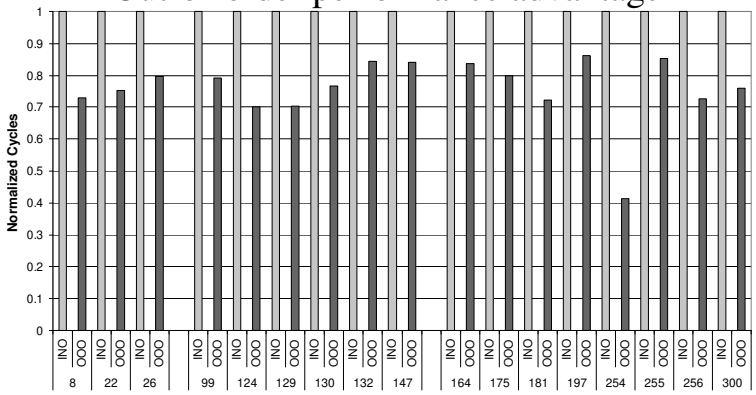
# EPIC's future:
## exploring the space between in- and out- of order

Ronald Barnes, John Sias,
Erik Nystrom and Wen-mei Hwu



3rd Workshop on EPIC Architectures and Compiler Technology March 21, 2004

## Out-of-order performance advantage




- Previous work [WangHPCA7-8] showed significant speedups for EPIC with out-of-order execution (particularly on memory intensive apps)
- Out-of-order execution techniques assumed to benefit primarily from reducing in-order cache miss stalls (esp. tolerance of short misses)

EPIC's future: exploring the space between in- and out- of order
Ronald D. Barnes

# Cost effectiveness of OoO EPIC

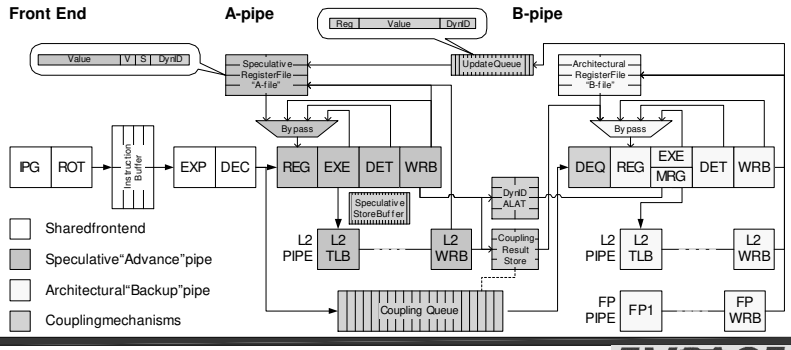
- OoO capabilities come at an increasingly costly expense
  - Power-hungry renaming and scheduling hardware
  - Multiple additional pipeline stages ([WangHPCA7-8] assumed four stages (30% of front-end pipeline)
  - Increasingly speculative instruction wakeup
- Additionally, EPIC features incompatible with OoO
  - Predication—adds complexity to renaming
  - Wide issue/large architected register files—adds complexity to scheduling and renaming
- OoO goes to great expense to capture benefit and provides more general rescheduling capabilities than needed





# Our alternative approach [BarnesMICRO36]

- Observation: even when a cache miss occurs, OoO leaves connected parts of the schedule largely unchanged—only shifted wrt other parts
- Flea-flicker—in-order execution model where not ready instructions are deferred rather than stalling
- First implementation: two-pass pipelining



# Flea-flicker evaluation [BarnesMICRO36]

- 10 benchmarks from SPEC95 and SPEC2000 were evaluated on an **8-wide** EPIC processor
  (*099.go, 129.compress, 130.li, 175.vpr, 181.mcf, 183.equake, 197.parser, 254.gap, 255.vortex, 300.twolf)*
- Benchmarks evaluated on shorted inputs
- Achieved an average of a 55% reduction in load stall cycles
- Average speedup of 1.16 shown for this range of apps
  - Speedups ranged from 0.96 to 1.42
- Not yet achieving speedups expected from dynamic execution, a closer analysis of out-of-order was needed





# Categorizing the benefits of dynamic execution

- Breakdown of OoO performance
- Memory tolerance benefits
  - Hiding near (L1/L2) misses
  - Starting *positionally* blocked far misses early
- Non-memory-tolerance benefits
  - Extracts parallelism across branches
  - Dynamic data speculation

| Latency tolerance | Scheduling |
|---|---|
| Hiding near (L1/L2) misses | Control speculation |
| Overlapping far misses | Data speculation |

# EPIC experimental configuration

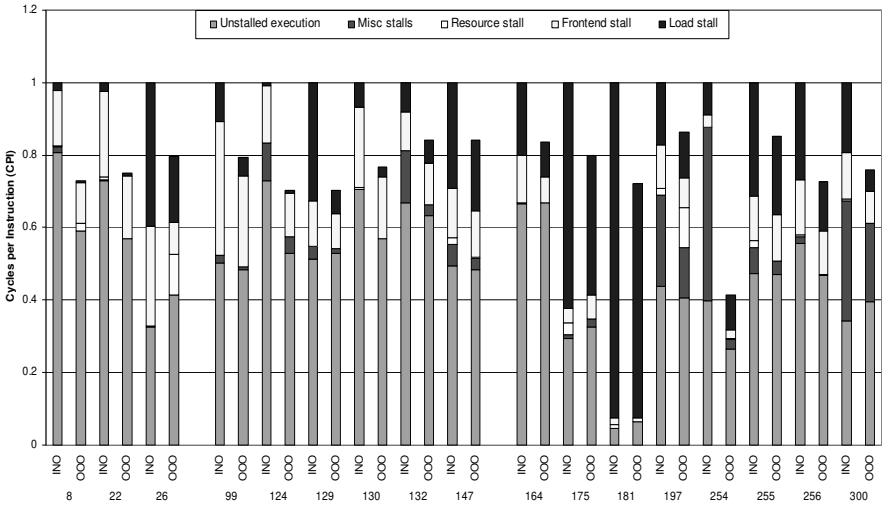
| Feature | Parameters |
|---|---|
| Functional Units | 6-issue, 2LD, 2ST, (Itanium II FU distribution) |
| L1I Cache<br>L1D Cache<br>L2D Cache<br>L3D Cache<br>Main Memory | 1 cycles, 16 KB, 4-way, 64 B lines<br>1 cycles, 16 KB, 4-way, 64 B lines<br>5 cycles, 256 KB, 4-way, 128 B lines<br>12 cycles, 1.5 MB, 12-way, 128 B lines<br>145 cycles |
| Data Model | ILP32 |
| Inputs | Reference (Sampled full execution) |
| Ideal OoO Model | 128-entry scheduling window<br>256-entry ROB/256 Physical Registers<br>Perfect latency prediction<br>No additional pipeline stages |

- Benchmarks compiled with IMPACT—Hyperblock and modulo sched.
- IMPACT EPIC instruction set—complex-addressing-mode memory operations and single instruction very-long-latency operations (unlike IPF)
- FP accesses cached in L1D Cache

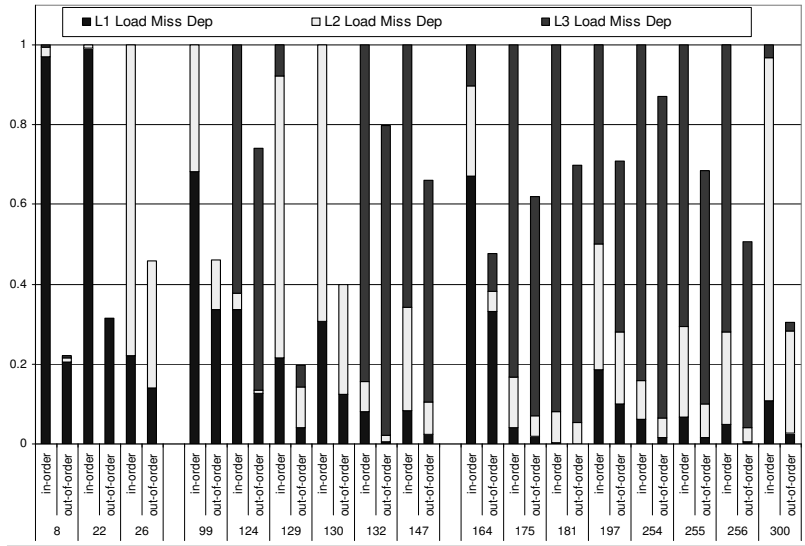




# Out of order cycle breakdowns



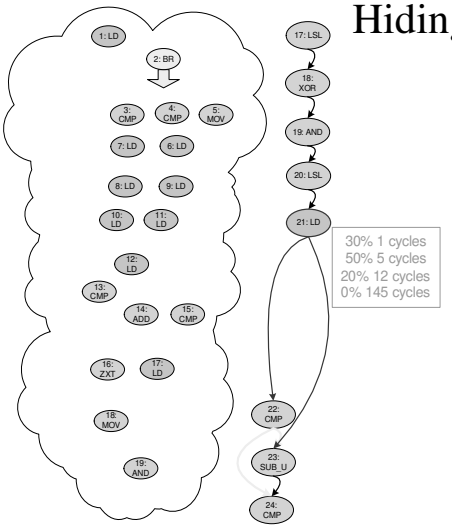
# OoO Memory Latency Tolerance







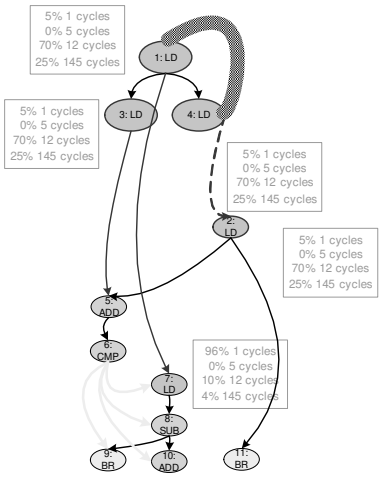

# Hiding misses




- Address calculation and load independent of preceding operations
- Can move up in schedule
- OoO allows slack between load and consumer to stretch to L1 miss lat.
- Compiler could hide some latency in slack between load and consumer
- OoO hoists independent instructions even across taken control-flow

**from deflate_fast_defate in *164.gzip***

# Starting far misses early



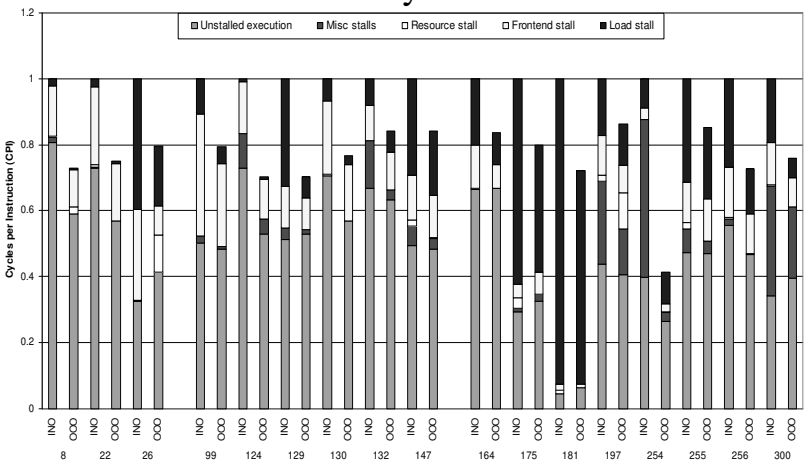

**from price_out_impl in *181.mcf***

- Most important loop in *181.mcf*—dominated by long-latency misses
- Critical loop recurrence though loads 1 and 4—produce address for next loop iteration
- Schedule can stretch allowing likely misses in next iteration to begin early
- OoO adjusts this slack to match actual latencies
- Long misses provide opportunity / connected misses limit speedups—on *181.mcf*, 1.4X speedup on ref vs. 2X speedup on train input
- Ld 7 to same cache line as 3&4



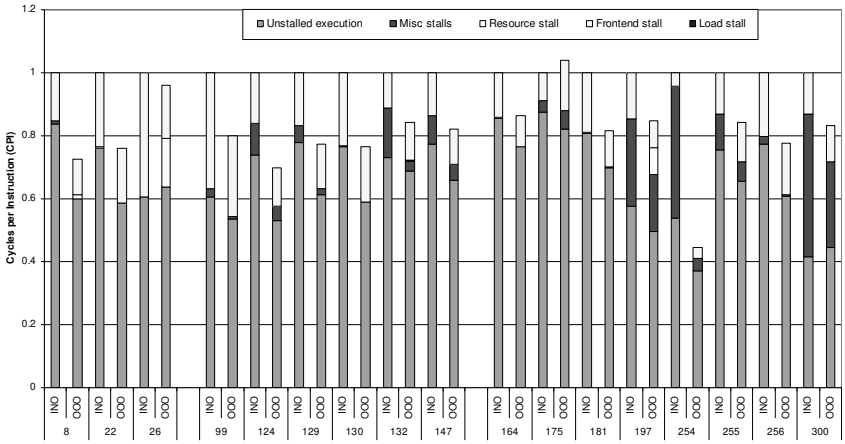


# Out of order cycle breakdowns




- While memory stalls are reduced, this does not fully explain the benefits

# Perfect cache in-order vs. out-of-order

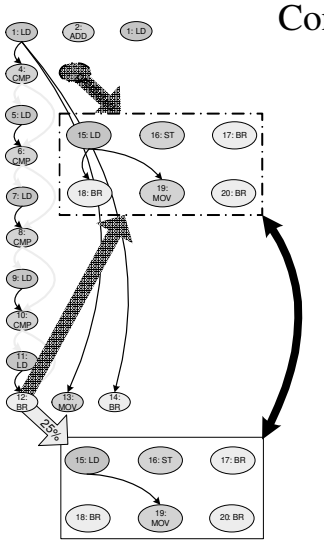



- Experiment attempts to isolate non-memory tolerance benefits



3rd Workshop on EPIC Architectures and Compiler Technology March 21, 2004

# Control speculation

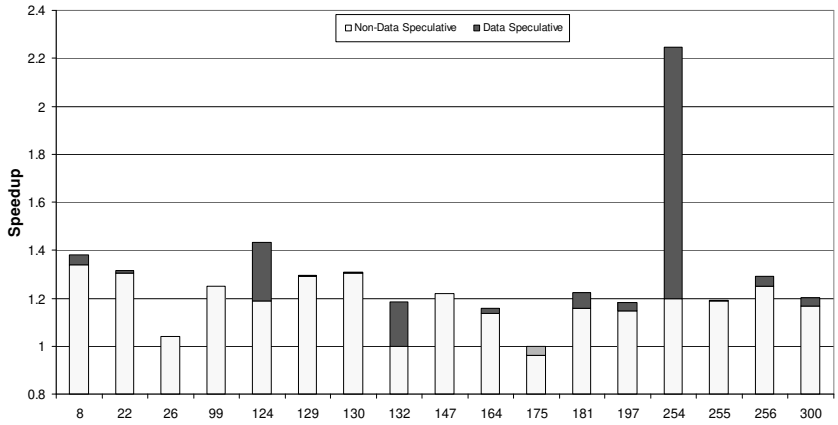



**From *197.parser***

- Relatively frequent side exit (25%)
- Independent code at target of branch can be executed with loop body when branch dynamically predicted
- Compiler speculation of operations into loop potentially costly

EPIC's future: exploring the space between in- and out- of order
Ronald D. Barnes

## Dynamic speculation benefits




- Data speculative benefit extracted by preventing reordering of loads and stores





## Addressing the non-memory opportunities

- Compiler schedules are very effective within its window of scope—but opportunities lie in boundaries
- Overlapping of scheduling regions can yield significant fraction of OoO benefit [SpadiniCGO1]
- The important remaining barriers also represent opportunities for the compiler
  - Infrequent control-flow
  - Low trip count loops
  - Function calls
  - Potentially conflicting memory accesses
- Non-negligible overlap between memory tolerance and non-memory opportunities (e.g. *164.gzip*)

## Conclusions

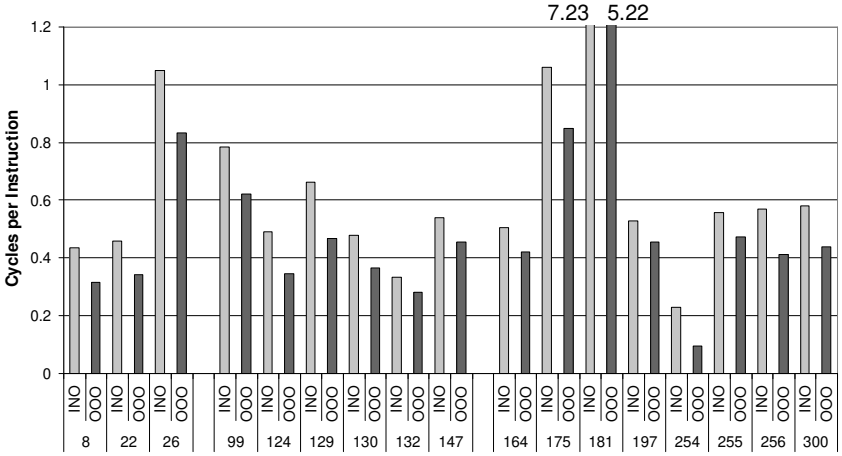
- Flea-flicker approach targeted what was considered to be the only important benefit of out-of-order execution
- Further analysis of the benefits of out-of-order execution suggest a multifaceted approach is needed
- Out-of-order transparently provides these benefits—breaking them down in the presented taxonomy useful in approaching these benefits more cheaply
- Opportunities lie in both compiler and microarchitecture
- Previous work suggests memory tolerance benefits may scale with longer memory latency, however, benefits hampered as connectedness of misses increases





## Backup slides

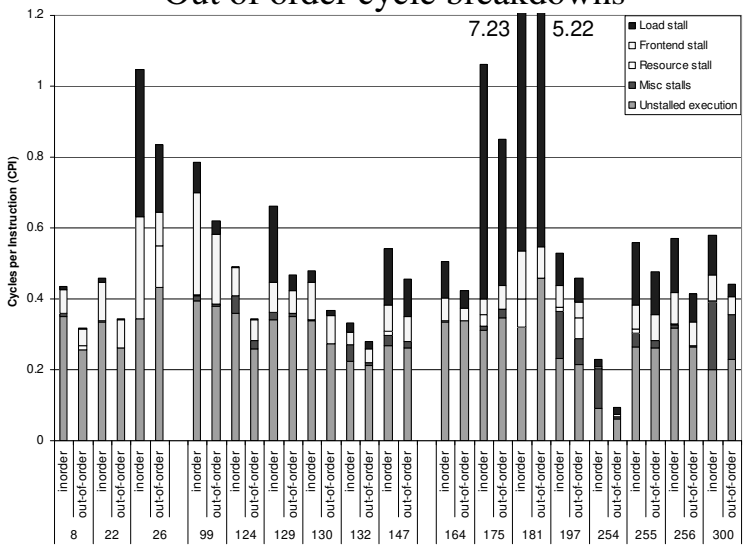
# Out-of-order performance advantage



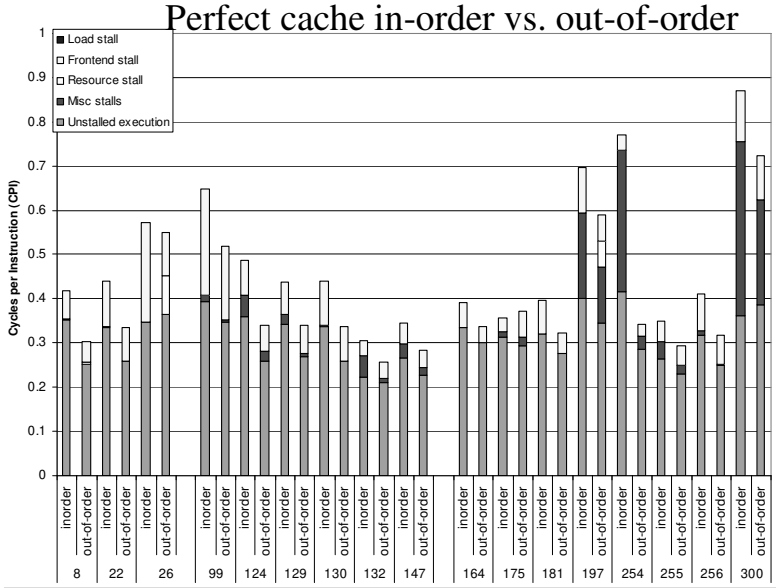



3rd Workshop on EPIC Architectures and Compiler Technology March 21, 2004

# Out of order cycle breakdowns



## Perfect cache in-order vs. out-of-order






3rd Workshop on EPIC Architectures and Compiler Technology March 21, 2004

## Are unanticipated latencies increasing?

- Memory stall cycles have grown from SPEC92 to SPEC2000
- Average latency of memory operations increasing
  - Low-level cache sizes and latencies (in cycles) not significantly improving
  - Main memory access becoming increasingly expensive
  - Caches are becoming increasingly complex
    - More cache levels
    - Increasingly ugly implementation details (i.e. micropipeline events)

EPIC's future: exploring the space between in- and out- of order
Ronald D. Barnes