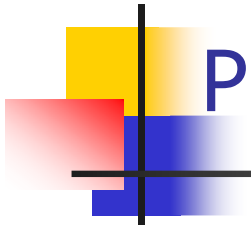
Speculative Data-Driven Multithreading (an implementation of pre-execution)

Amir Roth and Gurindar S. Sohi

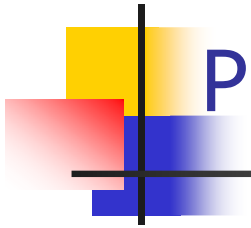
HPCA-7

Jan. 22, 2001



Pre-Execution

- **Goal**: high single-thread performance
- **Problem**: μarchitectural latencies of "problem" instructions (PIs)
  - Memory: cache misses, Pipeline: mispredicted branches
- **Solution**: decouple μarchitectural latencies from main thread
  - Execute copies of PI computations in parallel with whole program
  - Copies execute PIs faster than main thread → "pre-execute"
    - Why? Fewer instructions
    - Initiate Cache misses earlier
    - Pre-computed branch outcomes, relay to main thread
- **DDMT**: an implementation of pre-execution

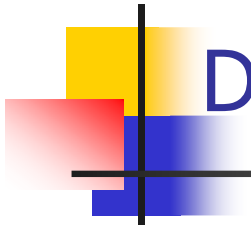


Pre-Execution is a Supplement

- **Fundamentally**: tolerating non-execution latencies (pipeline, memory) requires values faster than execution can provide them

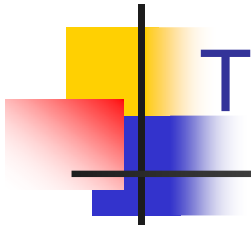
- Ways of providing values faster than execution
  - **Old: Behavioral prediction**: table-lookup
    - + small effort per value, - less than perfect ("problems")
  - **New: Pre-execution**: executes fewer instructions
    - + perfect accuracy, - more effort per value

- **Solution**: supplement behavioral prediction with pre-execution
  - **Key**: behavioral prediction must handle majority of cases
  - **Good news**: it already does



Data-Driven Multithreading (DDMT)

- **DDMT**: an implementation of pre-execution
  - Data-Driven Thread (**DDT**): pre-executed computation of PI
- **Implementation**: extension to simultaneous multithreading (SMT)
  - SMT is a reality (21464)
  - Low static cost: minimal additional hardware
    - Pre-execution siphons execution resources
  - Low dynamic cost: fine-grain, flexible bandwidth partitioning
    - Take only as much as you need
    - Minimize contention, overhead
- Paper: Metrics, algorithms and mechanics
- Talk: Mostly mechanics



Talk Outline

- Working example in 3 parts
- Some details
- Numbers, numbers, numbers

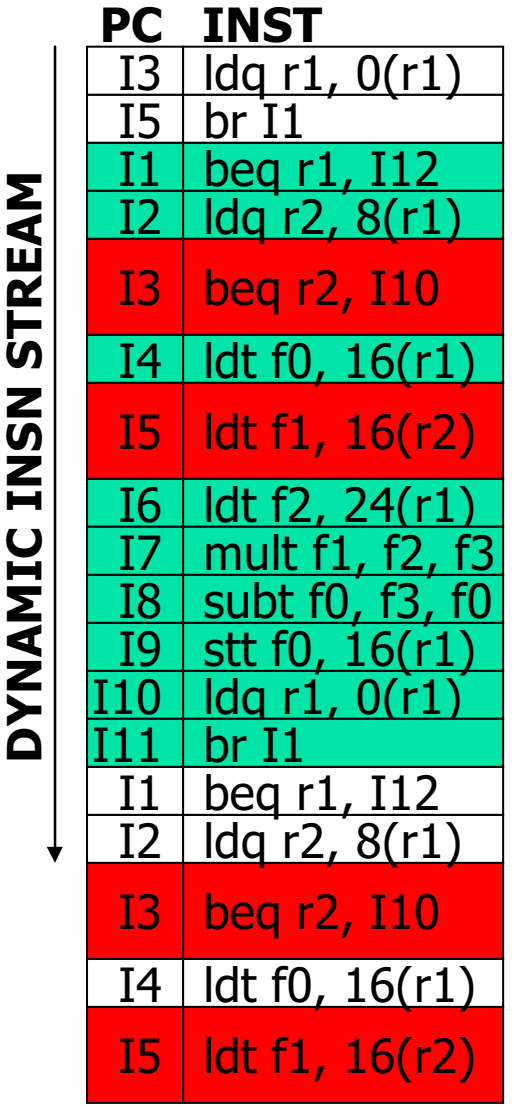
Example.1 Identify PIs

- Running example: same as the paper
  - Simplified loop from EM3D

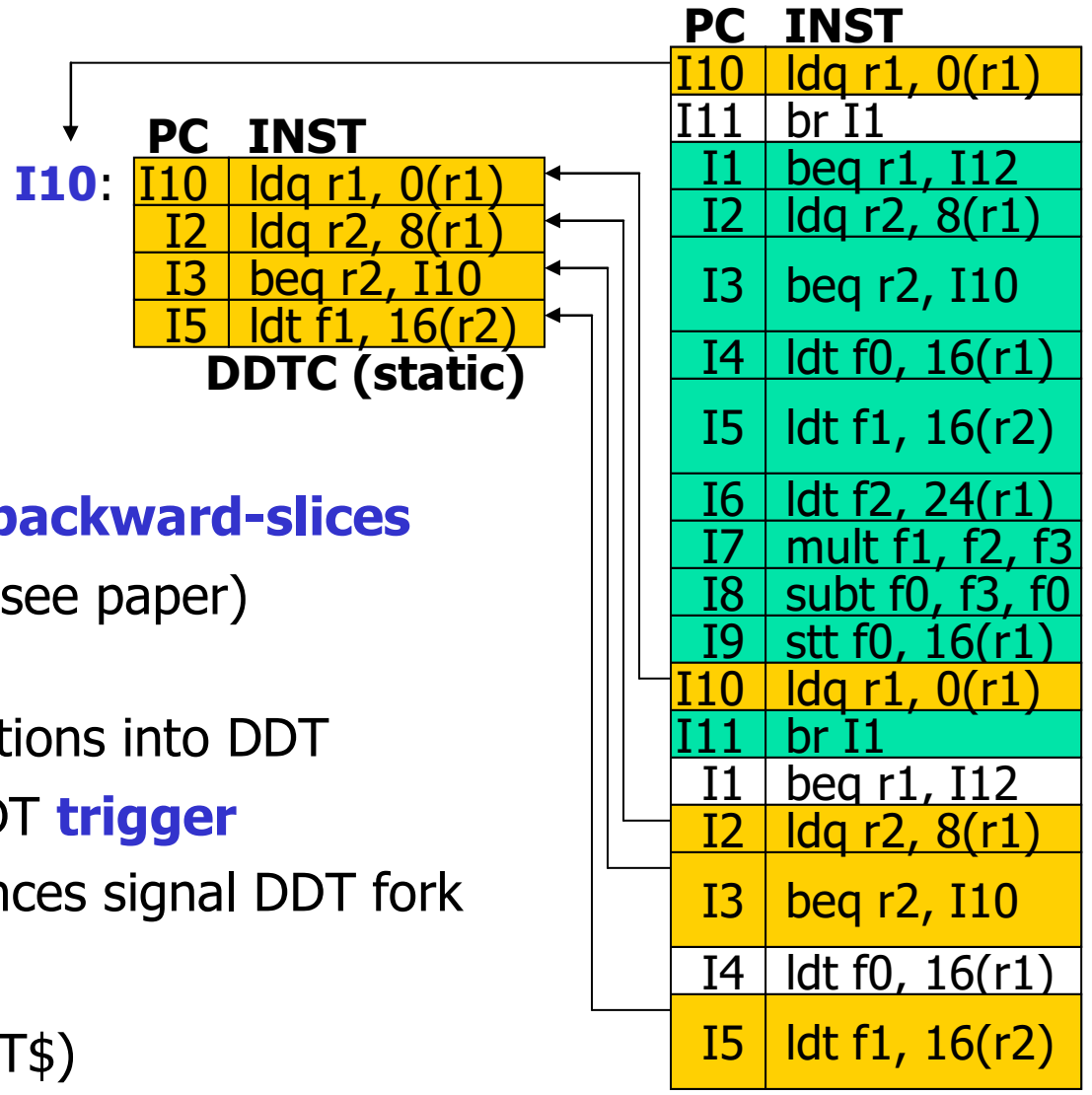
STATIC CODE

```
for (node=list; node; node = node->next)
  if (node->neighbor != NULL)
    node->val -= node->neighbor->val * node->coeff;
```

- Use profiling to find PIs
- Few static PIs cause most dynamic "**problems**"
  - Good coverage with few static DDTs



Example.2 Extract DDTs



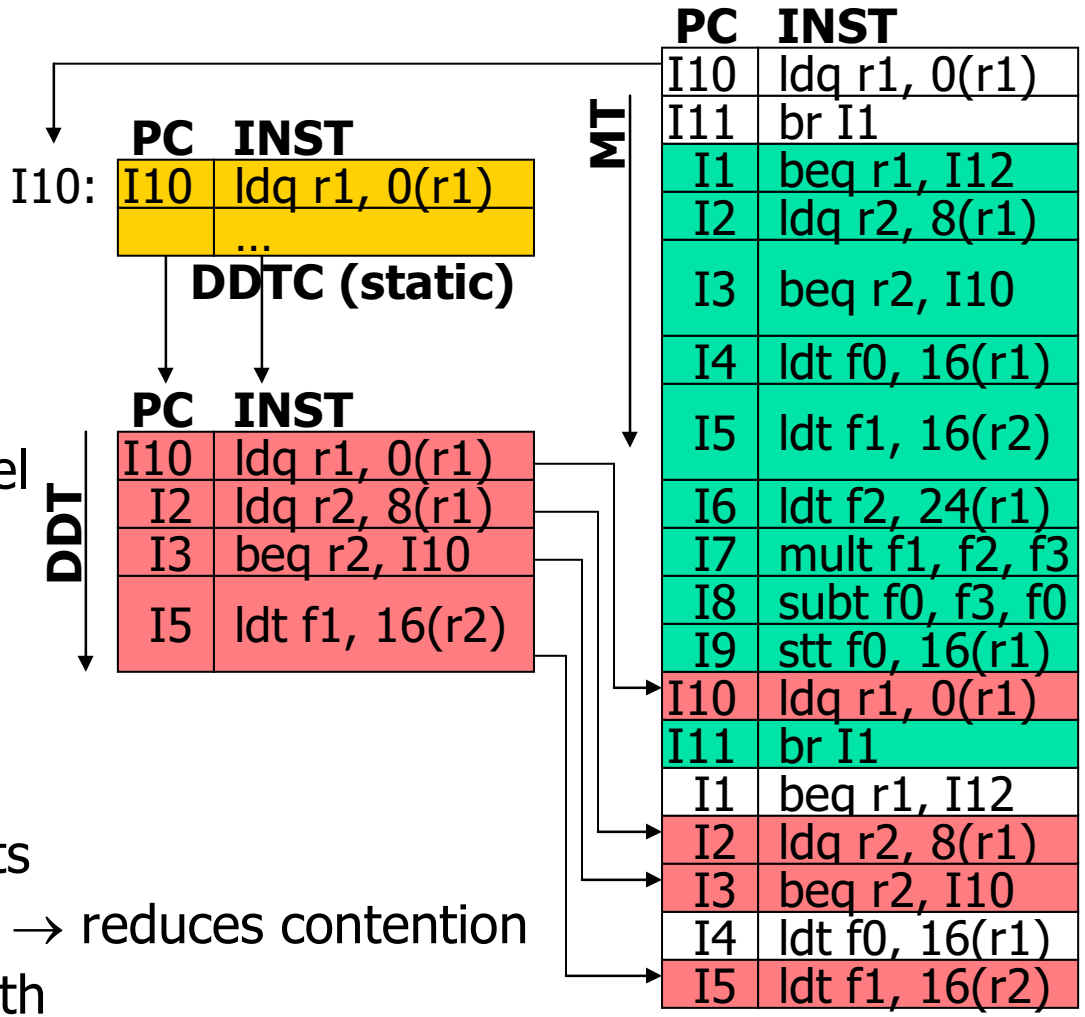
- Examine program traces
- Start with PIs
- Work backwards, gather **backward-slices**
- Eventually stop. When? (see paper)

- Pack last N-1 slice instructions into DDT
- Use first instruction as DDT **trigger**
  - Dynamic trigger instances signal DDT fork

- Load DDT into DDTC (DDT$)

Example.3 Pre-Execute DDTs

- Main thread (MT)
- Executed a trigger instr?
  - **Fork** DDT (μarch)
- MT, DDT execute in parallel
- DDT initiates cache miss
  - "Absorbs" latency
- MT **integrates** DDT results
  - Instr's not re-executed → reduces contention
  - Shortens MT critical path
  - Pre-computed branch avoids mis-prediction



Details.1 More About DDTs

- Composed of instr's from original program
  - Required by integration
- Should look like normal instr's to processor

I10:

PC	INST
I10	ldq r1, 0(r1)
I2	ldq r2, 8(r1)
I3	beq r2, I10
I5	ldt f1, 16(r2)

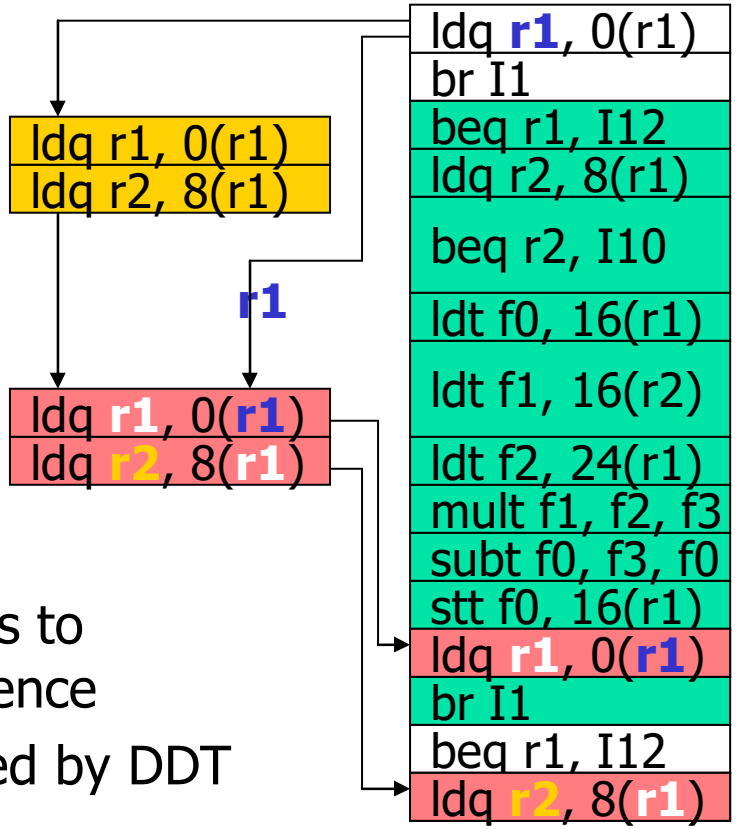
- **Data-driven**: instructions are not "sequential"
  - No explicit control-flow
  - How are they sequenced?
  - Pack into "traces" (in DDTC)
  - Execute all instructions (branches too)
  - Save results for integration
  - No runaway threads, better overhead control
  - "Contain" any control-flow (e.g. unrolled loops)

I10:

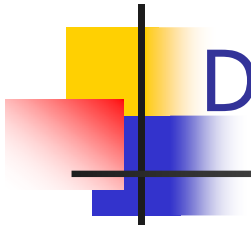
PC	INST
I10	ldq r1, 0(r1)
I10	ldq r1, 0(r1)
I10	ldq r1, 0(r1)
I10	ldq r1, 0(r1)
I2	ldq r2, 8(r1)
I3	beq r2, I10
I5	ldt f1, 16(r2)

Details.2 More About Integration

- Centralized physical register file
  - Use for DDT-MT communication
- **Fork**: copy MT rename map to DDT
  - DDT locates MT values via lookups
  - "Roots" integration
- **Integration**: match PC/physical registers to establish DDT-MT instruction correspondence
  - MT "claims" physical registers allocated by DDT
  - Modify register-renaming to do this



More on integration: implementation of squash-reuse [MICRO-33]

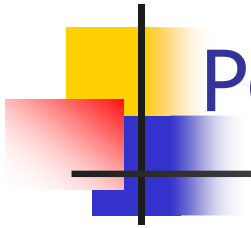


Details.3 More About DDT Selection

- Very important problem
- Very important problem

- Fundamental aspects
  - Metrics, algorithms
  - Promising start
  - See paper

- Practical aspects
  - Who implements algorithm? How do DDTs get into DDTC?
  - Paper: profile-driven, offline, executable annotations
  - Open question



Performance Evaluation

- SPEC2K, Olden, Alpha EV6, –O3 –fast
  - Chose programs with problems
- SimpleScalar-based simulation environment

- DDT selection phase: functional simulation on small input

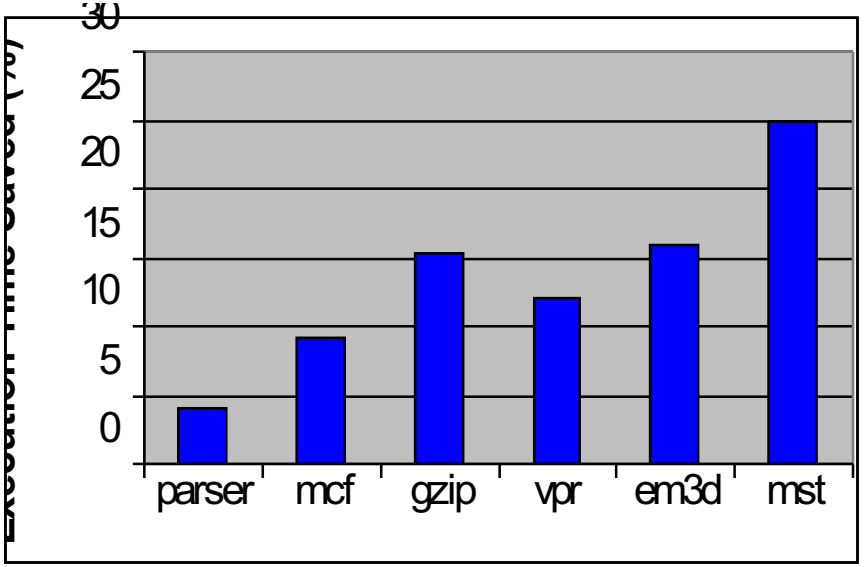
- DDT measurement phase: timing simulation on larger input
  - 8-wide, superscalar, out-of-order core
  - 128 ROB, 64 LDQ, 32 STQ, 80 RS (shared)
  - Pipe: 3 fetch, 2 rename/integrate, 2 schedule, 2 reg read, 2 load
  - 32KB I$/64KB D$ (2-way), 1MB L2$ (4-way), mem b/w: 8 b/cyc.

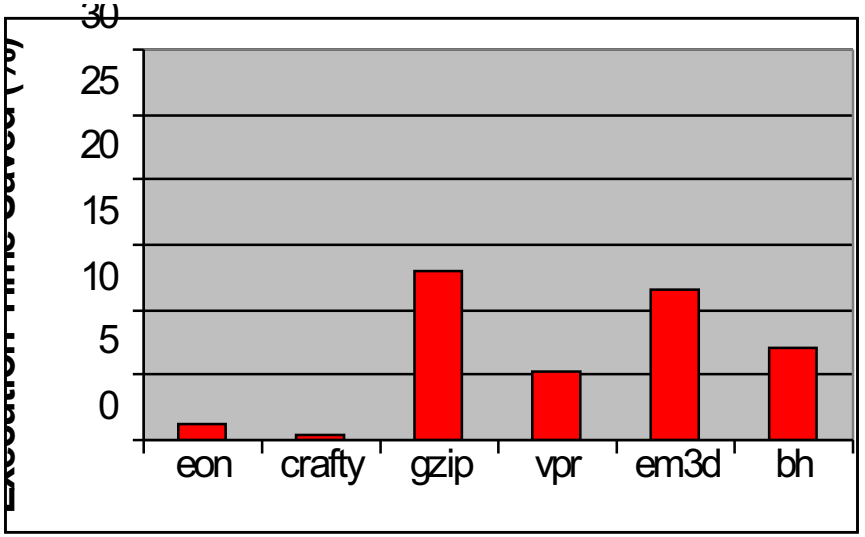
- DDTC: 16 DDTs, 32 instructions (max) per DDT

Numbers.1 The Bottom Line

- **Cache misses**
  - Speedups vary, 10-15%
  - DDT "unrolling": increases latency tolerance (paper)
- **Branch mispredictions**
  - Speedups lower, 5-10%
  - More PIs, lower coverage
  - Branch integration

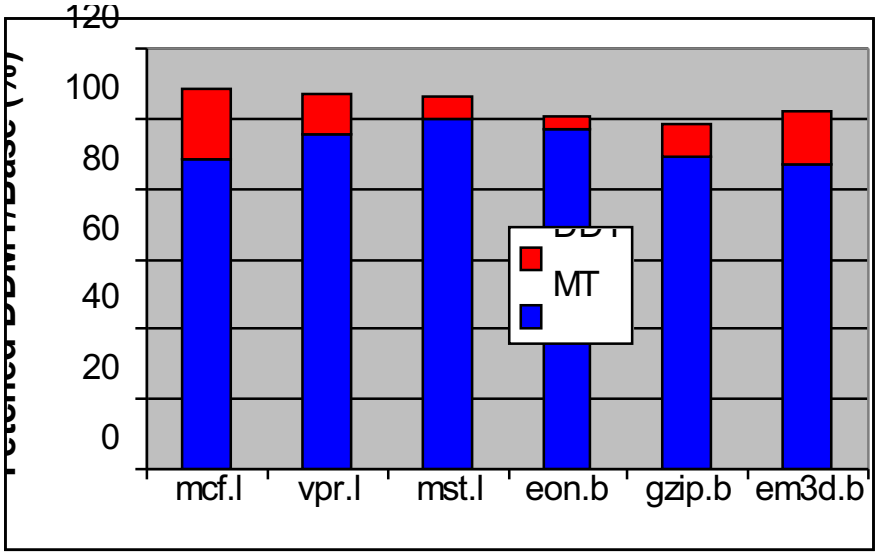
    != perfect branch prediction
- Effects mix

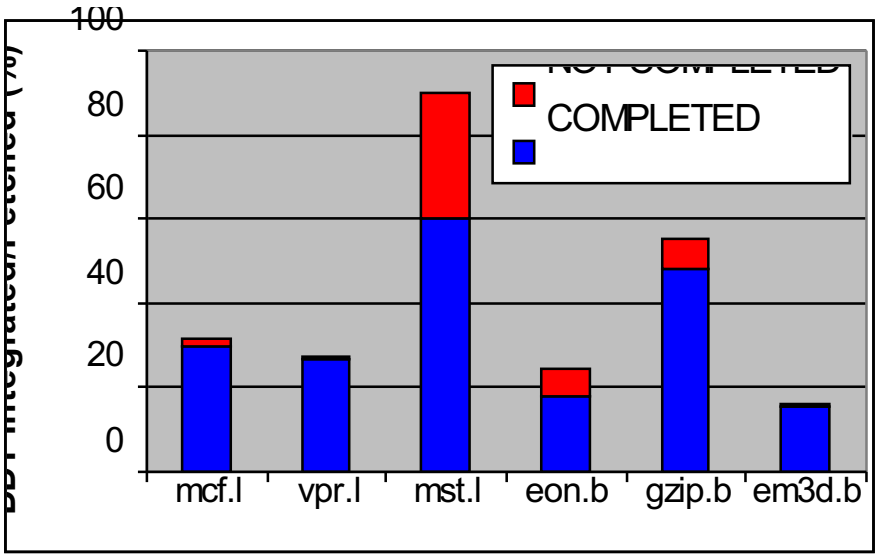




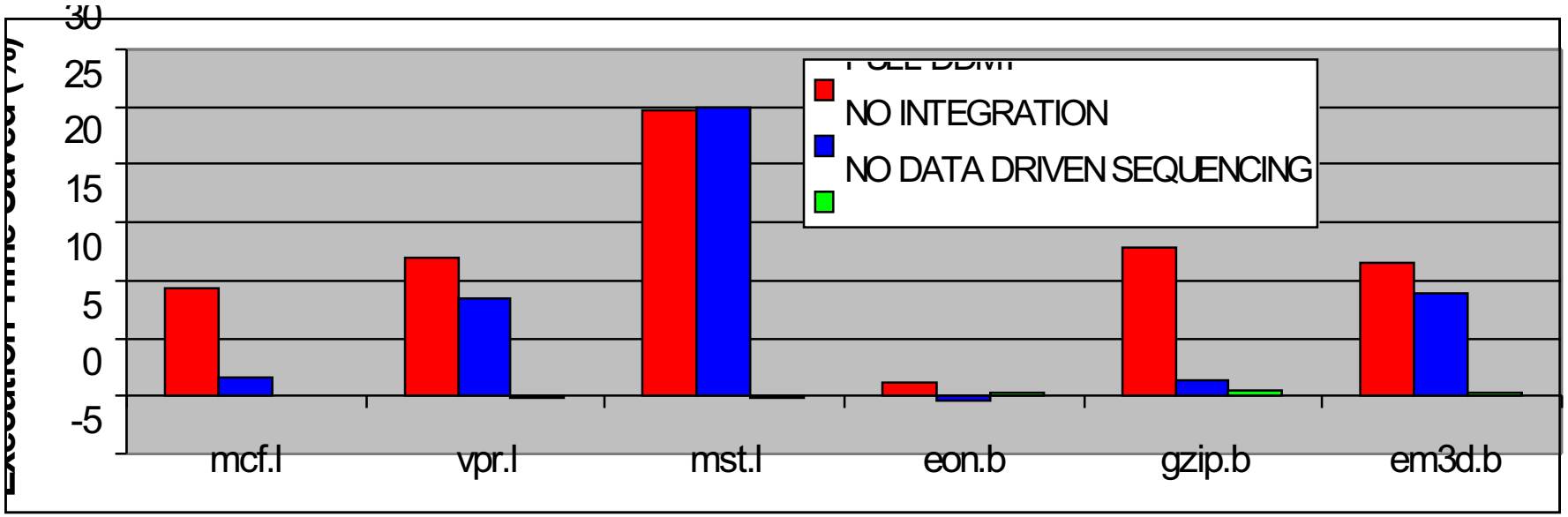
Numbers.2 A Closer Look

- **DDT overhead**: fetch utilization
  - ~5% (reasonable)
  - Fewer MT fetches (always)
    - Contention
  - Fewer total fetches
    - Early branch resolution
- **DDT utility**: integration rates
  - Vary, mostly ~30% (low)
  - Completed: well done
  - Not completed: a little late
  - Input-set differences
  - Greedy DDTs, no early exits

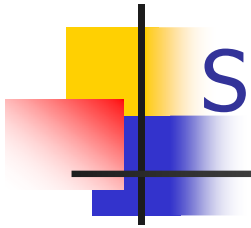




Numbers.3 Is Full DDMT necessary?



- How important is **integration**?
  - Important for branch resolution, less so for prefetching
- How important is **decoupling**? What if we just priority-scheduled?
  - Extremely. No data-driven sequencing → no speedup



Summary

- **Pre-execution**: supplements behavioral prediction
  - Decouple/absorb μarchitectural latencies of "problem" instructions
- **DDMT**: an implementation of pre-execution
  - An extension to SMT
    - Few hardware changes
    - Bandwidth allocation flexibility reduces overhead
- Future: DDT selection
  - Fundamental: better metrics/algorithms to increase utility/coverage
  - Practical: an easy implementation
- Coming to a university/research-lab near you