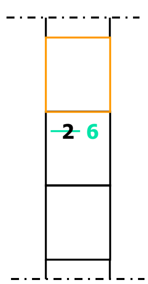
```
*** void foo() ***
sw $17,28($fp)
...
*** void bar() ***
sw $17,28($fp)
...
```



Callee-Saved Registers

call foo()
call bar()
...
call foo()

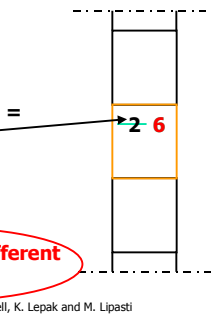
```
*** void foo() ***
sw $17,28($fp)
...
*** void bar() ***
sw $17,28($fp)
...
```



Callee-Saved Registers

```
call foo()
call bar()
...
call foo()

*** void foo() ***
sw $17,28($fp)
...
*** void bar() ***
sw $17,28($fp)
...
```



Stores Likely to be Silent

- Different Location, Different Value
  - A static silent store stores a different value to a different location as the last time it was executed
  - Example: nested loops

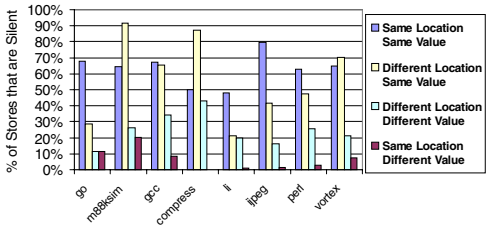
Different Location, Different Value

```
NODE ***xlsave(NODE **nptr,...){
...
for (; nptr != (NODE **) NULL; nptr = va_arg(pvar, NODE **)){
       ...
       *--xlstack = nptr;
       ...
}
```

Example from *li*

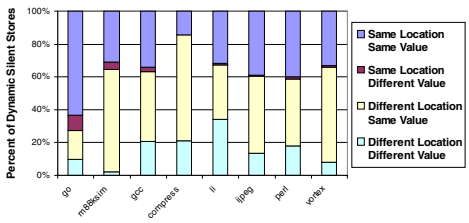
- *xlstack* is continually decremented (different location)
- *nptr* is set to next function argument (different value)
- Silent if subsequent calls to *xlsave* store the same set of nodes to the same starting stack address

Likelihood of Being Silent



Silence can be accurately predicted based on category

Silent Store Breakdown

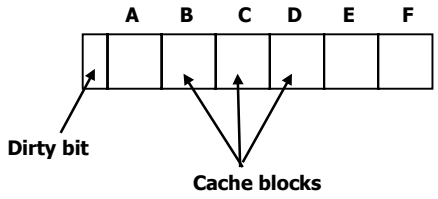


Stores that can be predicted silent (Same Value) are a large portion of all silent stores

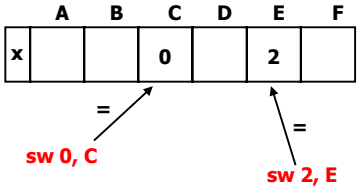
Critical Silent Stores

- ***Critical Silent Store:*** A specific dynamic silent store that, if not squashed, will cause a cacheline to be marked as dirty and hence require a writeback

Critical Silent Stores

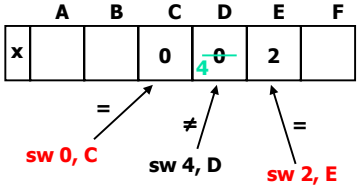


Critical Silent Stores



*Both silent stores are **critical** because the dirty bit would not have been set if silent stores are squashed*

Non-Critical Silent Stores



*No silent stores are **critical** because the dirty bit is set by a non-silent store (regardless of squashing)*

Critical Silent Stores Who Cares?

- It is sufficient to squash only critical silent stores to obtain maximal writeback reduction
- Squashing non-critical silent stores:
  - Incurs store verify overhead with no reduction in writebacks
  - Can cause additional address bus transactions in multiprocessors

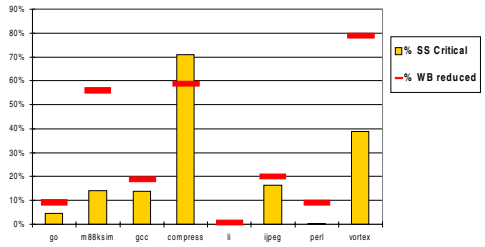
Critical Silent Stores: Example

```
do{
        *(htab_p-16) = -1;
        *(htab_p-15) = -1;
        *(htab_p-14) = -1;
        *(htab_p-13) = -1;
        ...
        *(htab_p-2)  = -1;
        *(htab_p-1)  = -1;
} while ((i -= 16) >= 0);
```

Example from *compress*

- These 16 stores fill entire cache lines
- If all stores to a line are silent, then they are all critical as well
- 19% of all writebacks can be eliminated

Writeback Reduction



Squashing only a subset of silent stores results in significant writeback reduction

Conclusion

- Silent Stores occur for a variety of values and execution frequencies
- Silent Store causes:
  - Algorithmic (bad programming?)
  - Architecture / compiler conventions
- Squashing only critical silent stores is sufficient for removing all writebacks

Future Work

- Silence prediction
  - Store verify only if have reason to believe that store is:
    - Silent
    - Critical
- Multiprocessor Silent Stores
  - Extend notion of criticality to include silent stores that cause sharing misses as well as writebacks