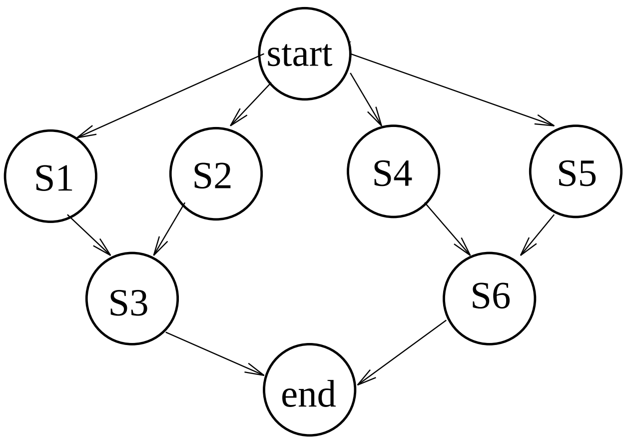
An Experimental Study of Several Cooperative Register Allocation and Instruction Scheduling Strategies

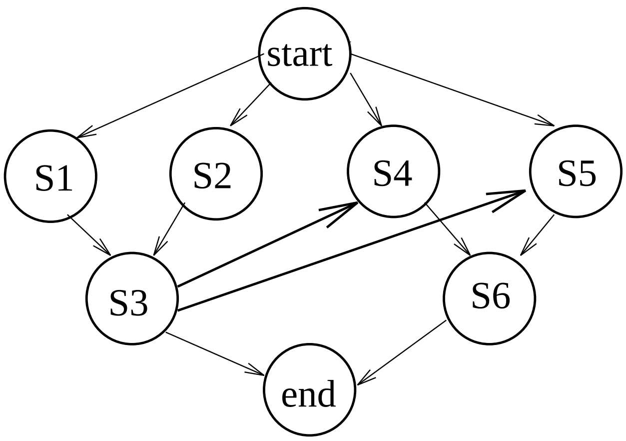
Cindy Norris
Appalachian State University
and
Lori Pollock
University of Delaware

Register Allocation Before Scheduling

- Adds data dependences
- Reduces potential for reordering

Unallocated Code	**Allocated Code**
S1: v1 = 3	S1: r1 = 3
S2: v2 = 5	S2: r2 = 5
S3: v3 = v1 * v2	S3: r1 = r1 * r2
S4: v4 = 7	S4: r1 = 7
S5: v5 = 9	S5: r2 = 9
S6: v6 = v4 + v5	S6: r1 = r1 + r2





Scheduling Before Register Allocation

- Increases number of variables competing for registers

- Increases spill code

Before Scheduling	**Live**
S1: v1 = 3	v1
S2: v2 = 5	v1, v2
S3: v3 = v1 * v2	
S4: v4 = 7	v4
S5: v5 = 9	v4, v5
S6: v6 = v4 + v5	

After Scheduling	**Live**
S1: v1 = 3	v1
S2: v2 = 5	v1, v2
S4: v4 = 7	v1, v2, v4
S5: v5 = 9	v1, v2, v4, v5
S3: v3 = v1 * v2	v4, v5
S6: v6 = v4 + v5	

Locally Cooperative Strategies

$*$ Goodman, Hsu (1988)

Integrated Prepass Scheduling
Dag-Driven Register Allocation

$*$ Bradlee, Eggers, Henry (1991)

Register Allocation with Schedule Estimates

$*$ Pinter (1993)

Parallel Interference Graph

$*$ Norris, Pollock (1993)

Scheduler-Sensitive Global Register Allocation

$*$ Brasier, Sweany, Beaty (1995)

Combining Register Assignment Interference Graphs

Globally Cooperative Strategies

∗ Freudenberger, Ruttenberg (1992)
Integrated Trace Scheduling and Register Assignment

∗ Moon, Ebcioglu (1992)
Resource-Constrained Scheduling

∗ Berson, Gupta, Soffa (1993, 1994)
Unified Resource Allocator
Resource Spackling

∗ Norris, Pollock (1995)
Register Allocation Sensitive Region Scheduling

Previous Work

- Focused on cooperative local instruction scheduling and register allocation

- Focused on cooperative global instruction scheduling and register allocation

- Has not considered providing cooperation at both levels

Our Work

* Maintain separate allocation and scheduling phases to avoid complexity
* Perform global scheduling, then allocation, then local scheduling
* Apply local scheduling after allocation to schedule spill code
* Provide cooperation at two levels

OVERALL PICTURE

RASER: Register Allocation Sensitive Region Scheduling

- Modifications made to Region Scheduling (Gupta, Soffa, 1990) to consider register allocation needs

- Region scheduling transformations to increase available fine-grain parallelism

- Prevent transformations that increase spilling

- Also transforms code to reduce competition for registers (live value reduction)

Algorithm RASER($entry$)

```
Loop {
  for (i = 0; i < |R|; i++)
    { est_parallelism(R_i), ips(R_i), max_live(R_i) }
  sch_list = need_parallelism(R)
  max_list = too_many_max_live(R)
  t = false,      i = 0
  while (i < |max_list| && t = false)
    { t = live_value_reduction(max_list_i), i++ }
  i = 0
  while (i < |sch_list| && t = false) {
    /* allocation sensitive transformations */
    t = peel(sch_list_i)
    if (t = false) t = loop_invariant_motion(sch_list_i)
    if (t = false) t = forward_motion(sch_list_i)
    if (t = false) t = backward_motion(sch_list_i)
    i++
  }
  if (t) cleanup_pdg()
} while (t)
```

Allocation Sensitive Transformation

Before Loop Invariant Code Motion

```
for i = 1 to 10
{
    .
    .
    k = 3
    a[i] = k
    .
    .
}
```

After Loop Invariant Code Motion

```
k = 3
for i = 1 to 10
{
    .
    .
     a[i] = k
    .
    .
}
```

After code motion, k live during entire execution of loop

Live Value Reduction

Before LVR		**After LVR**	
	v1 = 10		.
	.	L2:	if (P) goto L1
L2:	if (P) goto L1		.
	.		v1 = 10
	.		v5 = v1 + 3
	v5 = v1 + 3		.
	.		v1 = 10
	v6 = v1 * 7		v6 = v1 * 7
	goto L2		goto L2
L1:	.	L1:	.

After LVR, v1 live immediately before each use and dead after each use

SSG: Scheduler Sensitive Global Register Allocation

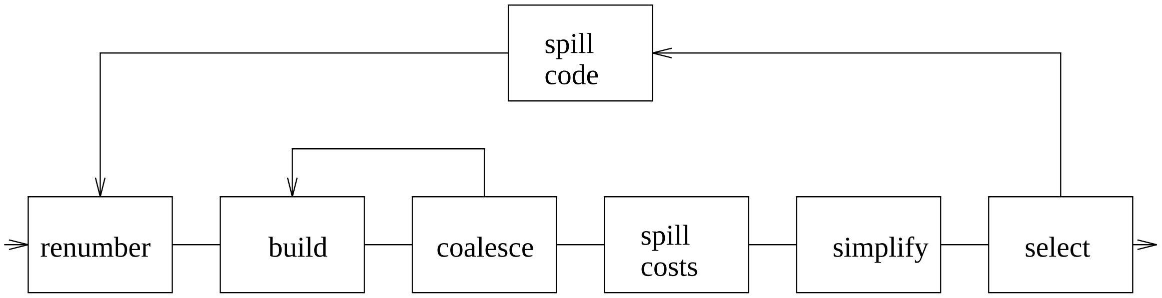
- Register allocation made sensitive to the subsequent local scheduler

- SSG anticipates decisions of scheduler

- User information used by the scheduler: data dependence graph

- Based on a successful global register allocator: OA (Briggs, et al. 1992)

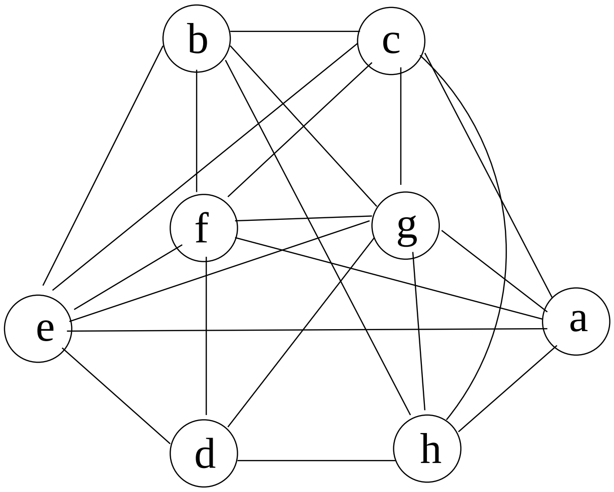
Scheduler-Sensitive Global Register allocation



* Renumber: Avoid adding back anti-dependences
* Build: Represent all possible code orderings in interference graph
* Spill Costs: Take into account instruction execution times
* Simplify: Spill, rematerialize or eliminate interferences by adding scheduler-sensitive dag edges

Interference Graph from DDG

Interference Graph

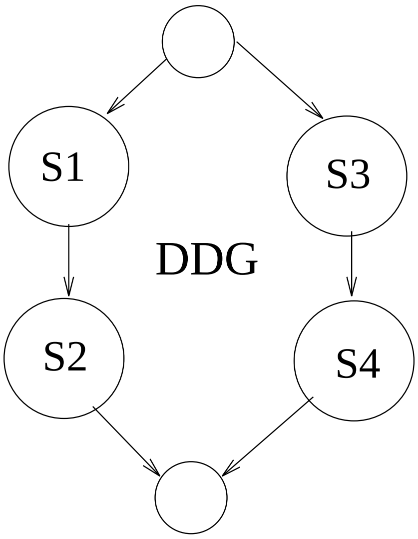


S1: a = b

S2: d = a * c

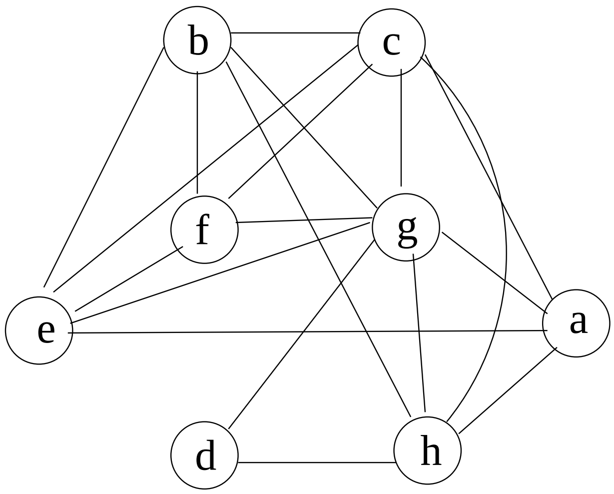
S3: e = f + g

S4: h = e * f

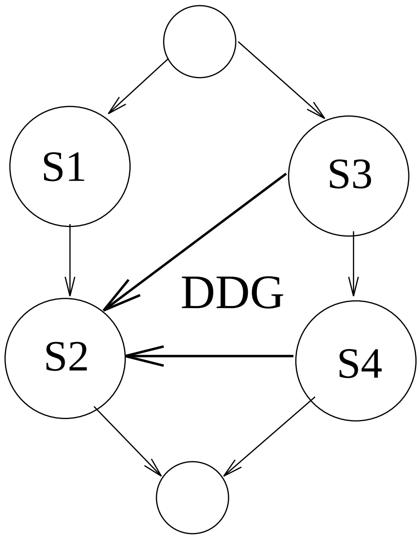


Eliminating Interferences

S1: a = b
S2: d = a * c
S3: e = f + g
S4: h = e * f

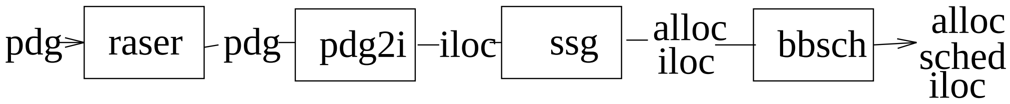


Interference Graph

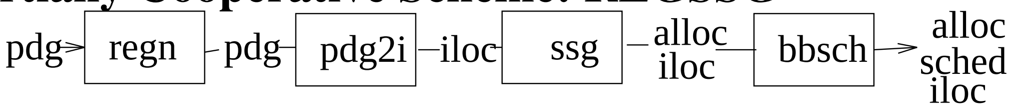


Experimental Study

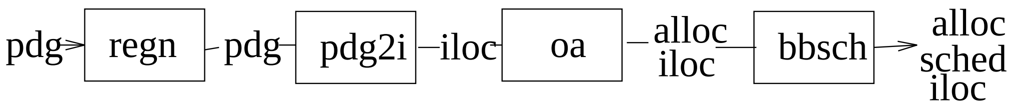
Fully Cooperative Scheme: RASSG



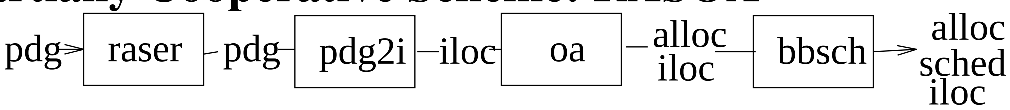
Partially Cooperative Scheme: REGSSG



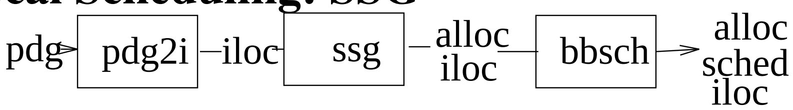
Uncooperative Scheme: REGOA



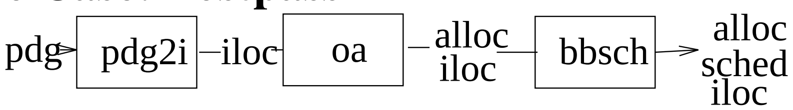
Partially Cooperative Scheme: RASOA



Local Scheduling: SSG



Base Case: Postpass



Benchmarks

- About 30 procedures:
  Clinpack, Stanford routines, Livemore loops

- Simulator for hypothetical machine

- Varied the pipelining: medium pipelined, highly pipelined

- Speedup: cycles(postpass)/cycles(scenario)

Results

- Each scenario performs better than Postpass

- Speedups for REGOA range from .68 to 2.66 for a highly-pipelined machine

- Speedups for RASSG range from .35 to 3.34 for a highly-pipelined machine

- Impact of the cooperative scheduling technique not as great as the cooperative register allocation
  Why? Global scheduler and register allocator operate on different code. Allocator and local scheduler operate on same code.

Future Work

- Expand Experimental Study:
  Bigger Benchmarks
  Real Machine

- Provide cooperation between register allocation and software pipelining on the Program Dependence Graph