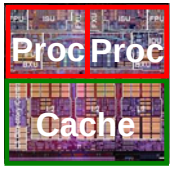
Compiler Optimization of Scalar Value Communication Between Speculative Threads

Antonia Zhai, Christopher B. Colohan,
J. Gregory Steffan and Todd C. Mowry

School of Computer Science
Carnegie Mellon University

Motivation

- Industry is delivering multithreaded processors



IBM Power4 processor:
- 2 processor cores per die
- 4 dies per module

→ 8 64-bit processors per unit

- Improving throughput is straight forward

How can we use multithreaded processors to improve the performance of a single application?

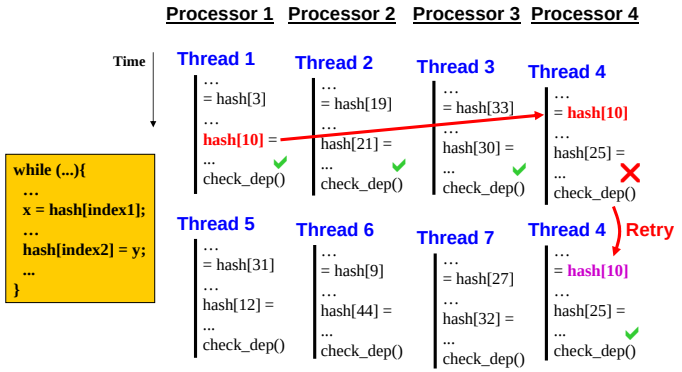
☞ We need parallel programs

Automatic Parallelization

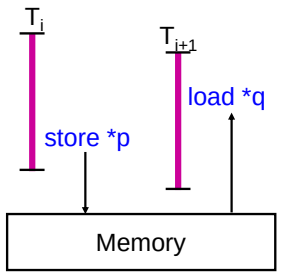
- Finding independent threads from integer programs is limited by
  - Complex control flow
  - Ambiguous data dependences
  - Runtime inputs
- Fundamental problem:
  - Parallelization is determined at compile time

☞ Thread-Level Speculation

Example

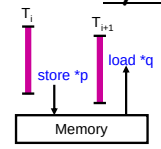


Speculation

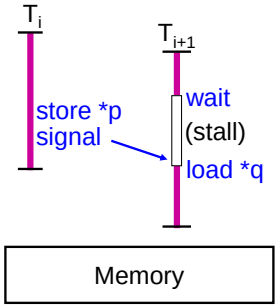


good when p != q

Synchronization (and Forwarding)

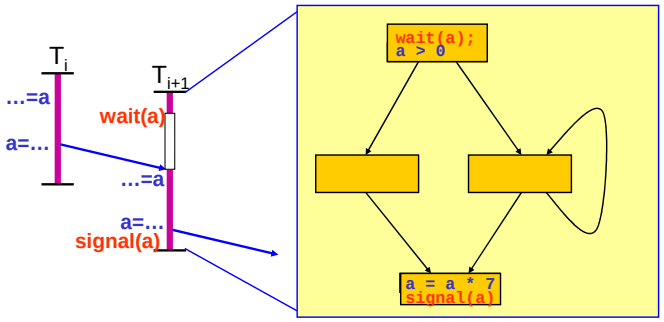


(Speculation)

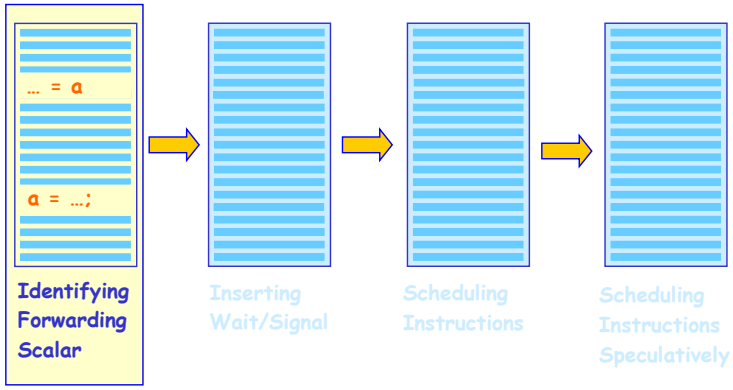


good when p == q

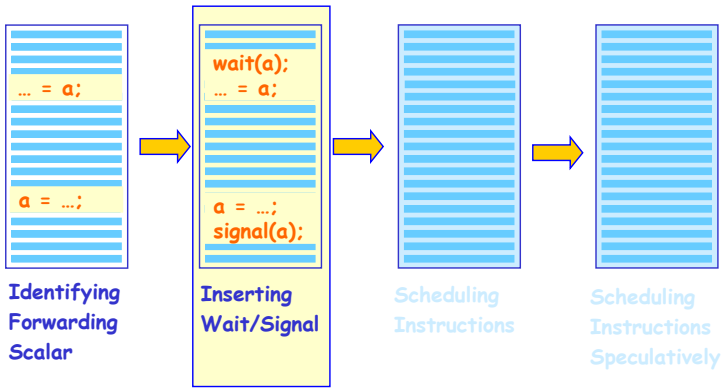
Synchronizing Scalars



Compiler's Tasks

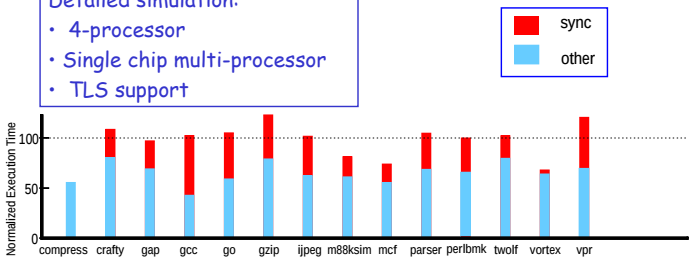


Compiler's Tasks



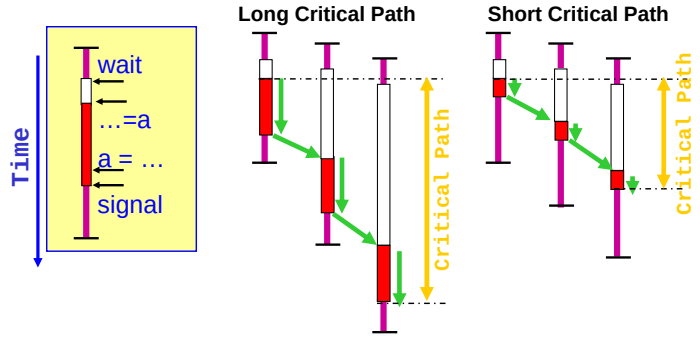
Cost of Synchronization

Detailed simulation:
- 4-processor
- Single chip multi-processor
- TLS support



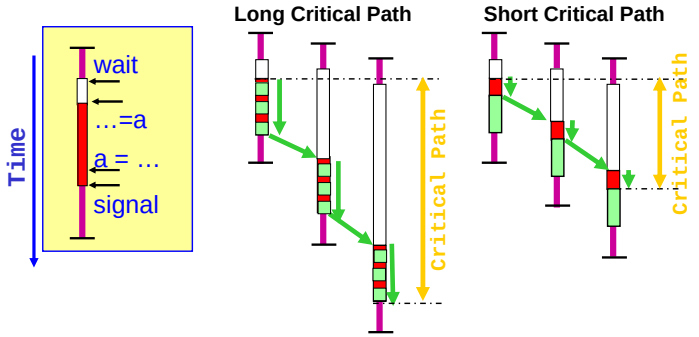
☞ Reducing the synchronization stall has great potential

The Critical Forwarding Path



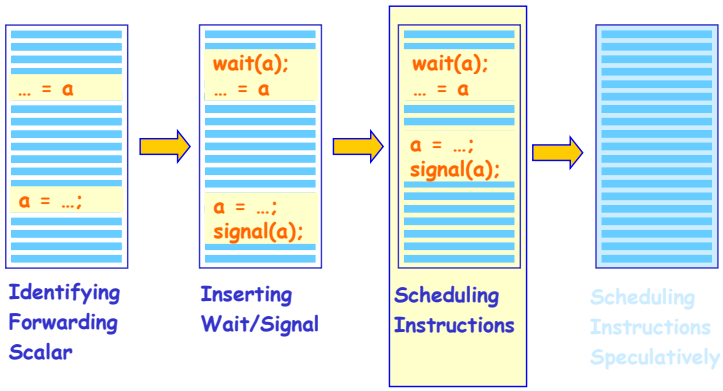
☞ Shorter critical forwarding path → less execution time

Reducing the Critical Forwarding Path



☞ Instruction scheduling can reduce critical forwarding path

Compiler's Tasks



Related Work and Contribution

Related work:

- Multiscalar instruction scheduling [Vijaykumar, Thesis '98]
  - Moving instructions backward one basic block at a time
  - Evaluated under the context of Multiscalar

Our contributions:

- A robust instruction scheduling algorithm
  - Deals with larger threads
  - Handles complex control flow
- Control and data dependence speculation
  - Extends our algorithm to accommodate speculative scheduling
  - Evaluates with detailed simulation
- Comparison with hardware techniques that reduce critical path

Scheduling Instructions

Dataflow analysis

Handles complex control flow

Define two dataflow analyses

☞ Stack

Find the instructions to compute the forwarded value?

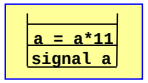
Earliest

Find the earliest node to compute the forwarded value?

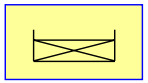
Computation Stack

☞ Stores the instructions to compute a forwarded value

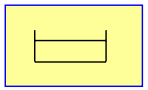
Associating a stack with every node for every forwarded scalar



We know how to compute the forwarded value

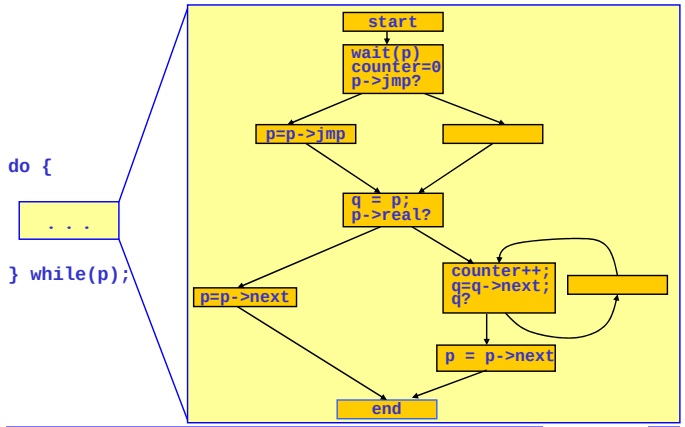


We don't know how to compute the forwarded value

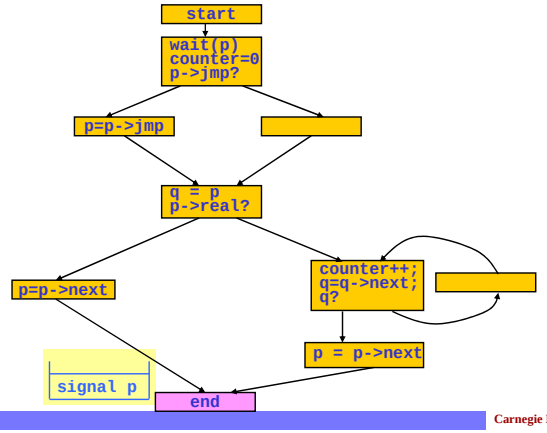


We have not evaluated this node

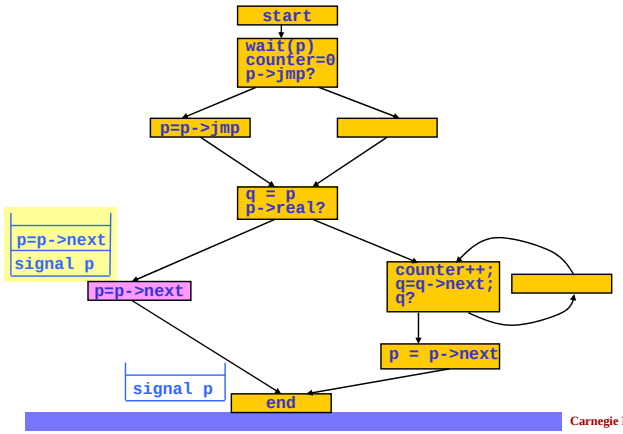
A Simplified Example from GCC



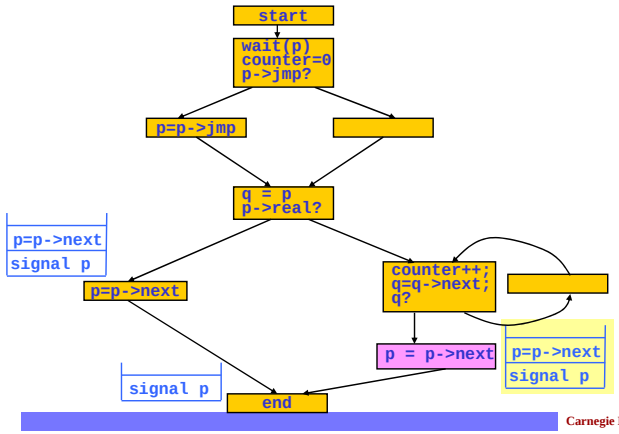
Stack Analysis

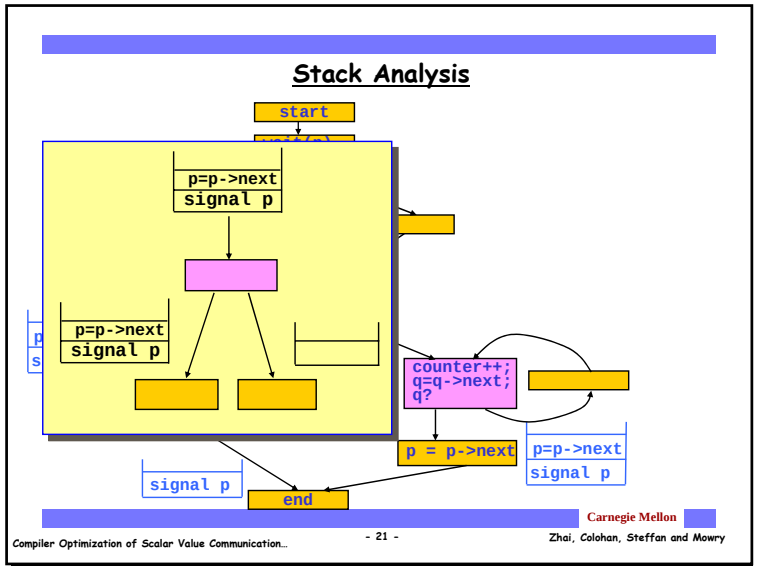


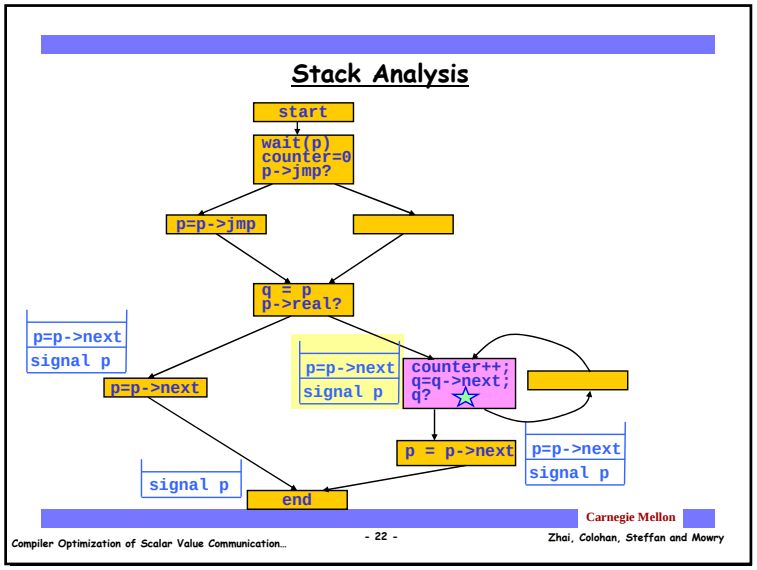
Stack Analysis

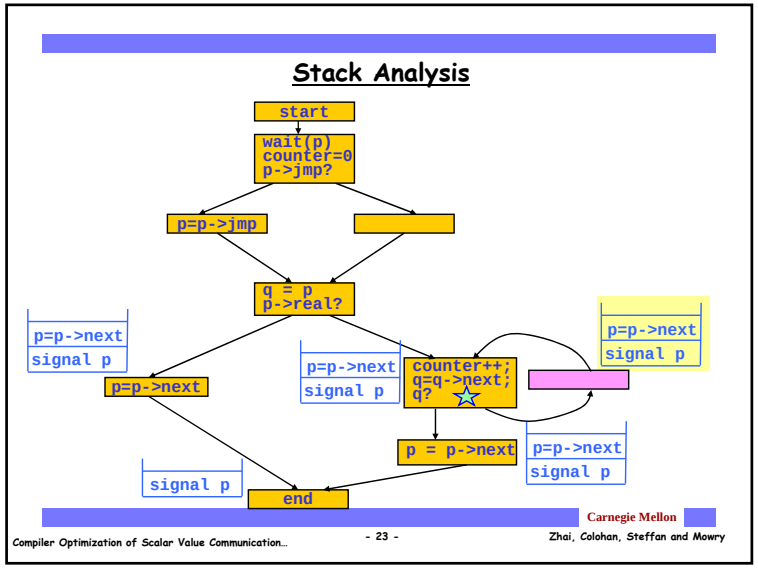


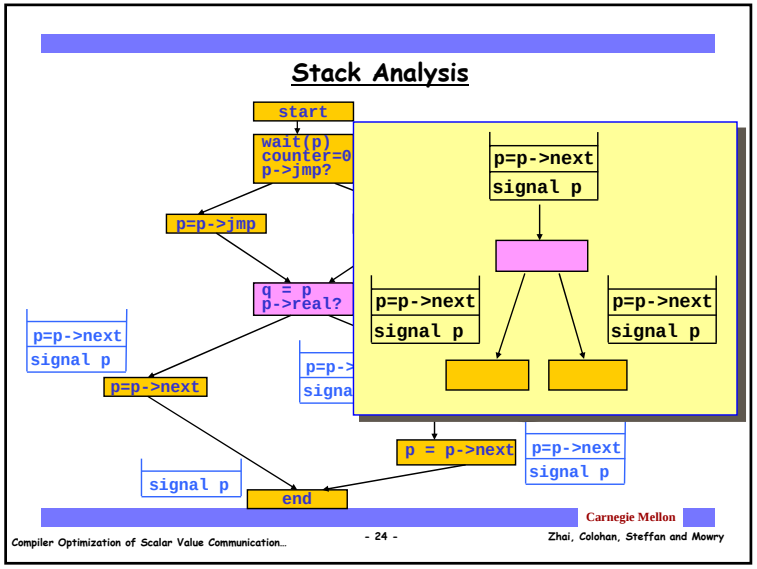
Stack Analysis



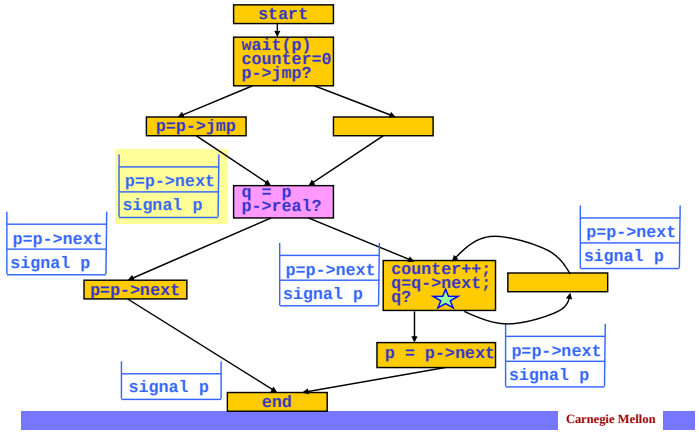

Stack Analysis
start
p=p->next
signal p
p=p->next
signal p
counter++;
q=q->next;
q?
p = p->next
p=p->next
signal p
signal p
end
Carnegie Mellon
Compiler Optimization of Scalar Value Communication…
- 21 -
Zhai, Colohan, Steffan and Mowry


Stack Analysis
start
wait(p)
counter=0
p->jmp?
p=p->jmp
q = p
p->real?
p=p->next
signal p
p=p->next
p=p->next
signal p
counter++;
q=q->next;
q?
p = p->next
p=p->next
signal p
signal p
end
Carnegie Mellon
Compiler Optimization of Scalar Value Communication…
- 22 -
Zhai, Colohan, Steffan and Mowry

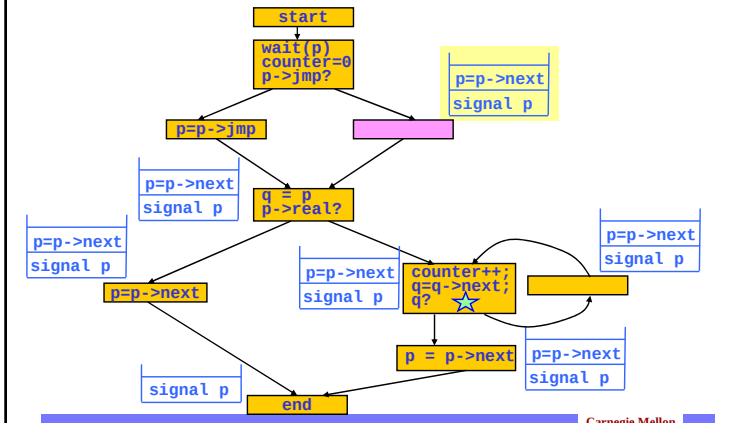

Stack Analysis
start
wait(p)
counter=0
p->jmp?
p=p->jmp
q = p
p->real?
p=p->next
signal p
p=p->next
signal p
p=p->next
signal p
counter++;
q=q->next;
q?
p = p->next
p=p->next
signal p
signal p
end
Carnegie Mellon
Compiler Optimization of Scalar Value Communication…
- 23 -
Zhai, Colohan, Steffan and Mowry


Stack Analysis
start
wait(p)
counter=0
p->jmp?
p=p->jmp
q = p
p->real?
p=p->next
signal p
p=p->next
signal p
p=p->next
signal p
p=p->next
signal p
p = p->next
p=p->next
signal p
signal p
end
Carnegie Mellon
Compiler Optimization of Scalar Value Communication…
- 24 -
Zhai, Colohan, Steffan and Mowry

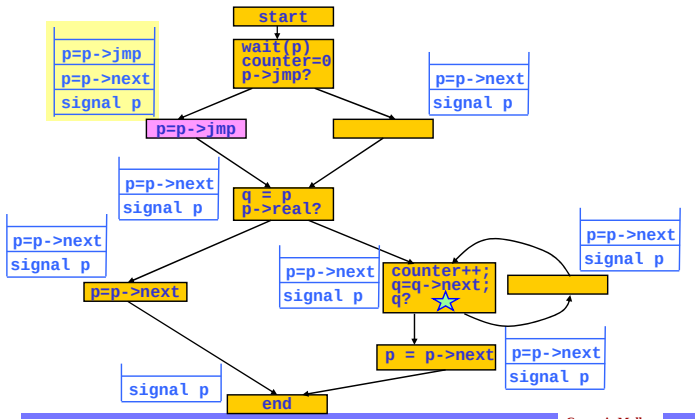
Stack Analysis



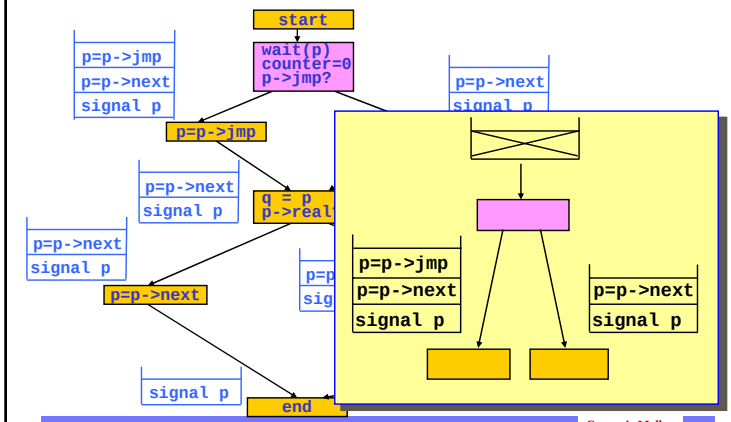
Stack Analysis



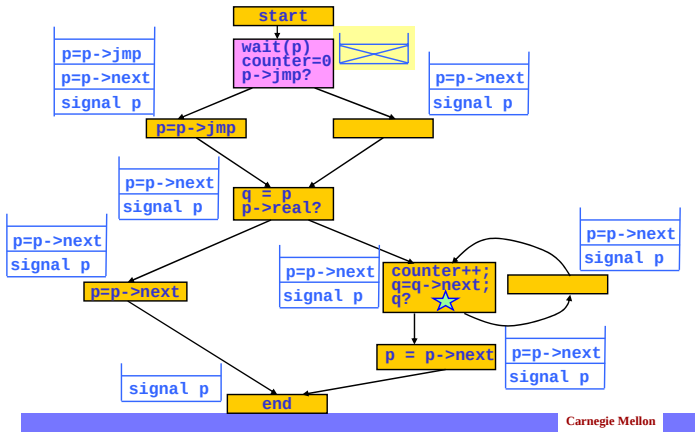
Stack Analysis



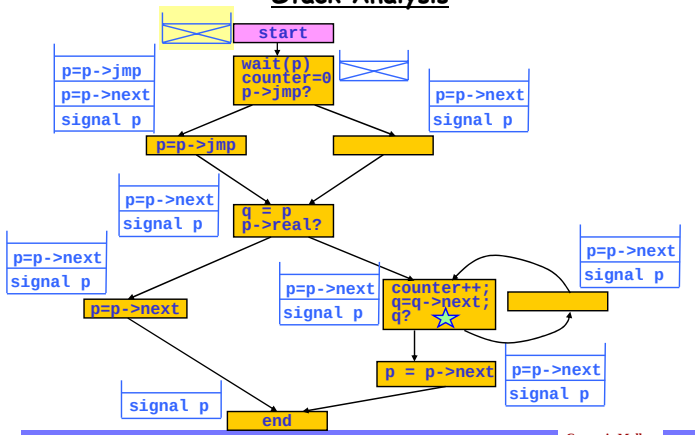
Stack Analysis



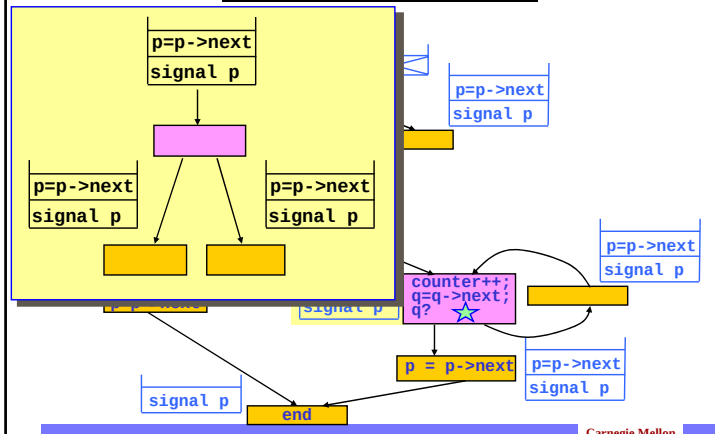
Stack Analysis



Stack Analysis



The Solution is Consistent



Scheduling Instructions

Dataflow analysis

Handles complex control flow

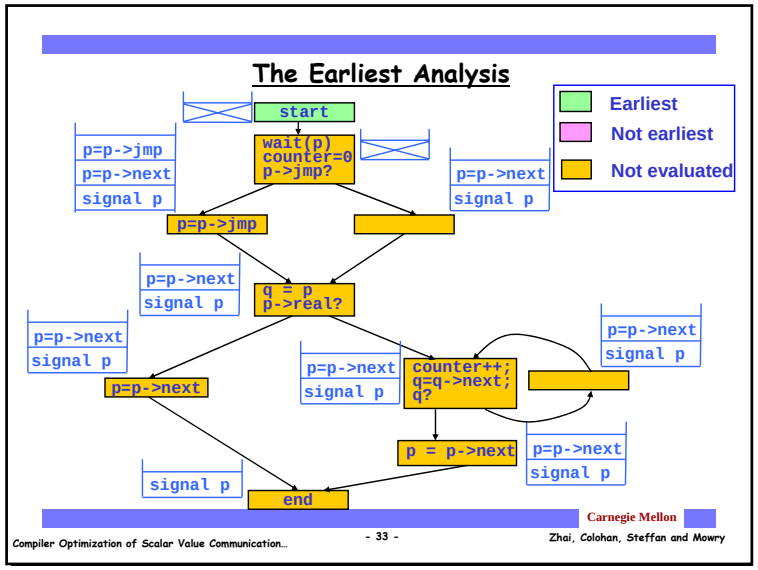
Define two dataflow analyses

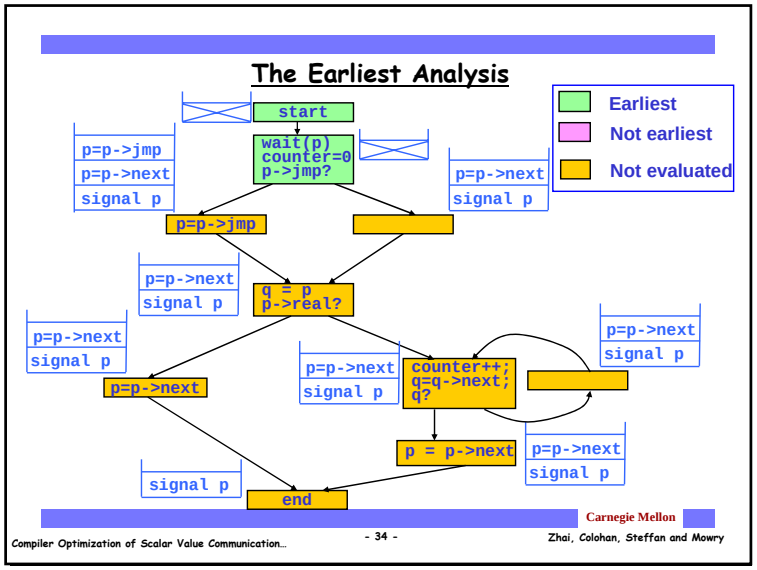
Stack

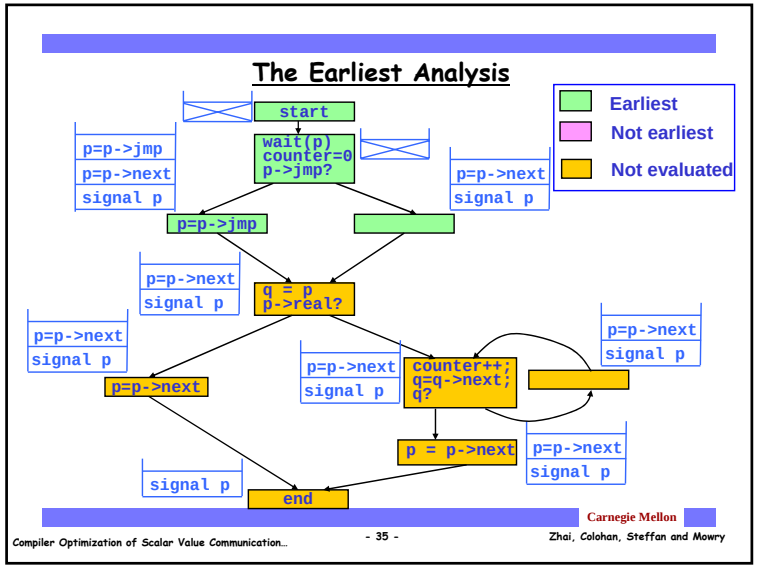
Find the instructions to compute the forwarded value?

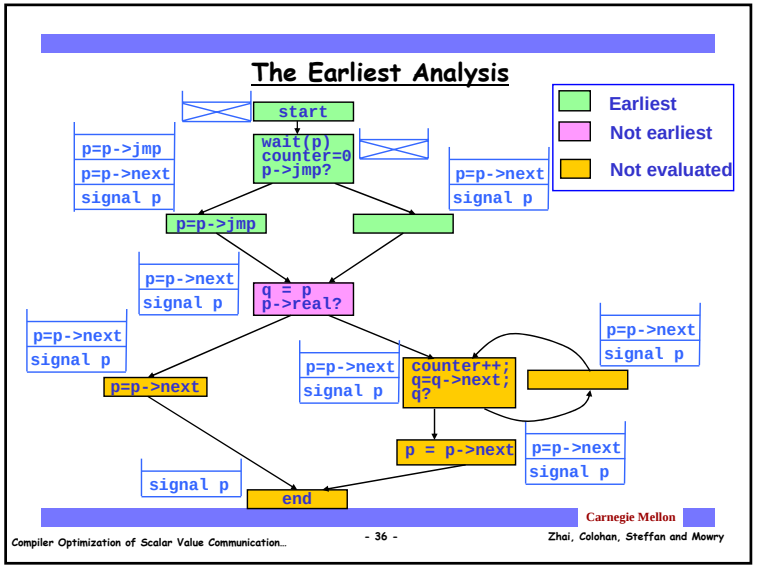
☞ Earliest

Find the earliest node to compute the forwarded value?

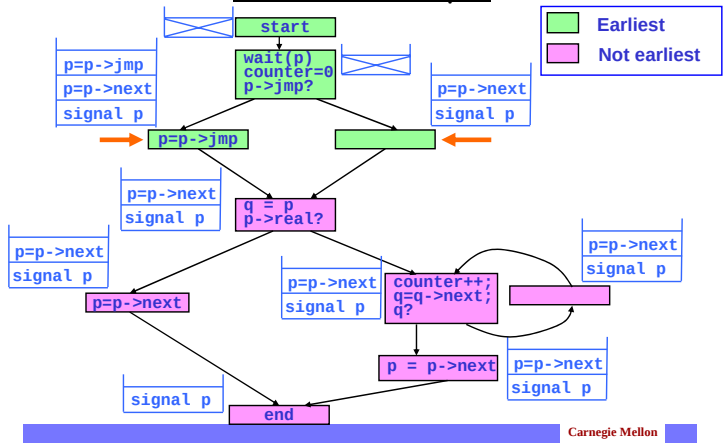

The Earliest Analysis
Earliest
Not earliest
Not evaluated
start
wait(p)
counter=0
p->jmp?
p=p->jmp
p=p->next
signal p
q = p
p->real?
counter++;
q=q->next;
q?
p = p->next
end
Carnegie Mellon
Compiler Optimization of Scalar Value Communication…
- 33 -
Zhai, Colohan, Steffan and Mowry


The Earliest Analysis
Earliest
Not earliest
Not evaluated
start
wait(p)
counter=0
p->jmp?
p=p->jmp
p=p->next
signal p
q = p
p->real?
counter++;
q=q->next;
q?
p = p->next
end
Carnegie Mellon
Compiler Optimization of Scalar Value Communication…
- 34 -
Zhai, Colohan, Steffan and Mowry

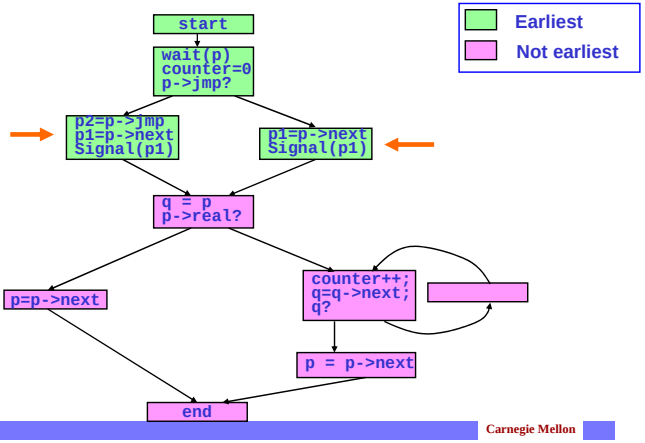

The Earliest Analysis
Earliest
Not earliest
Not evaluated
start
wait(p)
counter=0
p->jmp?
p=p->jmp
p=p->next
signal p
q = p
p->real?
counter++;
q=q->next;
q?
p = p->next
end
Carnegie Mellon
Compiler Optimization of Scalar Value Communication…
- 35 -
Zhai, Colohan, Steffan and Mowry


The Earliest Analysis
Earliest
Not earliest
Not evaluated
start
wait(p)
counter=0
p->jmp?
p=p->jmp
p=p->next
signal p
q = p
p->real?
counter++;
q=q->next;
q?
p = p->next
end
Carnegie Mellon
Compiler Optimization of Scalar Value Communication…
- 36 -
Zhai, Colohan, Steffan and Mowry

The Earliest Analysis



Code Transformation



Experimental Framework

Benchmarks

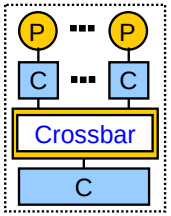
- from SPECint95 and SPECint2000, -O3 optimization

Underlying architecture

- 4-processor, single-chip multiprocessor
- speculation supported by coherence

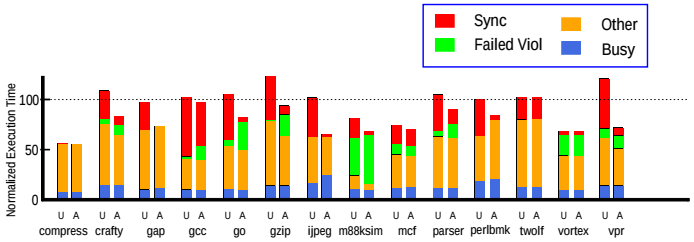
Simulator

- superscalar, similar to **MIPS R10K**
- models all bandwidth and contention



☞ **detailed simulation!**

Instruction Scheduling

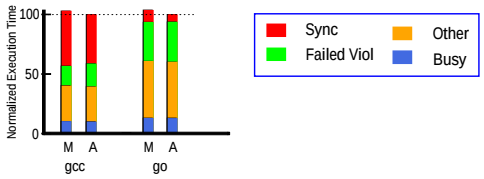


U=No Instruction Scheduling
A=Instruction Scheduling

☞ Improves performance by 18% over no instruction scheduling

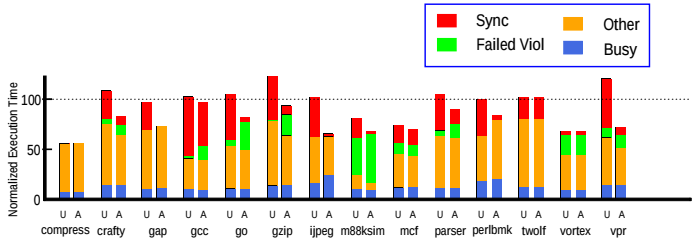
Benefits from Global Analysis

- Multiscalar instruction scheduling [Vijaykumar, Thesis '98]
  - Uses local analysis to schedule instructions across basic blocks
  - Does not allow scheduling of instructions across inner loops



M=Multiscalar Scheduling
A=Our Instruction Scheduling

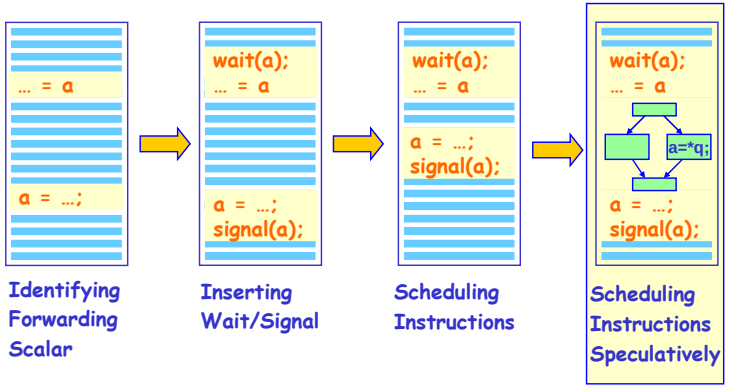
Instruction Scheduling



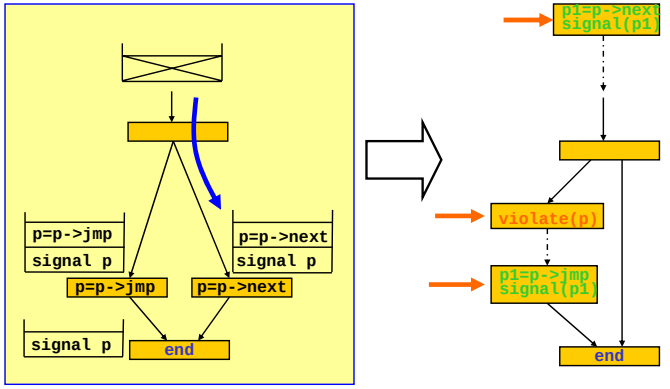
U=No Instruction Scheduling
A=Instruction Scheduling

☞ Improves performance by 18% over no instruction scheduling

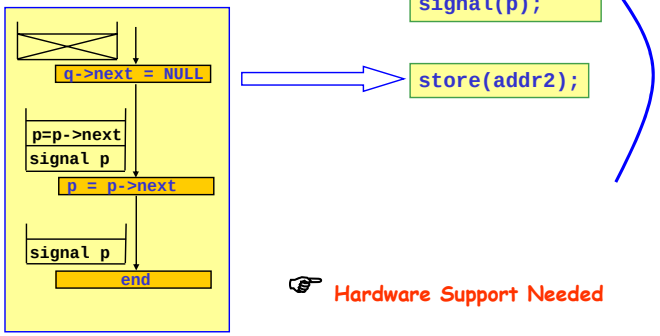
Compiler's Tasks



Speculating Beyond a Control Dependence

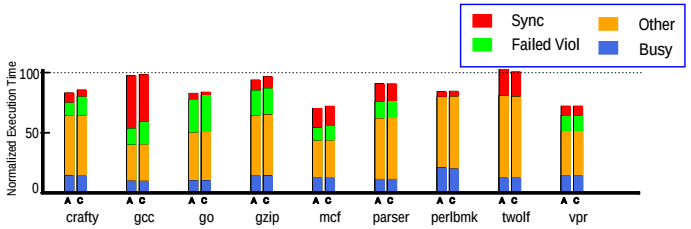


Speculating Beyond a Potential Data Dependence



☞ Hardware Support Needed

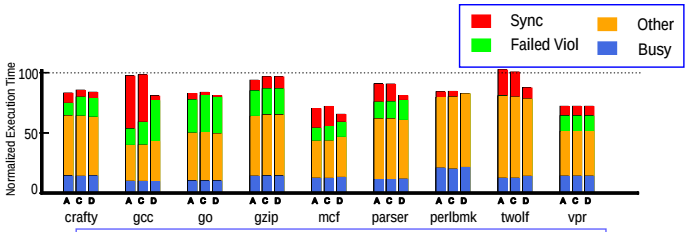
Speculatively Scheduling Instructions Across Control Dependences



A=Instruction Scheduling
C=Speculating Across Control Dependences

☞ No significant performance gain

Aggressively Scheduling Instructions Across Both Control and Data Dependences



A=Instruction Scheduling
C=Speculating Across Control Dependences
D=Speculating Across Control & Data Dependences

☞ Improves performance significantly for some benchmarks

Hardware Optimization to Reduce Synchronization

Hardware optimization techniques [Steffan et al, HPCA'02]

- Avoid synchronization:
  - use the value from a hardware value predictor
- Reduce synchronization stalls:
  - prioritize computation of forwarded value

Hardware optimization impact [Steffan et al, HPCA'02]

- No compiler optimization: Effective
- With compiler optimization: Negligible

Conclusions

Instruction scheduling for reducing synchronization

- Is effective in reducing critical forwarding path
  - Performance improved by 18%
- Is beneficial to handle complex control flow, such as inner loops
  - Improved GCC by 3%
- Gives additional benefit with speculative instruction scheduling
  - Our robust instruction scheduling algorithm can be easily extended to accommodate this
  - One biggest benefactor is GCC, performance improved by 18%
- Reduces the importance of additional hardware optimization

☞ **Critical forwarding path can be addressed by the compiler**