Interprocedural register allocation

Allocate registers across calls to minimize overlap between caller and callee subgraph

Allocate global variables to registers over entire program

Could do compile-time interprocedural register allocation

+ gains most benefit
– might be expensive
– might require lots of recompilation after programming change

Or, could do link-time re-allocation

+ low compile-time cost
+ little impact on separate compilation
– cost at link time
– probably less effective

Wall's link-time register allocator

[Wall 86]

Compiler does local allocation + planning for linker

- generates call graph info
- generates variable usage info for each proc
- generates **register actions** executed by linker if variable allocated to register

Linker does interprocedural allocation & patches compiled code

- determines interference graph among variables
- picks best additional variables to allocate to registers
- executes register actions for those vars to patch compiled code

Register actions

Describe changes to code if given var allocated to a register

OPx(var): replace operand x with reg allocated to var
RESULT(var): replace result with reg allocated to var
REMOVE(var): delete instruction if var allocated to a reg
LOAD(var): replace load with move from reg allocated to var
STORE(var): replace store with move to reg allocated to var

Source:

```
w = x = y++ * z;
```

original code	register action(s)
`r1 := load y`	LOAD(y)
`r2 := r1 + 1`	RESULT(y)
`store y := r2`	REMOVE(y)
`r2 := load z`	REMOVE(z)
`r1 := r1 * r2`	OP2(z), RESULT(w)
`store x := r1`	STORE(x), OP1(w)
`store w := r1`	REMOVE(w)

Link-time operations

Construct weighted call graph from compiler tables

- weights can come from static estimates or profile info
- each proc annotated with list of used local vars

Traverse call graph bottom-up, assigning locals to **groups** (a kind of interference graph)

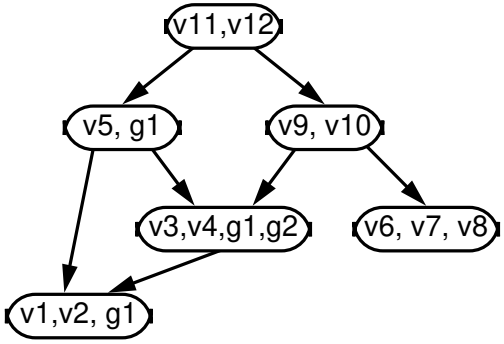
- no simultaneously-live locals in same group
- each global in its own group
- group weighted by sum of members' weights
- recursion & indirect calls pose complications

Allocate groups to registers in decreasing order of weight

Run register actions during code relocation to improve code

Example

Call graph:



Groups:

Possible improvements

Use real profile data to construct weights

Do intraprocedural register allocation at compile-time

Track liveness info var vars at each call site
Track intraprocedural interference graph

Use real interference graph to run link-time allocation

Results

DECWRL Titan RISC processor: 64 registers

Basic experiment:

- local compile-time allocation uses 8 registers
- interprocedural link-time allocator uses 52 registers
- simple static frequency estimates
- smallish benchmark programs

$\Rightarrow$ 10-25% speed-up over local allocation alone

Small improvements (0-6%) with real profile data

Small improvements (0-5%) if use intraprocedural allocation too

- more pronounced for larger, real benchmarks

Less benefit if fewer registers available for global allocation
e.g. 5-20% for 8 global registers

Link-time + local better than intraprocedural register allocation