Register Allocation Deconstructed

David Ryan Koes

Seth Copen Goldstein

Carnegie Mellon

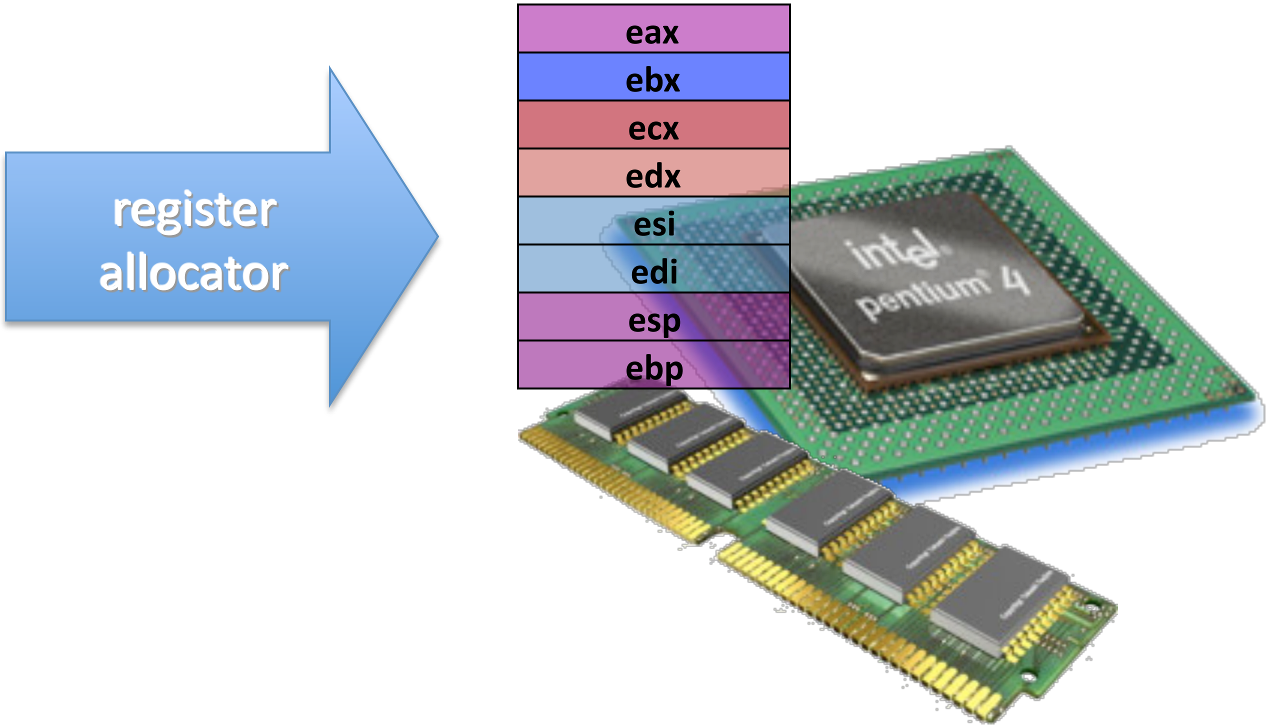
12th International Workshop on Software and Compilers for Embedded Systems
April 24, 2009

Register Allocation Problem

unbounded number of
program variables

```
…
v =   1
w =   v + 3
x =   w + v
u =   v
t =   u + x
print(x);
print(w);
print(t);
print(u);
…
```

limited number of
processor registers +
slow memory

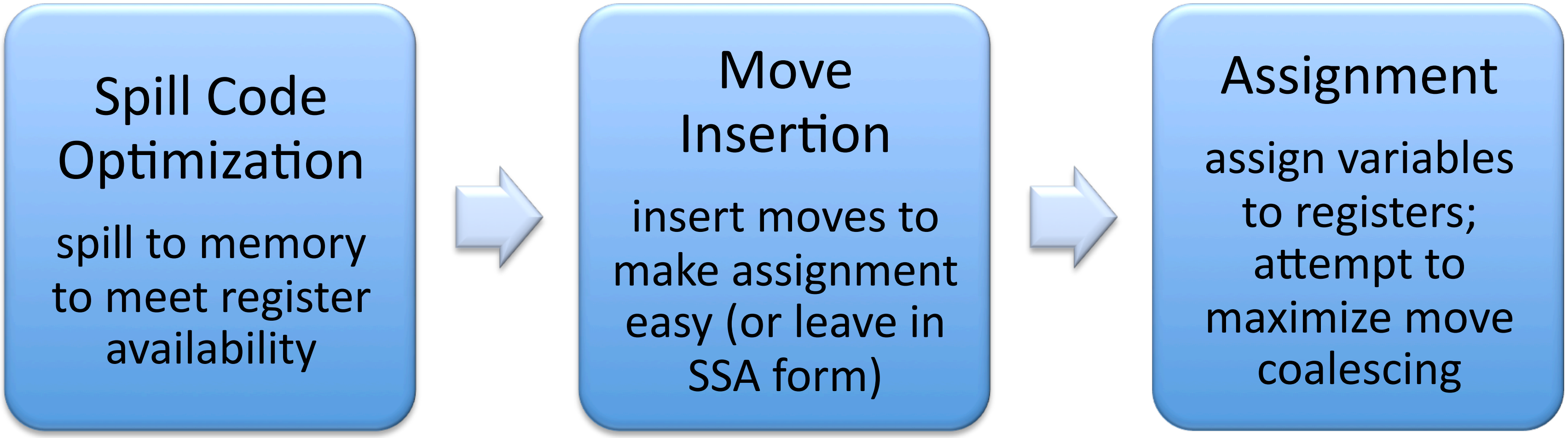


register allocator

eax
ebx
ecx
edx
esi
edi
esp
ebp

Register Allocation

- Graph coloring
- Linear scan
- Optimal frameworks
- “Move elimination” allocators



Spill Code Optimization

spill to memory to meet register availability

Move Insertion

insert moves to make assignment easy (or leave in SSA form)

Assignment

assign variables to registers; attempt to maximize move coalescing

Questions

- What is the penalty of decomposing register allocation into individual components?
- What is the individual impact of each component on code quality?
- How far from optimal are existing heuristics?

Our goal is to answer these questions.

An optimal register allocation framework is used to empirically evaluate the importance of the components of register allocation, the impact of component integration, and the effectiveness of existing heuristics.

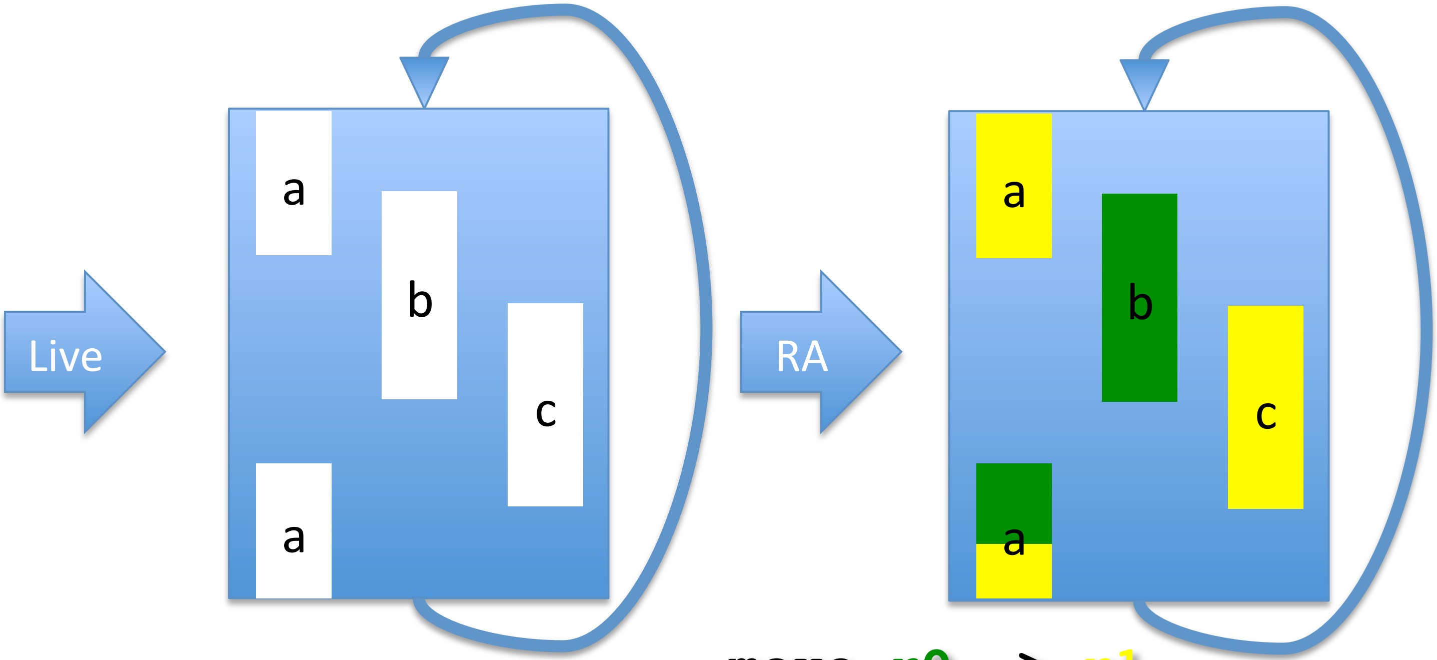
Outline

- Motivation
- Register Allocation Components
  - Move Insertion
  - Coalescing
  - Spilling
  - Assignment
- Methodology
- Results
- Conclusion

Move Insertion

- Additional move instructions can simplify assignment problem
- Can eliminate need to spill
- Only indirect impact on code quality

```
L1:
write  b
read   a
write  c
read   b
write  a
read   c
branch L1
```



move r0 -> r1

Move Insertion Evaluation

full: move instructions may be inserted at any program point

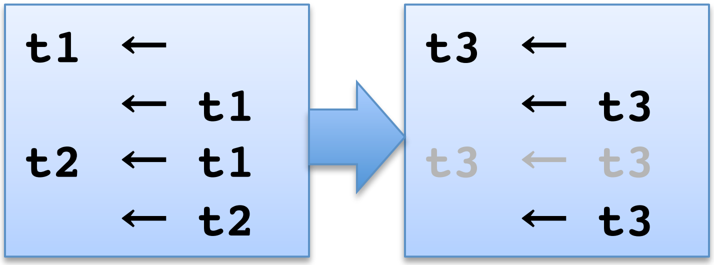
limited: move instructions may be inserted only at the entry and exit of basic blocks

none: no register-to-register move instructions are generated by the allocator

Coalescing

- Eliminate move instructions by assigning each operand to the same location
- Can be performed as separate pass
  - – lose ability to coalesce with physical registers
  - – lose ability to coalesce “uncoalescables”

Pre-alloc Coalescing



```
t1 ←
   ← t1
t2 ← t1
   ← t2
```

```
t3 ←
   ← t3
t3 ← t3
   ← t3
```

Physical Reg

```
t3 ← r0
```

“Uncoalescable”

```
t1 ←
   ← t1
t2 ← t1
   ← t2
   ← t1
```

Coalescing Evaluation

integrated optimal: move coalescing is solved optimally as part of the complete register allocation problem

integrated optimal ignoring uncoalescable: the register allocator fully optimizes only those move instructions identified as coalescable prior to register allocation

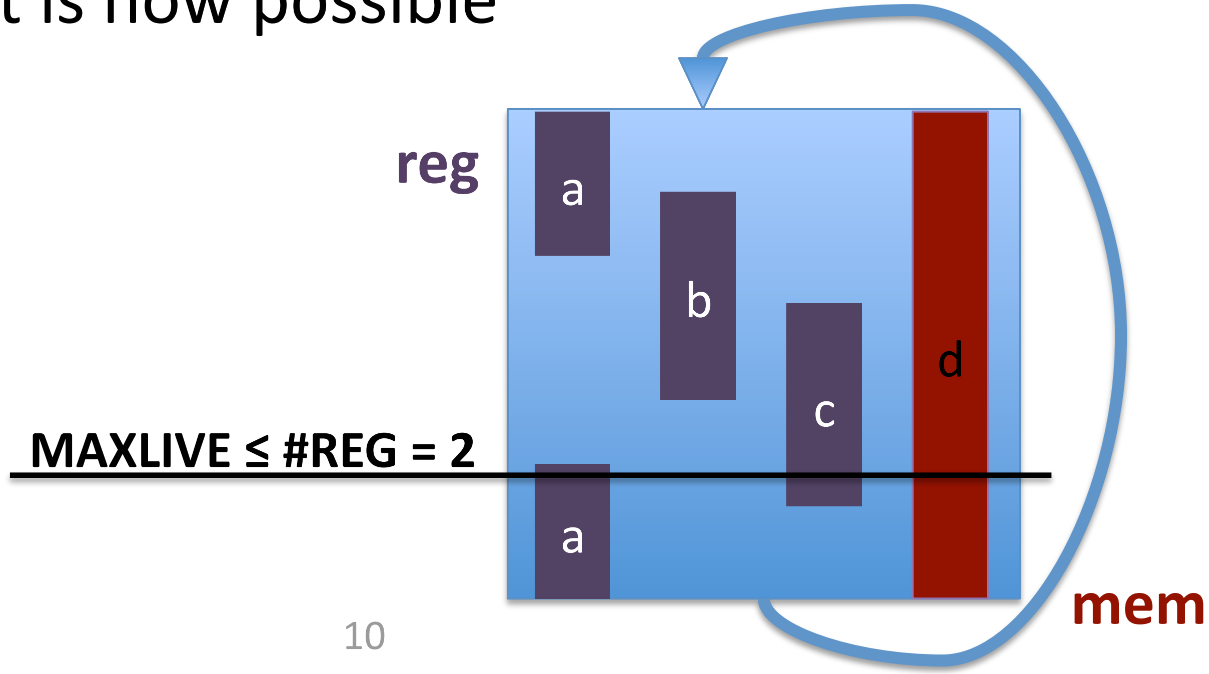
separate optimal: move coalescing is solved optimally as a separate problem prior to allocation

separate aggressive: a greedy heuristic aggressively eliminates coalescable moves prior to register allocation

none: no coalescing is performed

Spilling

- Can be performed as a separate pass
  - spill variables to memory to meet register needs at each program point
  - if move and swap insertions are allowed, assignment is now possible



Spilling Evaluation

integrated optimal: spill code generation is solved optimally as part of the complete register allocation problem

separate optimal: the spill code generation problem (reducing max liveness to meet register availability) is solved optimally as a standalone problem

separate heuristic: the spill code generation problem is solved as a standalone problem using a heuristic algorithm

Assignment

- assign physical register(s) to each variable at every program point
- may change assignment of variable by inserting move instruction
- if spilling and coalescing are performed separately, leaves assignment
- optimizes for register preferences

Assignment Evaluation

integrated optimal: assignment is solved optimally as part of the complete register allocation problem

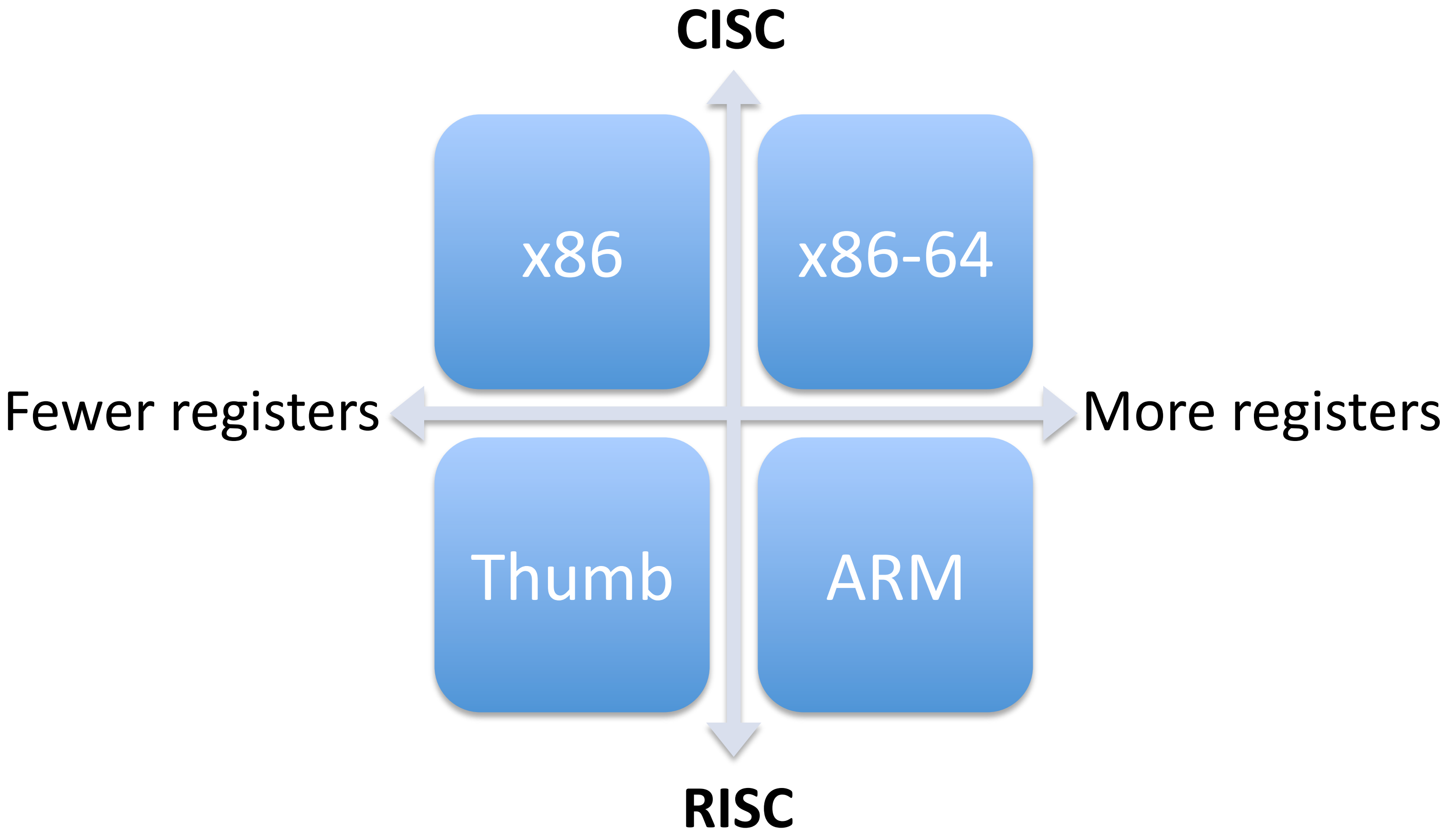
graph heuristic: a graph-coloring based heuristic is used to assign registers to the results of spill code generation; move instructions may be inserted to improve colorability

linear scan heuristic: a linear scan based heuristic is used to assign register to the results of spill code generation; move instructions may be inserted to improve colorability

Methodology

Implement optimal register allocation framework in LLVM 2.4

Consider four target architectures and two code quality metrics



CISC

x86

x86-64

Fewer registers

More registers

Thumb

ARM

RISC

Limitations

- Self-selecting bias in results
  - limited to those functions where an optimal solution can be found in reasonable timeframe
  - however, qualitative results do not appear to change as more time is allowed for optimal allocator
- Implement swap using memory location
- Performance metric necessarily inexact (weighted sum of memory operations)
- Evaluate performance only on desktop processors

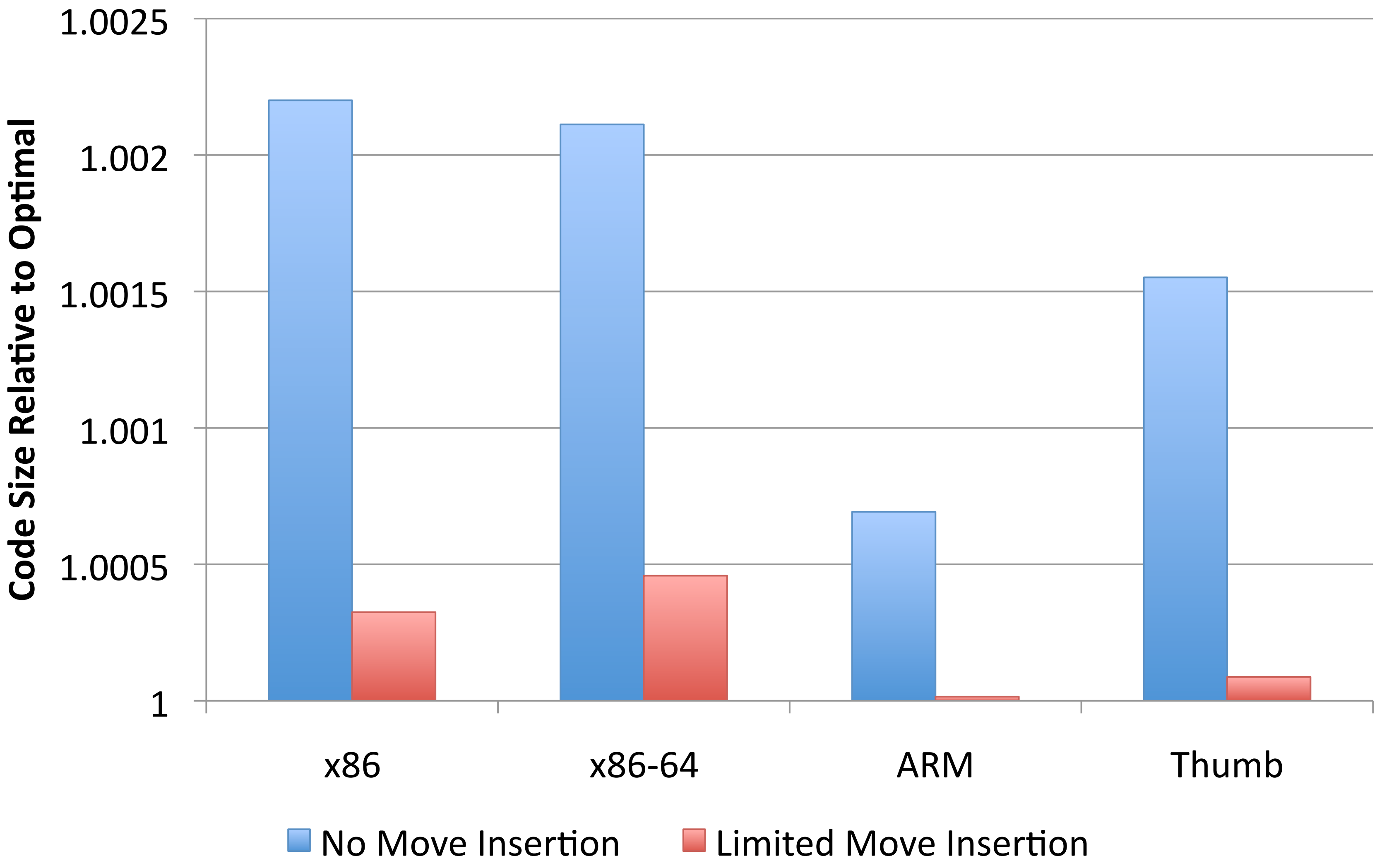
Results: Code Size

- Evaluate subset of Mibench
- Consider all functions where optimal solutions can be found in <10 minutes
  - more than 70% coverage of functions
- Report code size **increase** relative to fully optimal (1.0 best possible result)

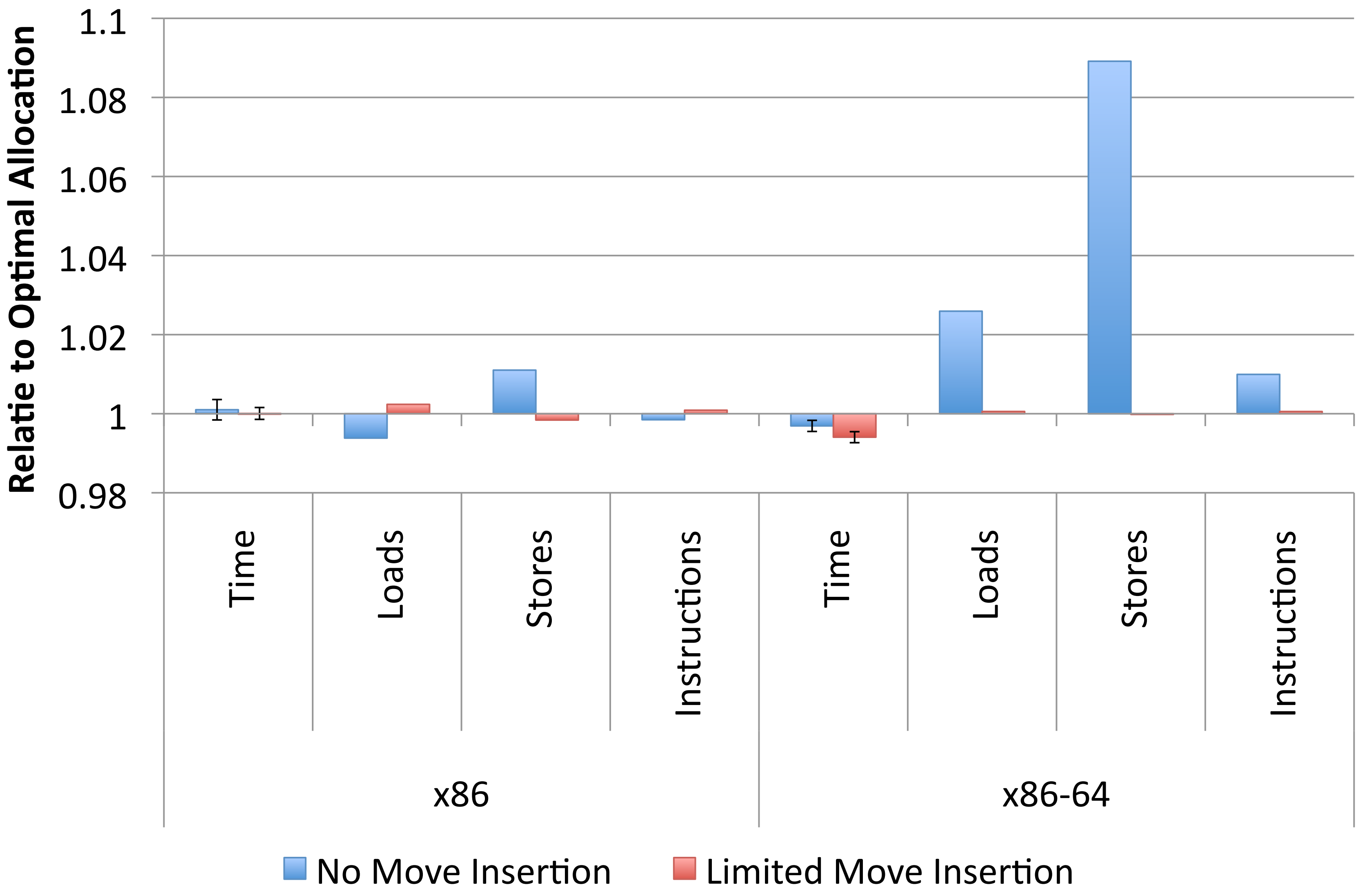
Results: Code Performance

- Evaluate subset of SPEC2006
- Optimize only critical(>85% of running time) functions
- Intel Core 2 Quad (Q6600) @ 2.4GHz
- Report geometric mean relative to fully optimal model
- Possible to do better than optimal due to limitations of metric

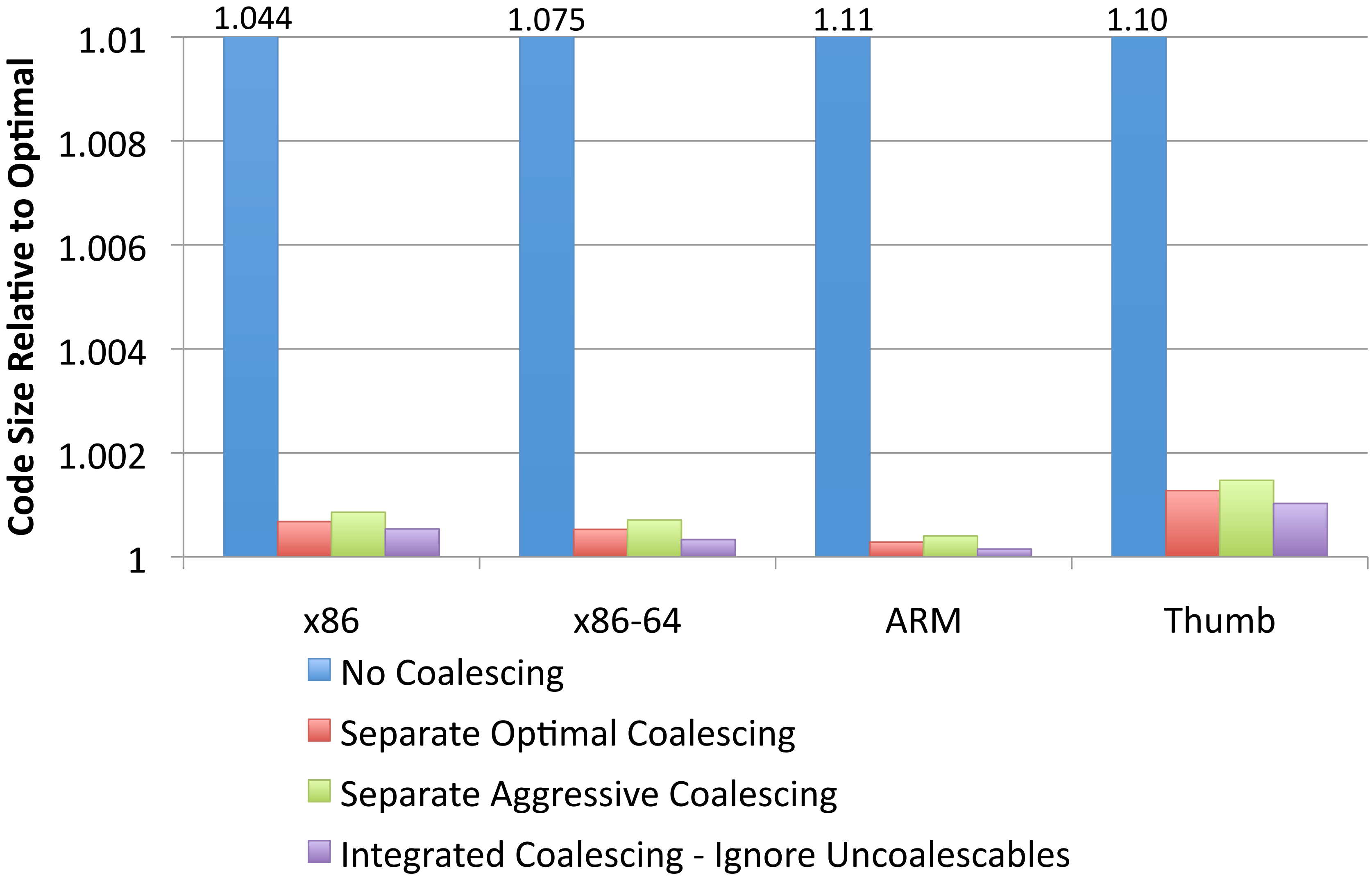
Move Insertion: Code Size



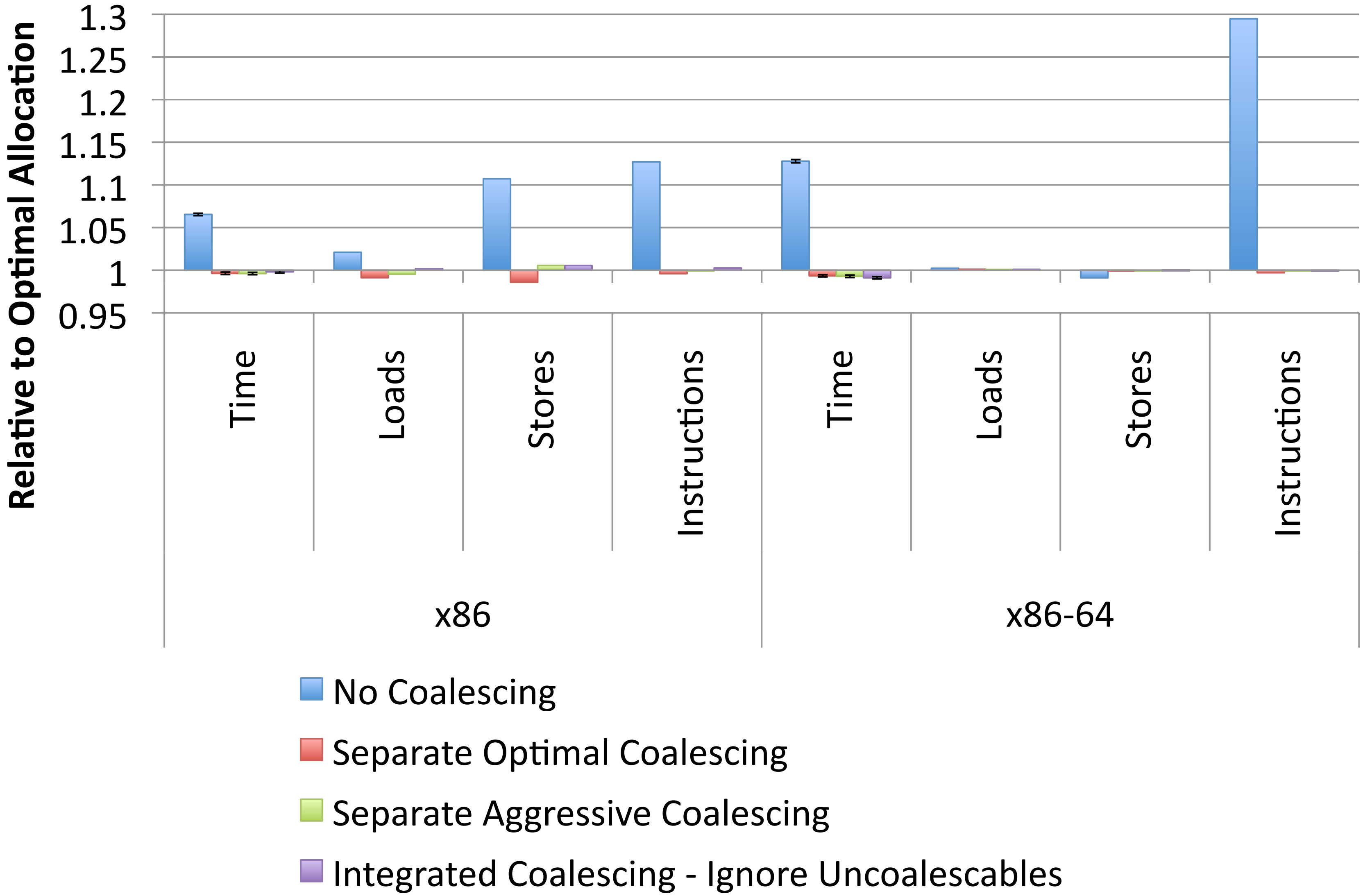
Move Insertion: Code Performance



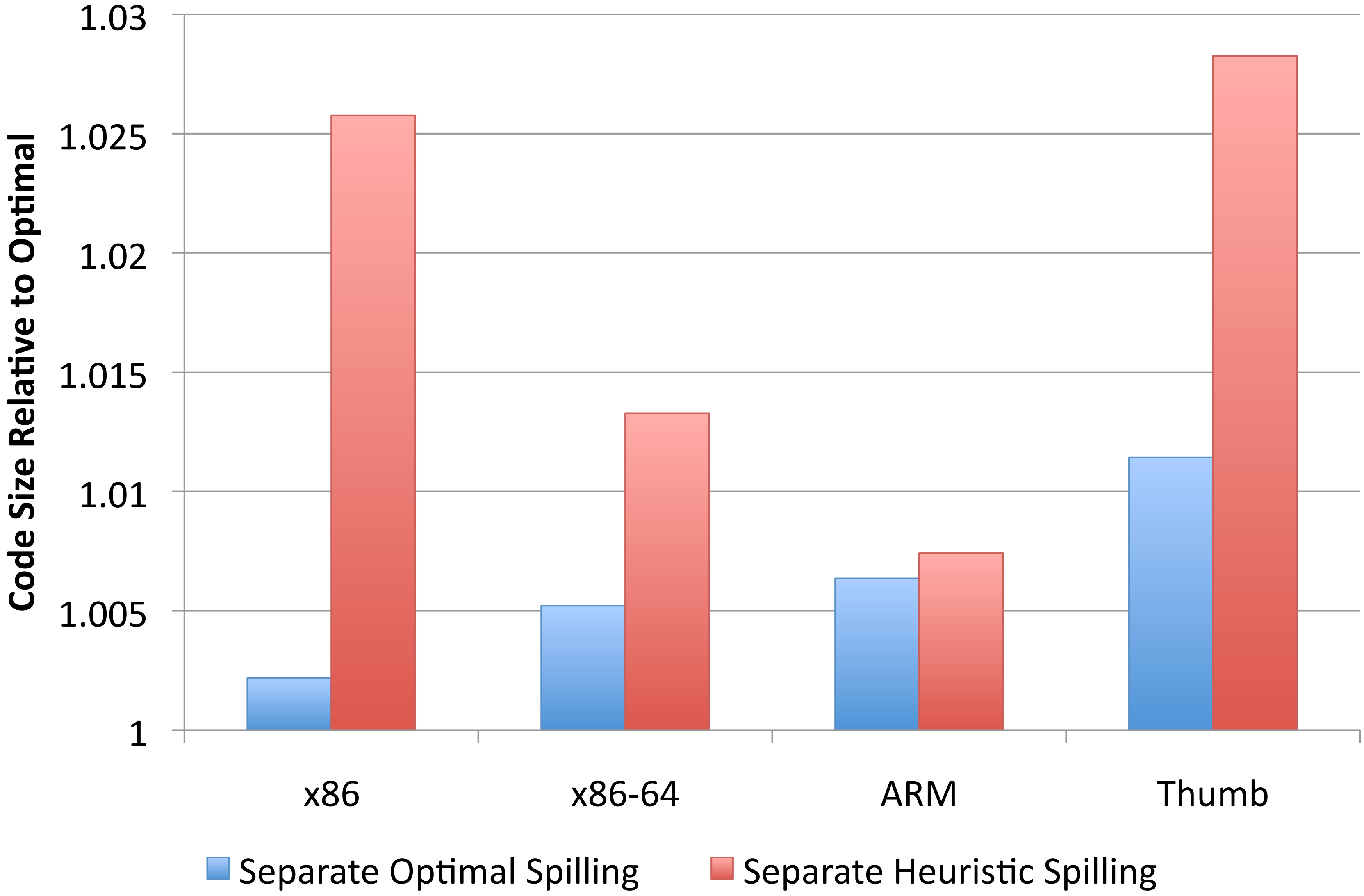
Coalescing: Code Size



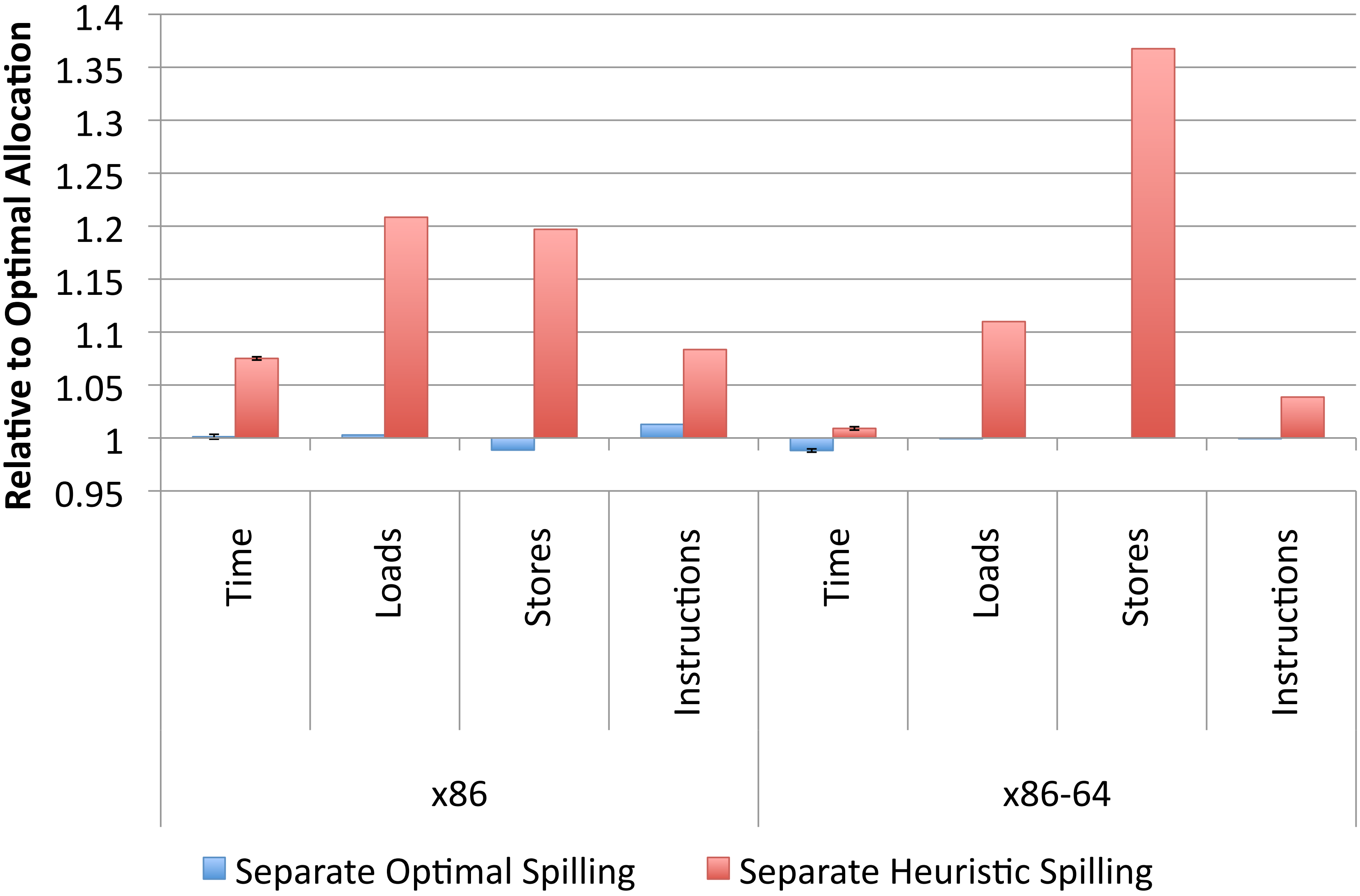
Coalescing: Code Performance



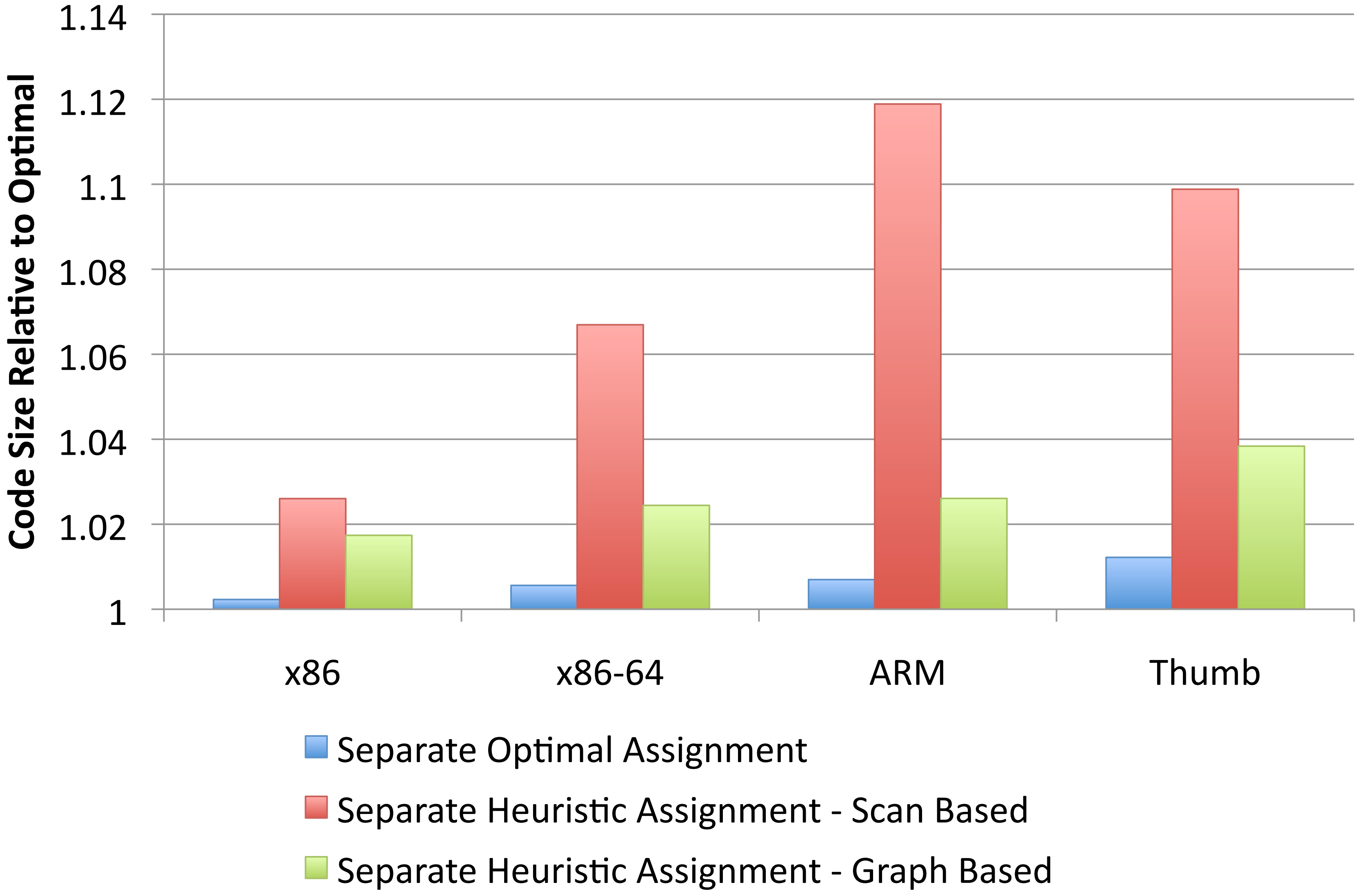
Spilling: Code Size



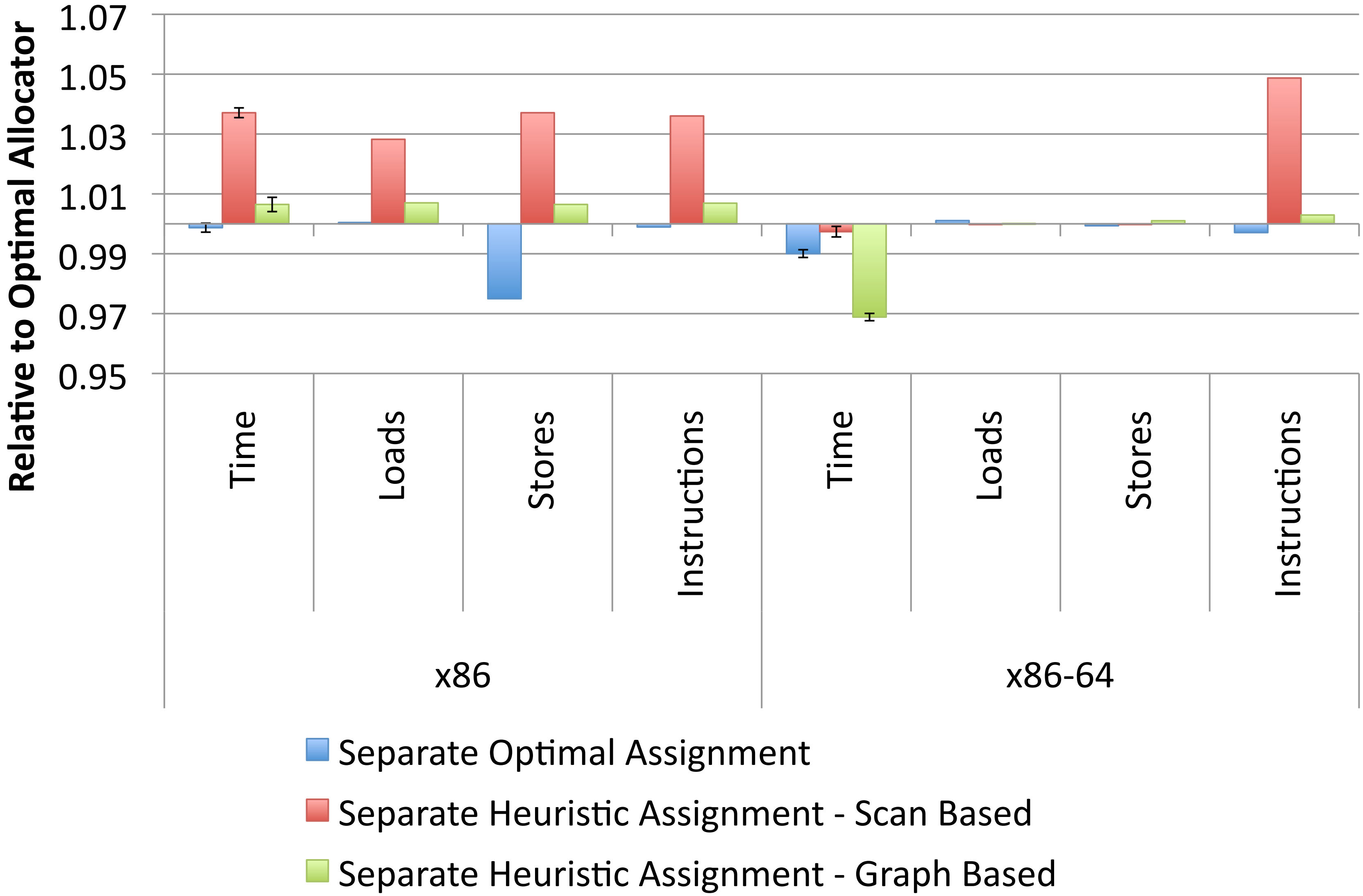
Spilling: Code Performance



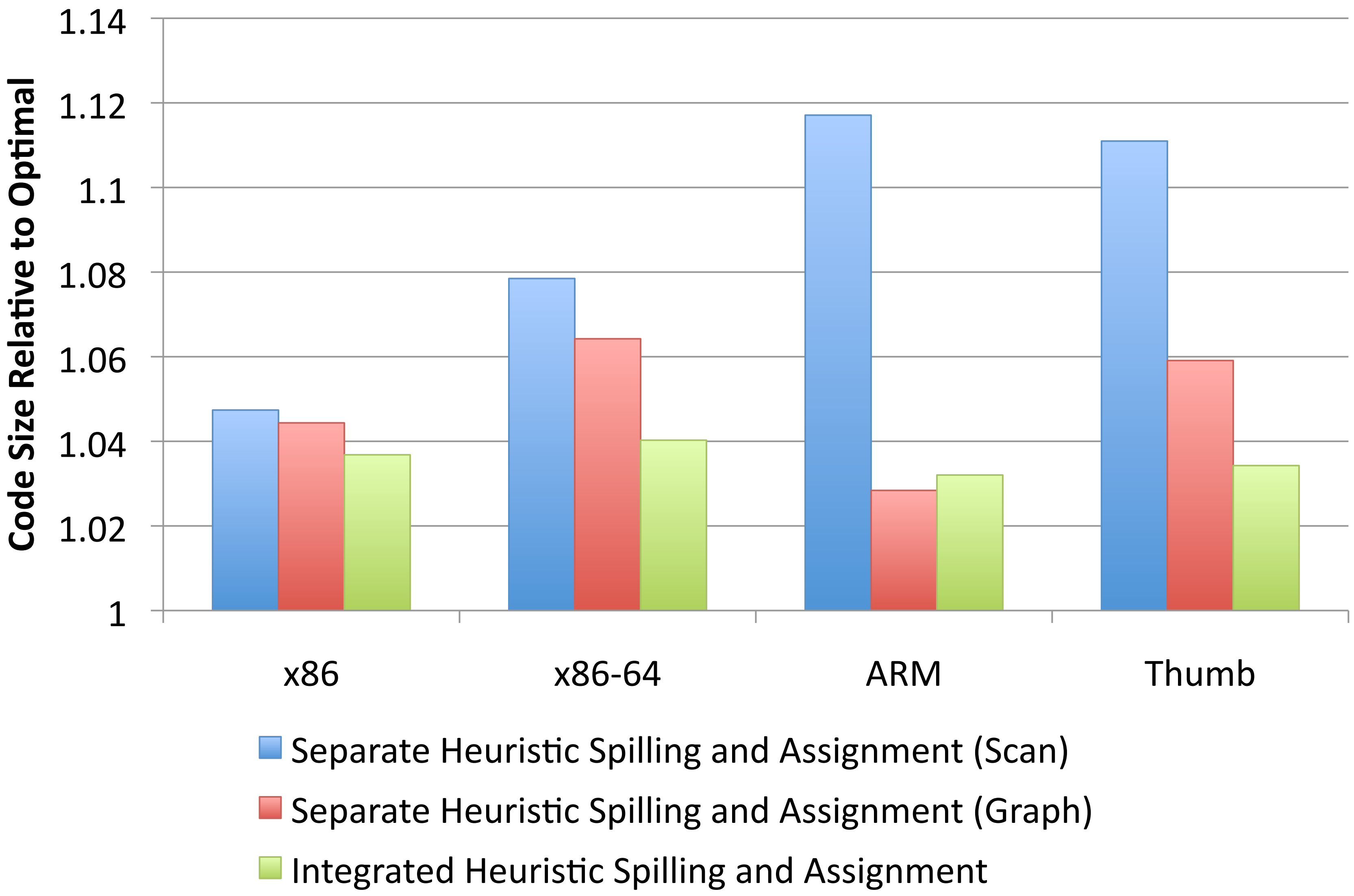
Assignment: Code Size



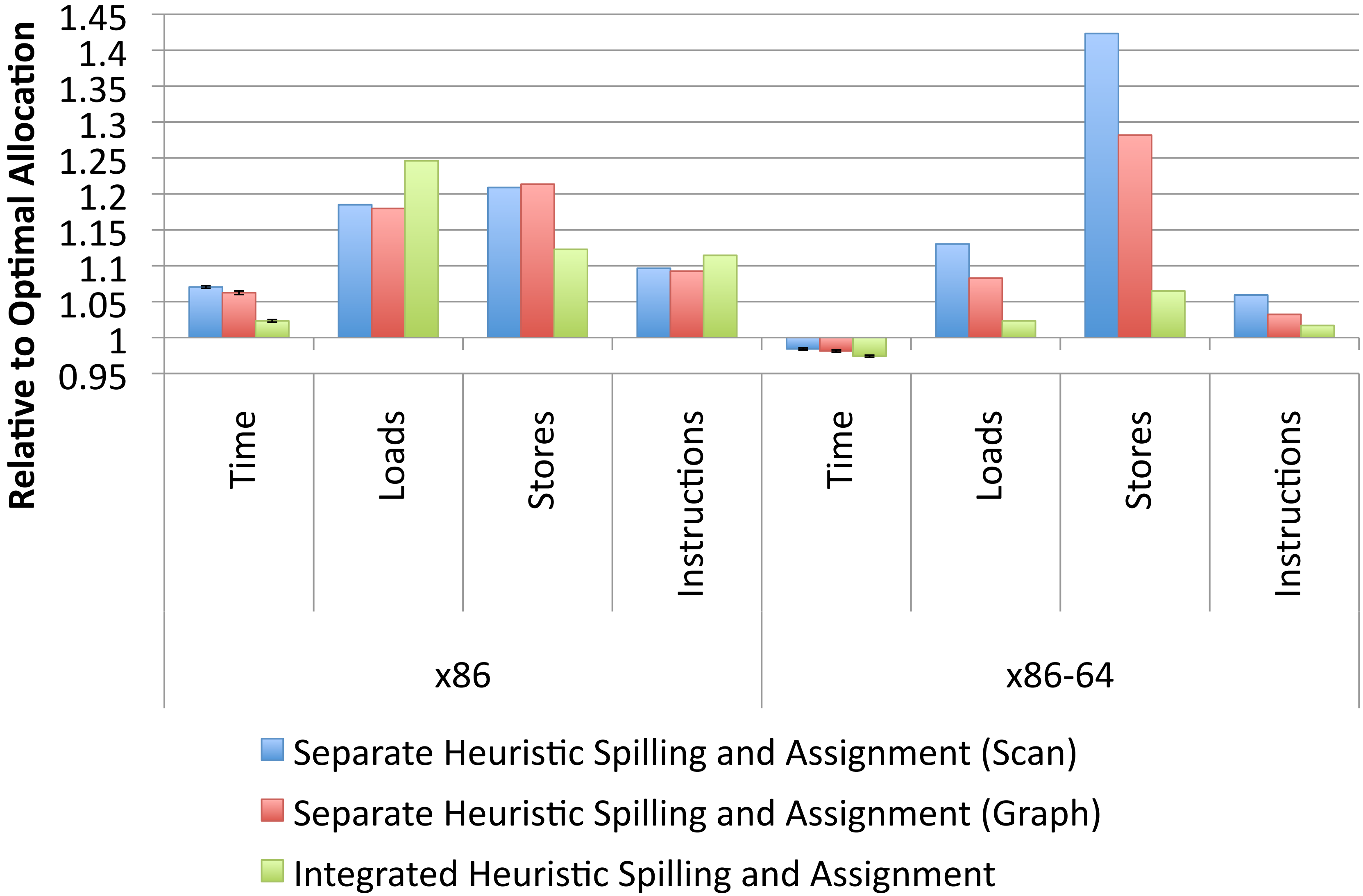
Assignment: Code Performance



Heuristics: Code Size



Heuristics: Code Performance



Conclusions

- When targeting processor **performance**, new register allocator designs should focus on solving **spill code optimization** as the coalescing, move insertion, and register assignment problems are adequately solved using existing heuristics.
- When targeting **code size**, new register allocator designs should focus on solving both the **spill code optimization and register assignment** problems, possibly in an **integrated** framework.