Compiler and Architecture Issues with CRegs

Dept. of Electrical Engineering
University of Minnesota

dahl@ee.umn.edu

Ambiguously Aliased Objects

- An ambiguous alias occurs when two or more names may or may not refer to the same object. (arrays, pointers, call by reference parameters, common blocks)
  Examples:

```
void func1(int *a, int *b)          ...  = a[i] +
{   *a = 5;                         a[j] = ...
    *b = 4;                         ...  = a[i]
    print *a,*b;
}
```

- Frequently used objects should be allocated to registers to reduce access time and memory bandwidth
- Register space is separate from memory address space, ambiguously aliased object may reside in multiple registers
- When one register is written to, the other copies must be updated for consistency

CRegs

CReg: name:

address	datum

Advantages:

- Fast access time
- Does not conflict with memory accesses
- Access time is predictable
- Instruction bandwidth is saved
- Solves the ambiguous alias problem
- No garbage data, “replacement policy” in compiler
- Backward compatible, easily turned off

CRegs

Disadvantages:

- May increase compile time analysis for sets.
- Increases register file chip area.
- Instruction set becomes larger.
- Requires special forwarding registers.
- Not transparent to system.

Related Work

- H. Dietz and C. Chi, “CRegs: A New Kind of Memory for Referencing Arrays and Pointers”, *Supercomputing ’88*, pages 360–367, 1988
- B. Heggy and M. Soffa, “Architectural Support for Register Allocation in the Presence of Aliasing”, *Supercomputing ’90*, pages 730–739, 1990.
- T. Chiueh, “An Integrated Memory Management Scheme for Dynamic Alias Resolution”, *Supercomputing ’91*, pages 682–691, 1991.
- C. Chi, “Compiler-Driven Cache Management Using a State Level Transition Diagram”, Ph.D. Thesis, Purdue University, 1989.

CRegs Operation

- Store to memory:
  1. calculate effective address
  2. if address match, copy value to other CReg(s)
  3. store value to memory

  {4. update address field of CReg}

- Load from memory:
  1. calculate effective address
  2. if address match, copy from another CReg
     else load from memory into value
  3. update address field of CReg

- Write to CReg file:
  1. write destination register value field
  2. clear address field

CRegs Example

1) `input i,j`
2) `a[i] = 7`
3) ...
4) `a[j] = a[i] + a[k]`

1) `r1` ← `i`, `r2` ← `j`
2) `r4` ← 7; `&r4` ← null

copy `r4` to CRegs matching `&a[i]`	CReg
store `r4` to `&a[i]`	store
`&r4` ← `&a[i]`	(allocating)

3) ...
4) (no load of `a[i]`, it's allocated to `r4`)

if (`&a[k]` = `&r4`)	`r5` ← `r4` (squash)	CReg
else	load `r5` from `&a[k]`	load

`r6` ← `r4` + `r5`; `&r6` ← null

copy `r6` to CRegs matching `&a[j]`	CReg
store `r6` to `&a[j]`	store
	(nonallocating)

CRegs Architecture

- Associative set size
  - matching
  - writing
- Register file size
  - chip area
  - more objects allocated
- Pipeline impact

name	address	data
7	0x7abc	-1
6	0x01d9	0
5	0x220b	-2
4	0x1000	32
3	0x0000	978
2	0x100a	42
1	0x100a	42
0	0x1000	0

CReg Set Size Tradeoffs

Small CReg Set Advantages:

- Easier to write all matching CRegs
- Fewer simultaneous reads during matching
- Fewer dual issue constraints
- More freedom for scheduler

Big CReg Set Advantages:

- Requires less alias analysis
- Easier to allocate pointers
- Fewer constraints on the register allocator

Instruction Set Architecture

- Original load and store instructions
- CReg matching instructions
- Address clear instructions
- Context switching

Live Range Construction

- Iterative data flow
- Organize local and formal array references by index calculation.
- Index calculation annotated with value numbers for uniqueness.
- Create live ranges for each unique reference to same array.
- General pointers not allocated
- Globals not allocated

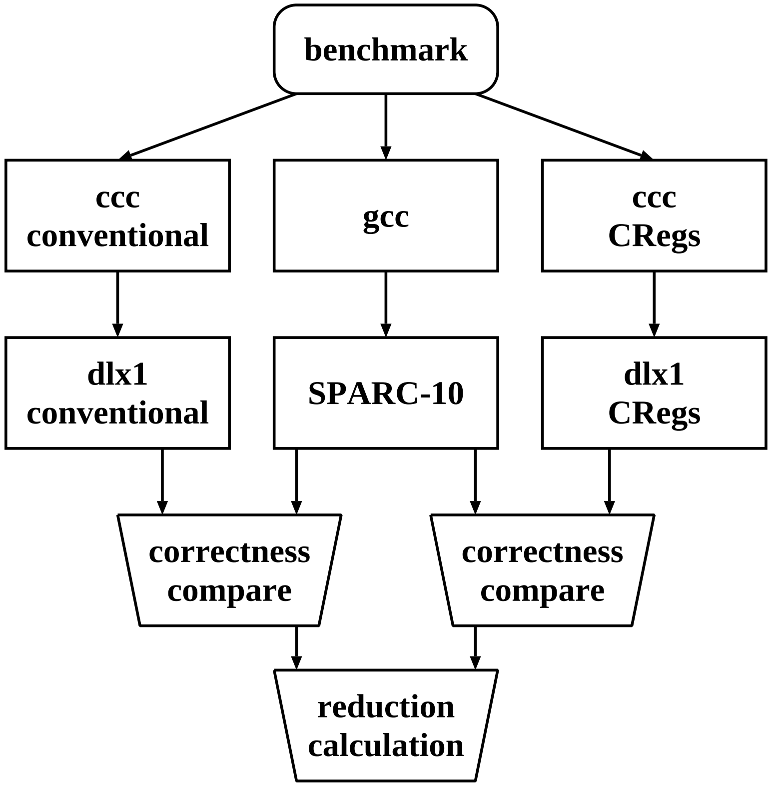
Alias Set Construction

- Only need if CRegs divided into sets
- Simple algorithm: accumulation of objects that are at some point ambiguously aliased.
- $A \equiv B$ and $B \equiv C \Rightarrow A \equiv C$
- Rewrite as alias pair representation, less conservative
- Operations: `find_alias_color(node)`, `find_first_last(color)`
- Represented as a circularly linked list of live range nodes.

Register Allocation

- Modification to Briggs' Optimistic Coloring Algorithm.
- Alias set controls which colors are available during select phase.
- If alias set has not been assigned a CRegs set, use a round-robin scheme to pick a set.
- If alias set has a member that is colored, allow only colors from that CReg set.
- Spill code heuristic: store at every DEF of an ambiguously aliased object
- Can be used with other refinements: Bernstein

Testing Methodology



$$\%\ reduction = \frac{\mathit{Conventional} - \mathit{CReg}}{\mathit{Conventional}} \times 100$$

Speedup Rationale

- Fewer loads for ambiguously aliased objects.
- Not all loads in instruction stream generate memory read (squashed).
- Stores can only be eliminated if all members of alias set are allocated.
- Eliminate loads → smaller code
  more objects allocated → more isolation code

What to Measure?

Measurements based on dynamic instruction counts not execution time.

- Eliminates variability:
  - memory latency
  - memory architecture
  - TLB delays
  - cache size
  - cache line size
  - replacement policy
  - hit ratio

- Loss of detail:
  - pipeline effects
  - cache access patterns
  - execution time

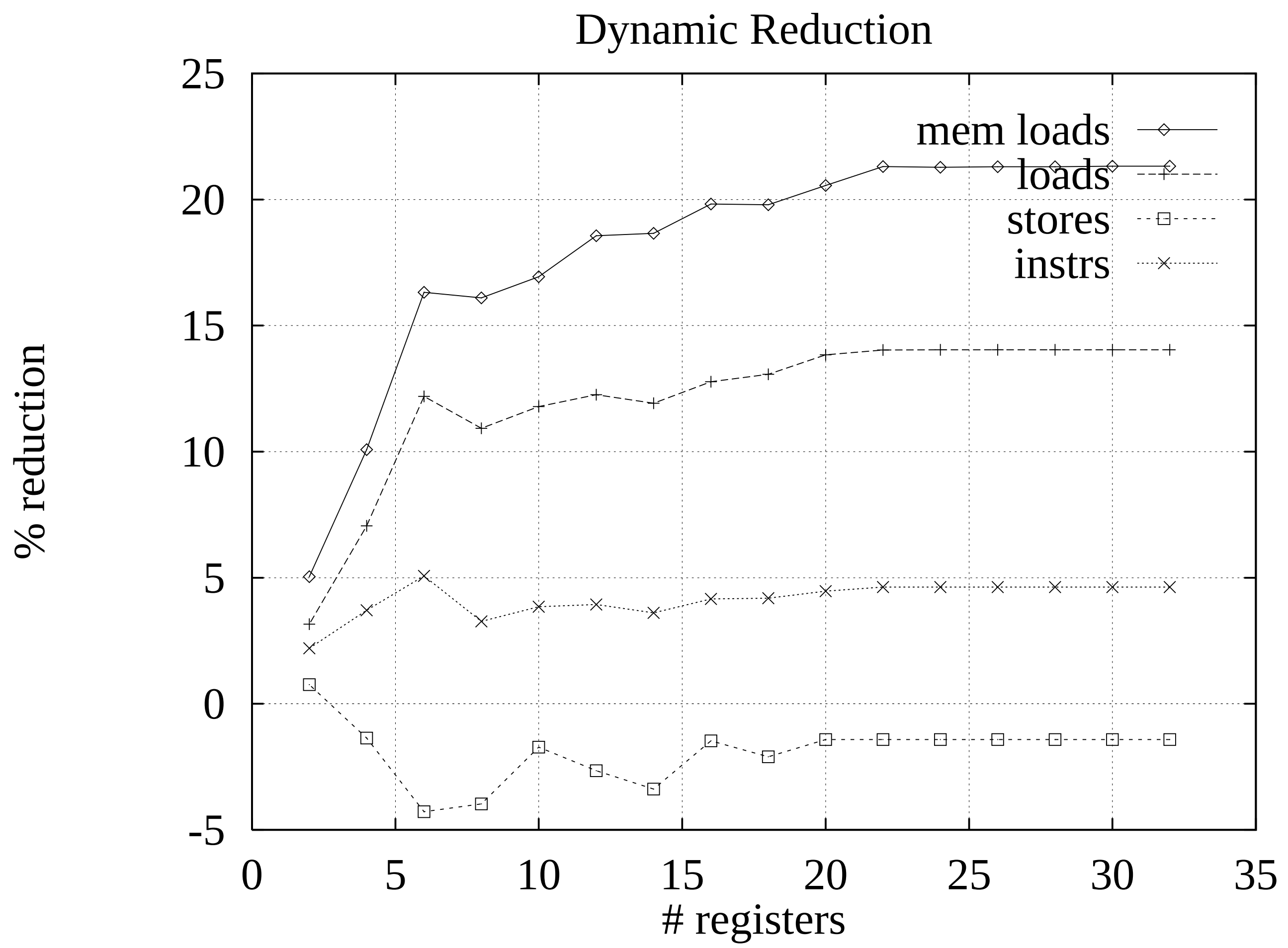
Percent Reduction Summary for Register File Size of 32

	Maximal CReg Set			
Benchmark	mem loads	loads	stores	instrs.
nasa7/vpenta	36.62	36.60	-0.01	8.52
eqntot	5.58	0.95	-1.11	1.94
tomcatv	21.06	20.80	0.00	7.11
matrix300	0.11	0.11	0.00	-0.03
doduc/debico1	47.56	40.67	-0.31	2.90
doduc/debico2	74.44	66.60	-0.12	13.75
doduc/dcoera	9.00	0.00	0.00	-0.61
doduc/dyeh	4.21	0.00	0.00	0.00
doduc/yeh	11.22	0.00	-0.03	0.00
Livermore Loops	8.39	4.62	-0.02	1.45
ppm	21.30	14.04	-1.42	4.63

Percent Reduction Summary for Register File Size of 32

	Set Size 4			
Benchmark	mem loads	loads	stores	instrs.
nasa7/vpenta	33.03	32.96	-0.01	7.65
eqntot	4.88	0.95	-1.13	6.45
tomcatv	19.24	17.32	0.00	6.09
matrix300	0.11	0.11	0.00	-0.03
doduc/debico1	49.02	42.19	-0.38	2.86
doduc/debico2	75.33	65.35	-0.12	13.49
doduc/dcoera	4.38	-1.46	-1.79	-1.51
doduc/dyeh	1.05	0.00	0.00	0.00
doduc/yeh	4.08	0.00	0.00	0.00
Livermore Loops	7.92	4.16	-0.07	1.23
ppm	19.78	13.74	-1.57	4.47

ppm, maximal CReg set



Analysis

- Dynamic loads were reduced 0 to 75%.
- Small CReg sets cause only a slight increase in memory operations compared to a maximal set.
- The dynamic instruction count reduction ranged from -1.5% to 13.8%.
- Increased register pressure presents fewer opportunities to reduce loads.

Summary

- There are generally fewer instructions executed with CRegs.
- There are fewer memory operations executed.
- Overall data access latencies are reduced by referencing registers more.
- Compiler analysis can be minimal

Contributions

- Define CRegs operation.
- Creation of new alias analysis algorithm.
- New register allocation strategy for CRegs.
- CRegs architecture performance study.
- Design and develop prototype compiler that allocates ambiguous aliases to CRegs.
- Measure empirically the spill code reduction of CRegs over conventional registers.