Improving Resource Utilization of the MIPS R8000 via Post-Scheduling Global Instruction Distribution

Raymond Lo, Sun Chan, Fred Chow, Shin-Ming Liu

Mips Technology Inc.,

Silicon Graphics Computer Systems

Outline

1. Motivations
2. What is Global Instruction Distribution (GID)?
3. MIPS R8000 processor
4. How Does GID Work?
5. Effects of GID
6. Results
7. Conclusion

Motivations

- Exact effects of optimizations are approximate until final code generation.
- Limited compilation time makes post-compilation clean-up phase desirable.
- Superscalar machine performance suffers more seriously due to non-optimal code sequence.

What is GID?

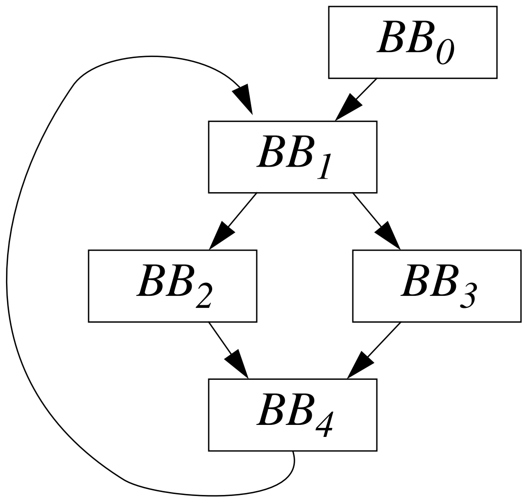
- A framework to re-distribute instructions across basic blocks.
- Moves instructions both forward and backward.
- Does not require profiling feedback.
- Driven by a precise model of R8000.

MIPS R8000 Processor

- 4-issue superscalar
- Functional Units
  - 2 ALUs
  - 2 FPUs
  - 2 Load/Store Units
- Instruction dispatch
  - in-order execution
  - no speculation
  - affected by branch target alignment
- Branch delay slot

How Does GID Work?

Basic Block Level:



Identify *source* and *target* basic blocks pair. Classify movement as (backward):

- BB_4 to BB_1: equivalent.
- BB_2 to BB_1: speculative.
- BB_4 to BB_2 and BB_3: duplicating.
- BB_1 to BB_0 and BB_4: circular.

Instruction Level:

1. Select an instruction that can be moved legally.
2. Determine a window in the *target* basic block for insertion.
3. Compute resource utilization/peephole optimization.
4. Move the instruction and update liveness.

Computing Legality Constraints

1. Instruction type and attributes
   - exceptions
   - control-flow instructions
   - volatile memory references
2. Register dependence
   - input dependence
   - output dependence
   - anti dependence
3. Memory aliases
4. Speculative/Circular motion
   - no store (except some spills)
   - check exposed use

Computing Resource Utilization

Resource usage table in target BB.

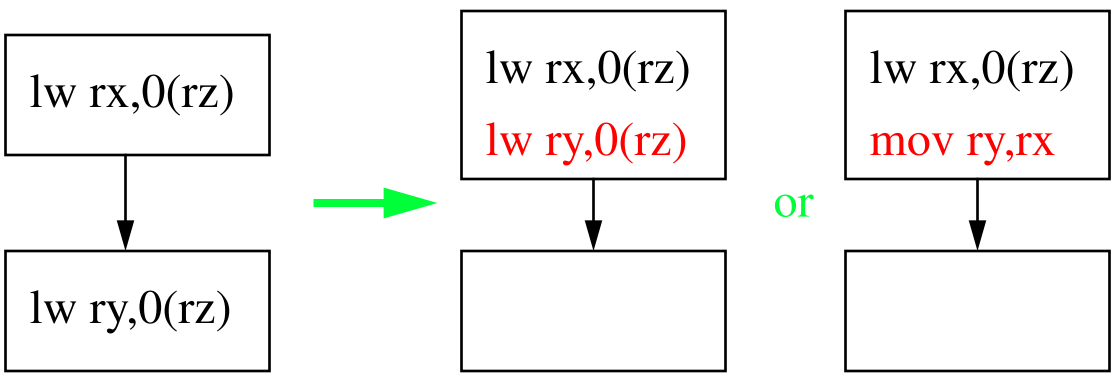
cycle	R0	R1	R2	R3
0	i1	i1		
1	i2	i1	i1	
2		i2		
3		i3	i2	
4			i3	i3

Resources needed by instruction being moved.

0			**X**	
1				X

Resource-Driven Peephole Optimization

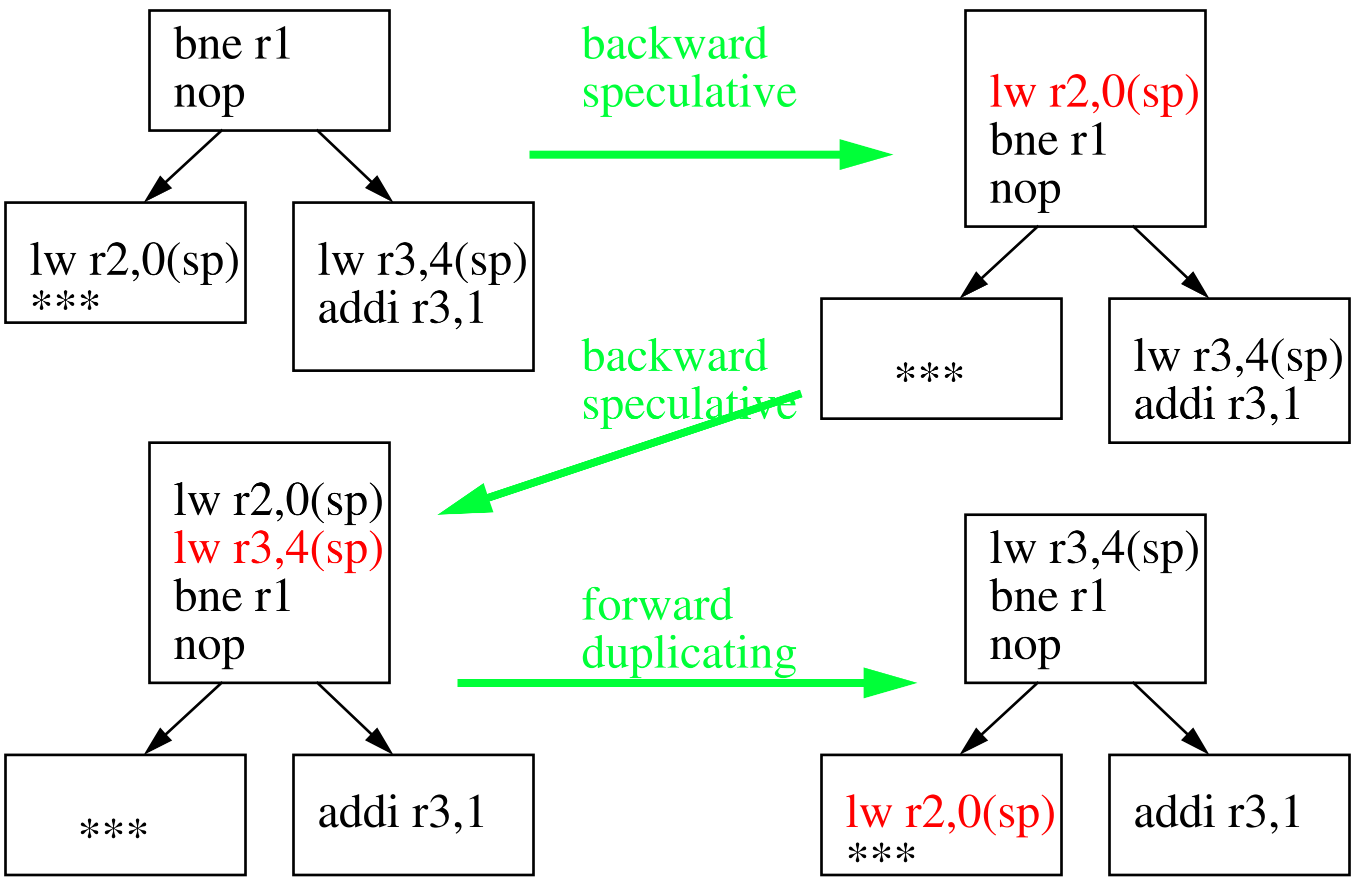
- Convert load into move.
- Delete unnecessary loads.



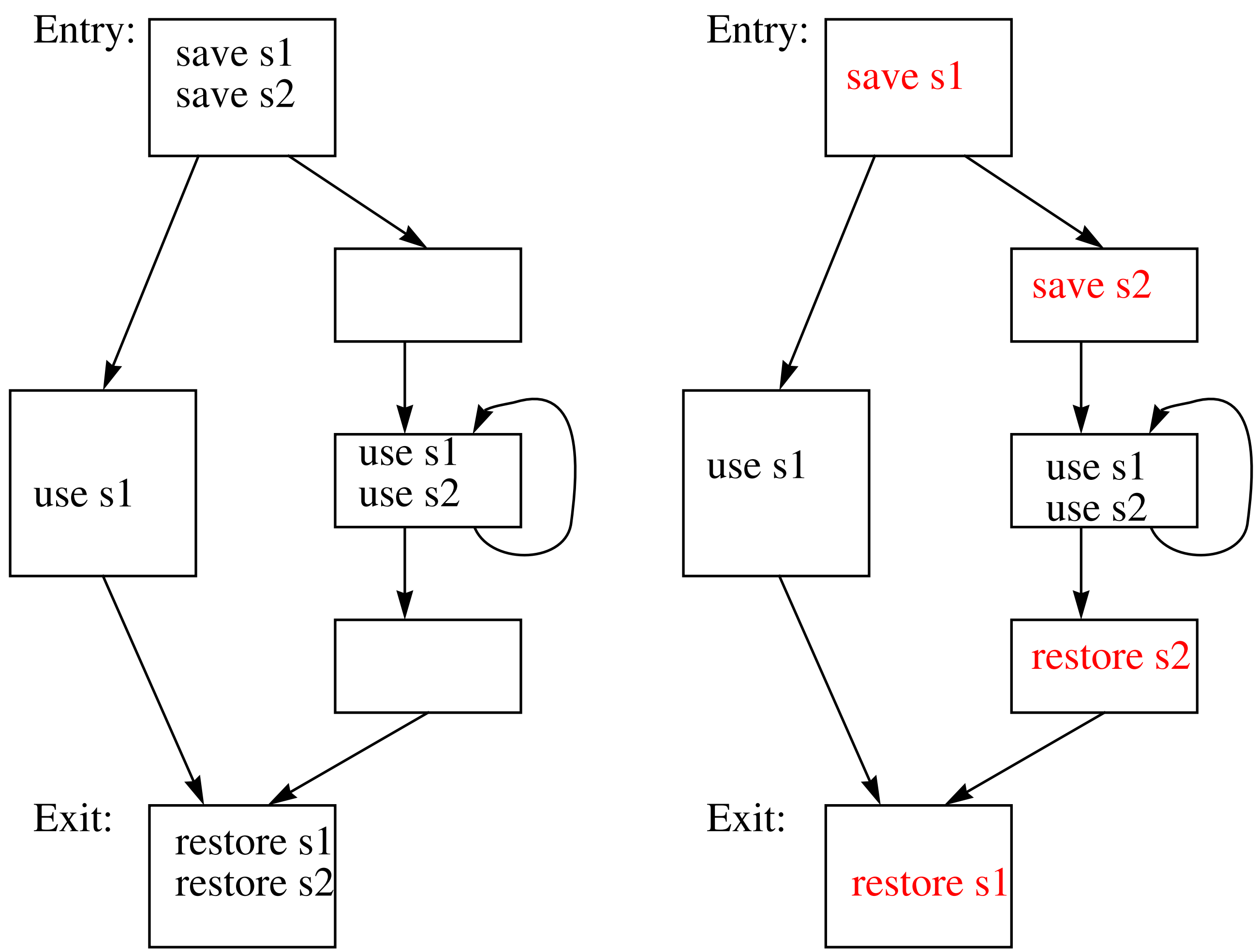
Effects of GID

- More efficient resource utilization.
- Distribute shrink-wrapped saves and restores.
- Fill branch delay slots.
- Align branch targets.

Improve Resource Utilization



Distribute Shrink-Wrapped Registers



Fill Branch Delay Slot

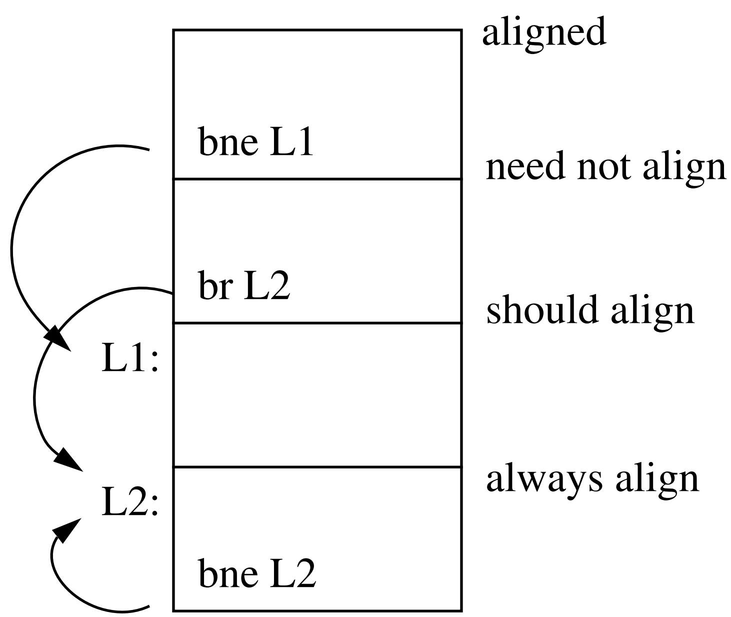
- Delay slot should not be filled if the fall through path is penalized.
- Exception: the fall through is a lower frequency block.

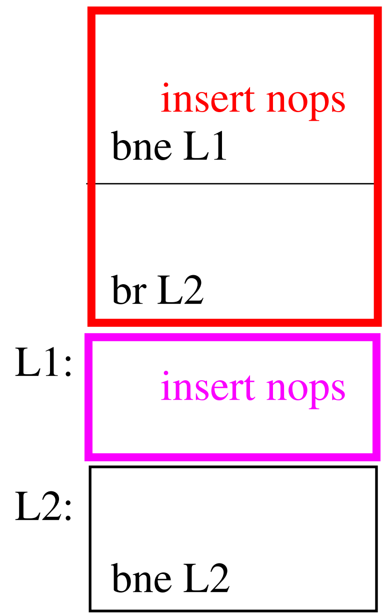
```
    lw r1, 0(r4)        # 0
    bne r0,r1, label    # 1
    nop                 # 1
    lw r2               # 1
    lw r31              # 1
    jr r31              # 2
    lw r3               # 2
label:
    lw r4
```

```
    lw r1, 0(r4)        # 0
    bnel r0,r1, label   # 1
    lw r4               # 1
    lw r2               # 2
    lw r31              # 2
    jr r21              # 3
    lw r3               # 3
label:
```

Align Branch Targets

- Misaligned branch target hurts superscalar performance.
- Achieve alignment by inserting nops.
- Balance potential improvement against code expansion.





Results

	SPEC int	SPEC fp
Runtime improved	6.3%	2.3%
Cycle counts reduced	8.7%	2.5%
Dynamic instruction counts increased	3.1%	1.5%
CPI decreased	11%	4.5%

- More effective for programs with small basic blocks (e.g., li, gcc).
- Less effective for programs that spend most of the time in loops.

Conclusion

- GID is an effective supplement to optimizations done in earlier parts of the compiler.
- Moving instructions globally does not necessarily involve full instruction scheduling.
- GID is a general framework that encompasses most optimizations that can only be done after instruction scheduling.
- Global fine-tuning of resource utilization is necessary for optimal performance in superscalar machines.