Compiler Controlled Register Stack Management

Alex Settle
Dan Connors
Gerolf Hoflehner
Dan Lavery

Software Managed RSE

- RSE reduces procedure boundary spill code
- RSE loads and stores stall processor
  - RSE stalls on some benchmarks are significant
- Compiler can improve efficiency of RSE
- Improve performance without changing hw
- Compile time control flow analysis

Design Space

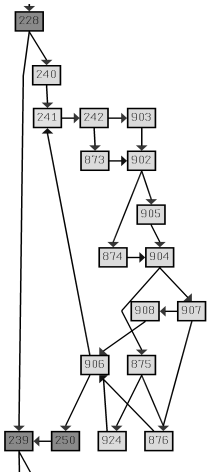
- Goal is to reduce RSE usage
- Static code analysis drove design decisions
  - Identify code regions that stress RSE
  - Areas for improvement for code generation
- Multi tiered approach
  - Compact stacked register allocation
  - alloc instructions to control RS size

Static Code Analysis

- Examined assembly code for SpecCint 2000
- Examples from hot functions in benchmarks
  - 186.crafty evaluate()
  - 176.gcc cse_insn
- Stacked register pressure
  - Number of live stacked registers across a block
  - Criteria for control flow region formation

186.crafty evaluate()

- Evaluate() 33% ex time 42 locals 0 unused 0 rot
- Green > 75% reg pressure
- Blue 50%-75% reg pressure
- Gold – call block
- Increase frame size for green region
- Several call sites above this region
- OK to enter function with frame size <75% of max



176.gcc cse_insn Large Live Ranges

- 9 of 87 locals set to 0 in first block
  - Push down assignments closer to first use
  - May not need stacked regs for these variables

```
alloc     r41=ar.pfs,2,85,4,8
ld8       r117=[r17]
mov       r116=ar.lc
mov       r115=pr
add       r113=0,r0
add       r111=0,r0
add       r109=0,r0 ;;
add       r108=0,r0
add       r107=0,r0
add       r106=0,r0
add       r105=0,r0
add       r63=0,r0
add       r60=0,r0
```

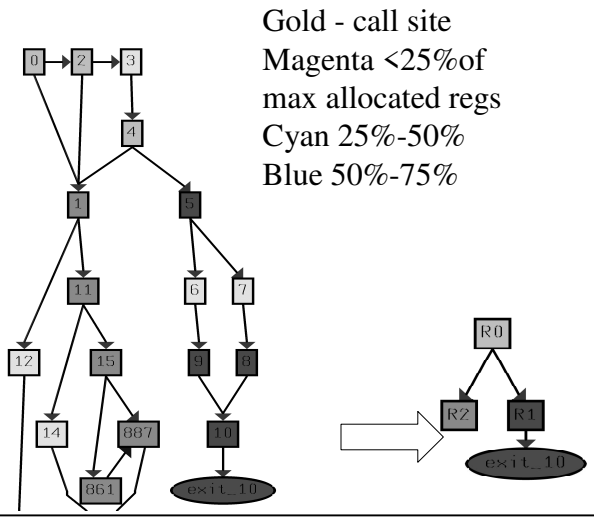
Call Site Interface

- Traditional architecture
  - Callee saves registers that it uses on entry (st)
  - Restores on exit (ld)
- Architecture with RSE
  - Caller - live ranges that cross calls are assigned stacked registers
    - Automatically saved in memory by RSE
  - Callee does not need to save registers that it uses
  - Fewer memory accesses

Register Pressure

- Internally fragmented Register Stack
- Control flow regions
  - Group adjacent blocks of similar register pressure
  - Let compiler size stack frame for each region
- Modify stacked register allocation
  - Long live ranges map to bottom of stack
  - Push frequently dead registers to top of stack
  - Reduce internal fragmentation

Regions

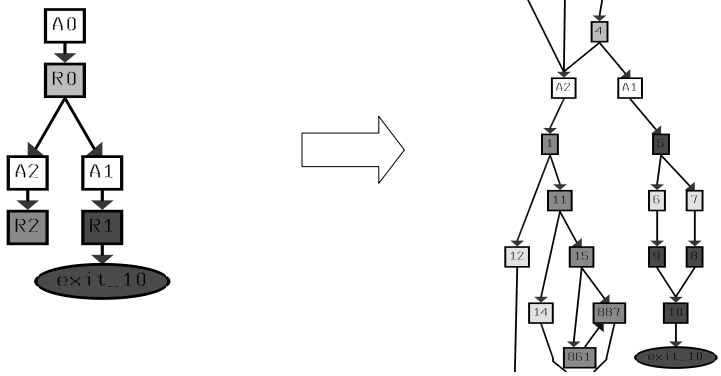


Alloc Instruction Uses

- Changes stack frame size at run time
- Compiler inserts alloc at each region entry block
- Only reduce frame size for regions with call sites
  - Outgoing parameter registers need to be renamed
  - Increase frame size with register pressure
- Rename function parameter registers
  - Additional mov instructions

Alloc Placement

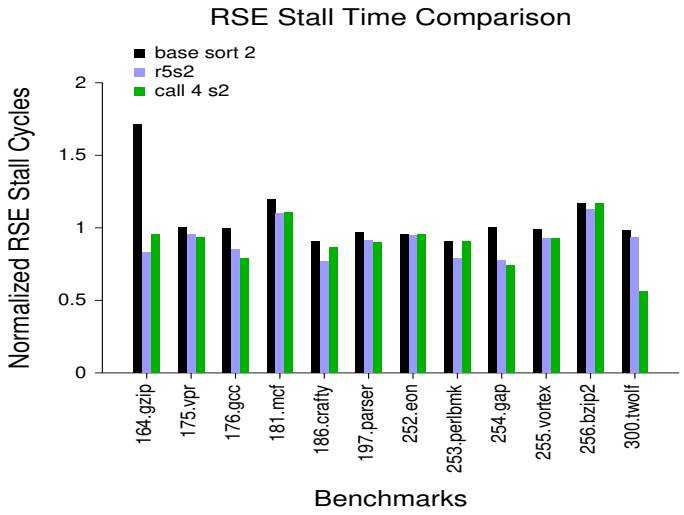
New alloc on entry into region
Tunable region formation heuristic



Stacked Register Sorting

- Reduce internal fragmentation
  - Optimization phase precedes alloc phase
- Facilitates stack frame reduction
- Can decrease the total stacked register count
- First stage in alloc optimization
- Example code
  - gcc propagate_block()
  - Reduces internal fragmentation

RSE Stall Cycle Comparison



Analysis

- Alloc insertion reduces RSE stall time
  - These reductions do not always lead to overall performance gains
- Many issues at play
- Register allocator first changes the order in which stacked regs are allocated
  - Can lead to improvement if there is a reduction in total number of stacked registers
- Introducing new instructions alters the instruction schedule
- Extra alloc and mov instructions can increase dynamic instruction count if introduced inside a loop

Related Work

- Settle et al
- Shrink Wrapping
- Douillet et al
- Chaitin – register coloring
- Weldon et al

Future Work

- Profile guided optimization
  - Combine path frequency info with register pressure
- Live range shrinking
  - Static techniques to make shorter live ranges
- Further analysis of the effects that register sorting has on performance

Performance Impact of RSE

- RSE stalls processor on overflow and underflow
- Can account for > 4% of total program execution time
- RSE spills and fills consume cache resources

The End

Performance Analysis

- Pick a bmark and show what is going on with performance