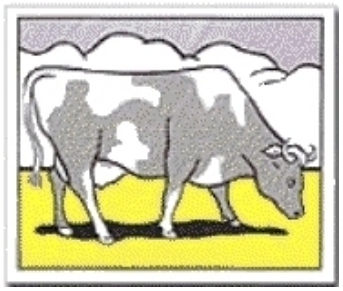
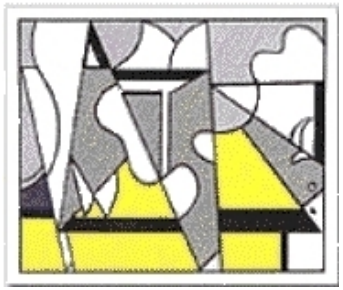
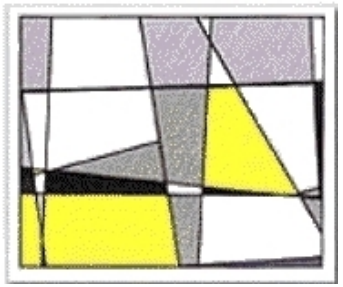
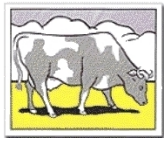




A Dynamic Binary Translation Approach to Architectural Simulation

Harold "Trey" Cain, Kevin Lepak, and Mikko Lipasti

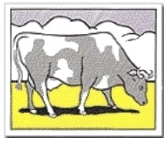
Computer Sciences Department
Department of Electrical and Computer Engineering
University of Wisconsin

http://www.ece.wisc.edu/~pharm



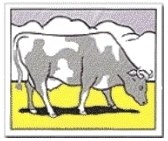
Introduction

- Developing execution-driven PowerPC architectural simulator, using existing out-of-order simulator - SimpleScalar.
- We would like to remain compatible with other versions of SimpleScalar.
- Perform dynamic binary translation from PowerPC to SimpleScalar's Portable Instruction Set Architecture (PISA).
- Translation occurs in extra pipeline stage between fetch and decode.
  - similar to x86 instruction cracking from CISC instructions to RISC-like μ–ops.



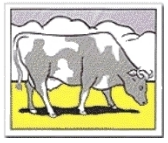
Motivations

- We change a minimum of the original SimpleScalar code.
- We save development time.
- We can use the translator to study new microarchitectural optimizations enabled by CISC to RISC translation.



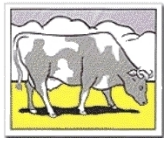
Outline

- Architectural Simulation: SimpleScalar
- Implications of using translation in a simulator
- Implementation:
  - State Mapping: PowerPC->PISA
  - Complications: Memory Operations
  - Solution: Speculative Decode
- Translation Efficiency



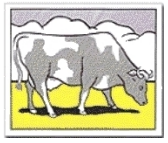
Architectural Simulation

- Hardware is expensive!
- Reasoning about complex systems using analytic models alone is difficult.
- Using simulation, we can test new architectural ideas without building hardware.
- Rapid growth in computer performance has enabled increasingly detailed simulators.
  - SimOS can boot commercial operating systems.



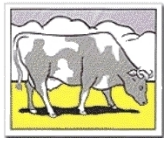
SimpleScalar

- Execution-driven simulator models the internals of out-of-order microprocessor
- Implements the Portable Instruction Set Architecture (PISA), a MIPS derivative
- Many different versions in existence:
  - More than ¼ of PACT 2000 papers use SimpleScalar.
- We hope to leverage this significant body of work.



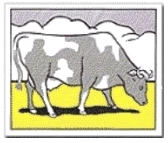
Why do binary translation?

- Another alternative is to directly modify SimpleScalar
  - It already includes hooks for supporting other architectures: e.g. Alpha
- However, PowerPC ISA does not easily map to SimpleScalar's machine.def architecture specification format
  - For instance, SimpleScalar assumes an instruction will change at most two operands
  - Some PowerPC instructions write up to 32 output registers



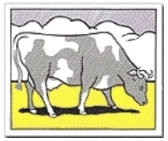
Implications

- We have different constraints than traditional binary translators
- Primary goal: to accurately model the internals of an out-of-order microprocessor
  - For some instructions, the overhead of performing binary translation affects simulation accuracy
- If accuracy is negatively affected by translation overhead, we have the luxury of a flexible target architecture.



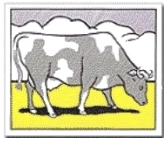
Notable Differences: PowerPC vs. PISA

PowerPC	PISA
32 bit Instructions	64 bit Instructions
Result of compares stored in special CR	Result of compares stored in GPRs
Allows unaligned memory references	Disallows unaligned memory references
Single instructions may modify up to 32 registers	All register-writing instructions modify at most two registers
Contains supervisor level state and instructions	All system calls proxied by SimpleScalar

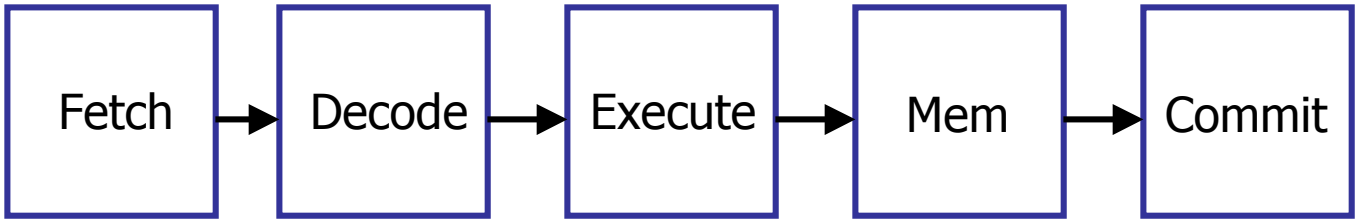


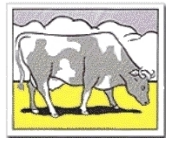
Outline

- Architectural Simulation: SimpleScalar
- Implications of using translation in a simulator
- **Implementation:**
  - State Mapping: PowerPC->PISA
  - Complications: Memory Operations
  - Solution: Speculative Decode
- Translation Efficiency

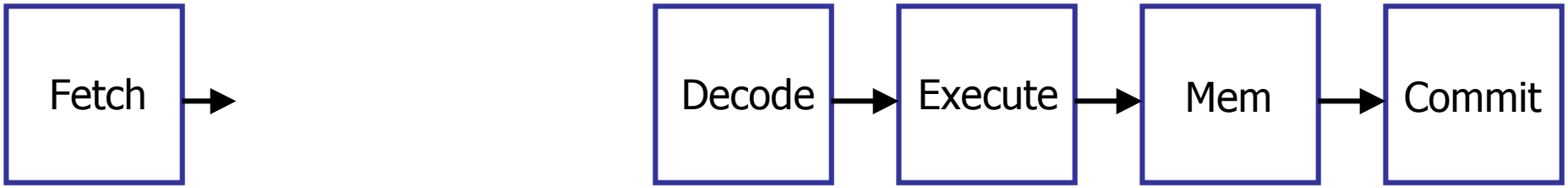


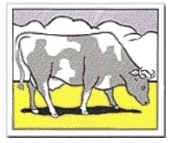
SimpleScalar Pipeline



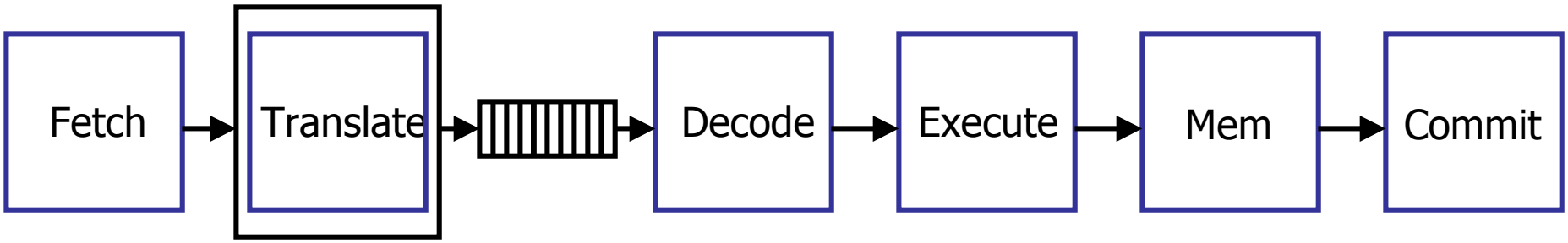


SimpleScalar Pipeline

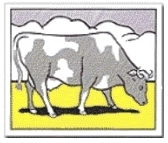




SimpleScalar Pipeline

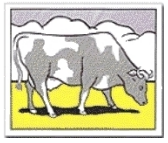


- Fetch stage minimally changed
- Pipeline stages from decode to commit unchanged



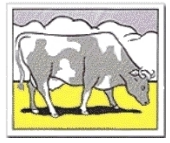
PowerPC->PISA State Mapping

PowerPC Registers	SimpleScalar Registers
32 GPRs	32 GPRs
Link Reg.	1 GPR
Count Reg.	1 GPR
Condition Reg.	8 GPRs
Exception Reg.	4 GPRs
FP Status Control Reg	1 GPR
32 Floating Point Regs	64 Floating Point Regs

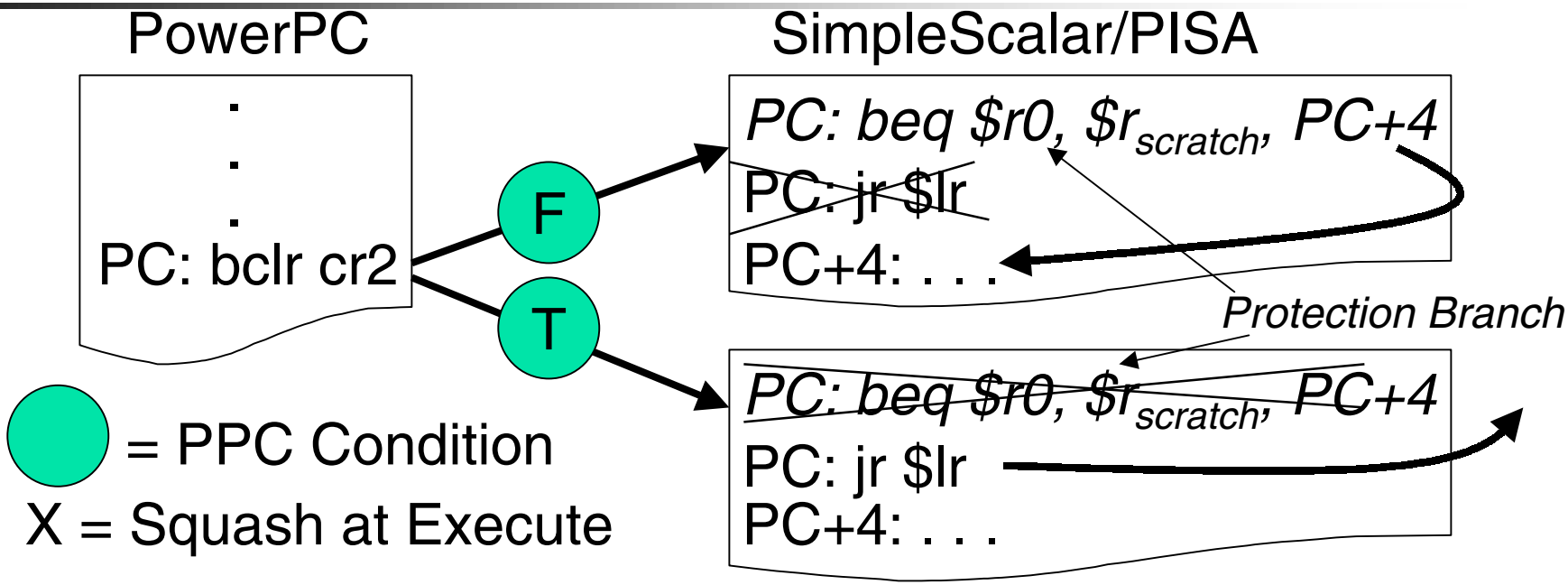


Control Transfer Instructions

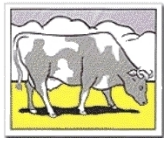
- Control instructions in PowerPC are powerful (e.g. *bdnztlrl*) or slightly more general (e.g. *bclr*)
  - To translate, we need to allow multiple branches in the translation of a single PowerPC instruction
- Need to insure that SimpleScalar branch predictors/etc. are not impacted substantially by two control instructions at the same PC
  - Also optimize common cases
- Need to assure superblock structures to eliminate instruction address space issues



Control Transfer--Continued

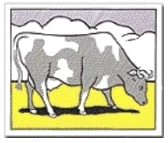


- Only one control transfer instruction "appears" at PC (PowerPC branch location)
- Translations maintain superblock properties



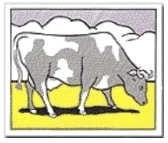
Memory Operations

- Two issues:
  - Alignment – PowerPC supports unaligned memory access, PISA does not.
    - Most memory operations use register+offset addressing mode
  - PowerPC *lswx* and *stswx* string instructions
    - Read/Write a variable number of bytes from memory, length specified by register
- Cannot perform translation until all operands have been written
- Naïve implementation would stall pipeline, affecting performance



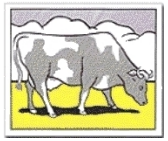
Speculative Decode

- Optimistically translate instructions into a simpler sequence by exploiting a runtime attribute
  - Translate all memory operations assuming natural alignment
  - Translate all *lswx* and *stswx* instructions by predicting their size with a history-based predictor
- If speculation is incorrect, roll back pipeline

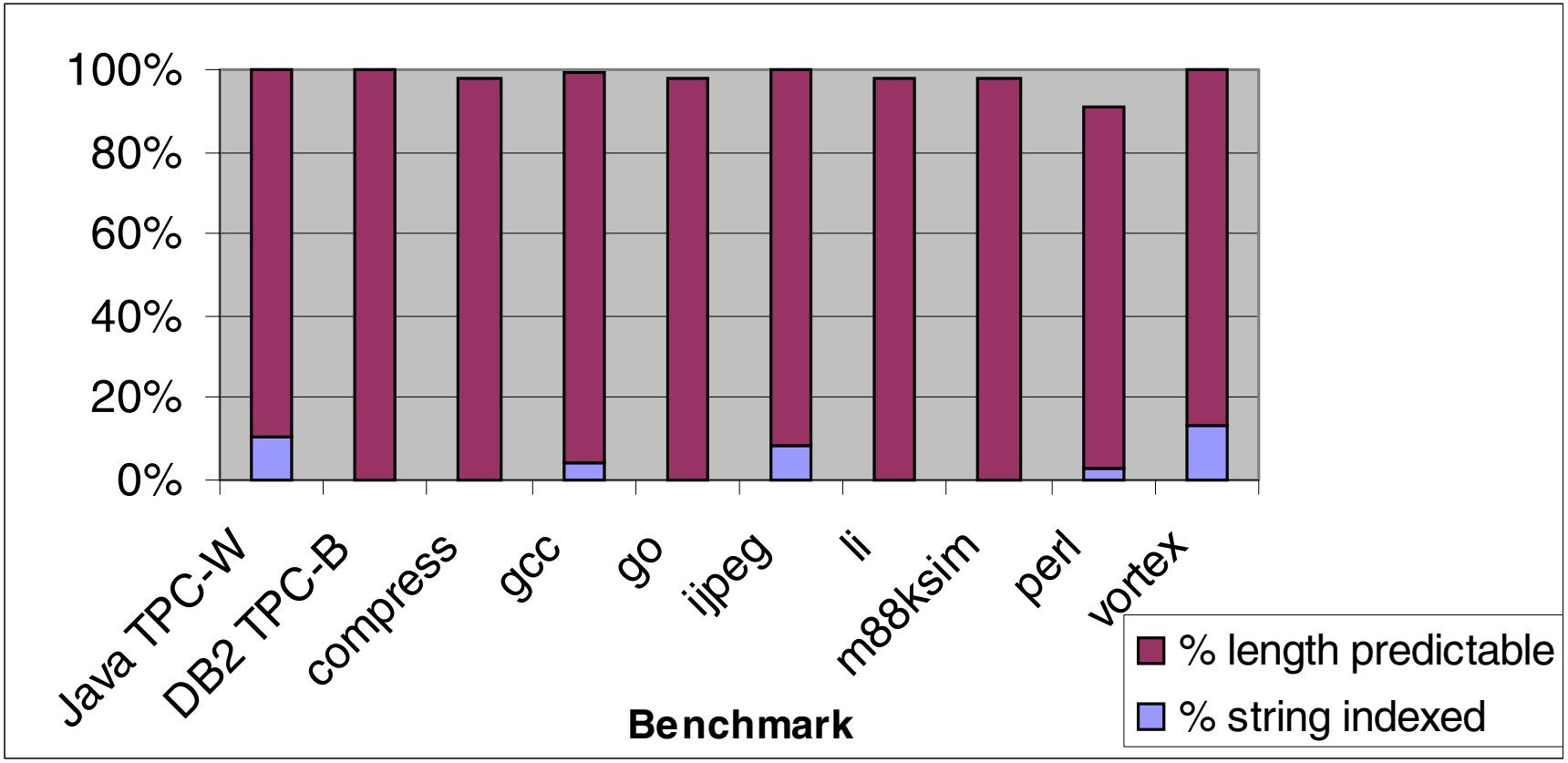


Alignment Prediction

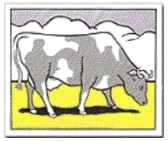
Application	unaligned references
DB2 TPC-B	.00%
Java TPC-W	.34%
compress	.00%
gcc	.01%
go	.00%
ijpeg	.02%
li	.00%
m88ksim	.00%
perl	.00%
vortex	.00%



Length Prediction



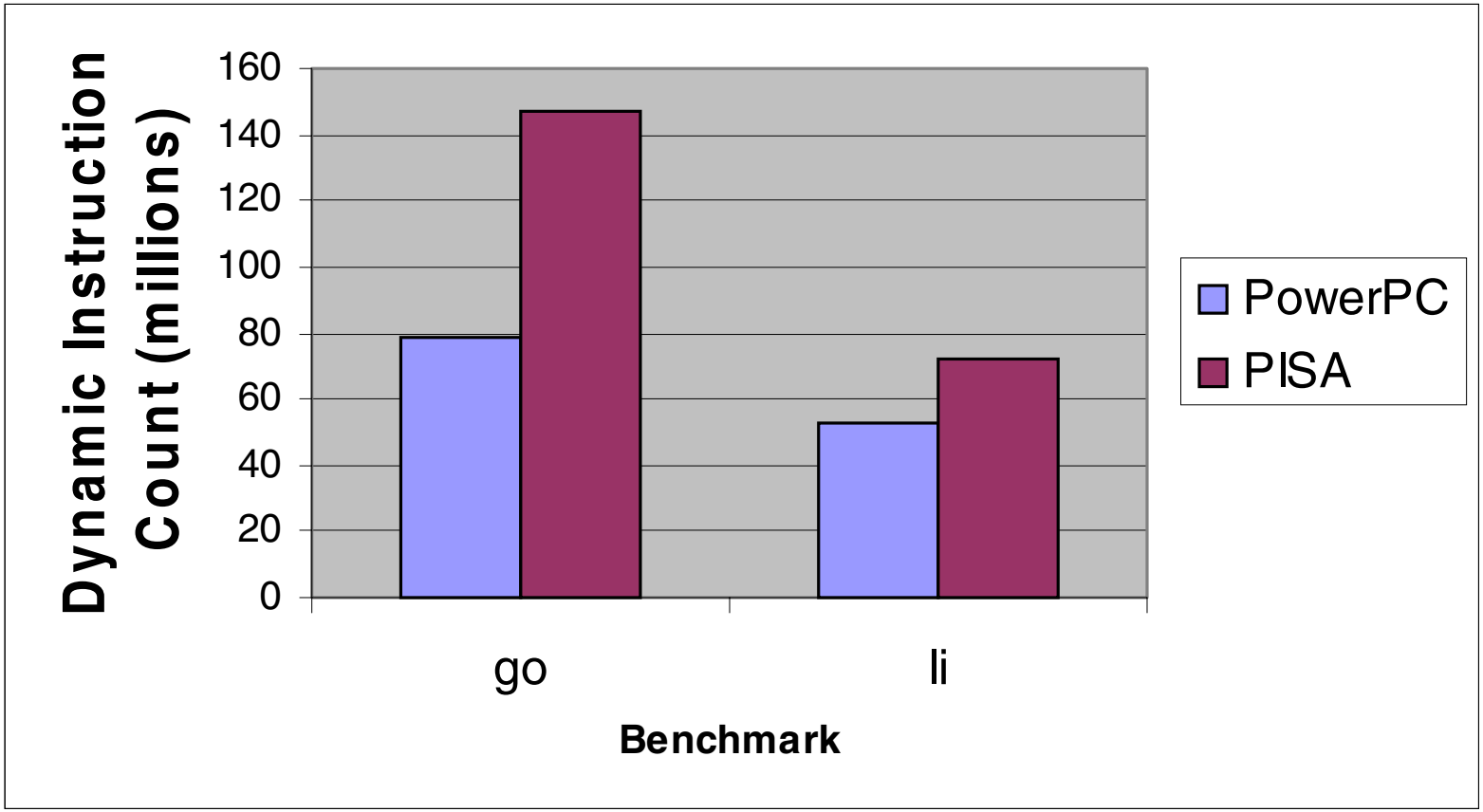
Using 256 entry last-value predictor



Outline

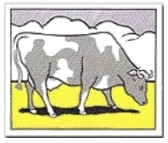
- Architectural Simulation: SimpleScalar
- Implications of using translation in a simulator
- Implementation:
  - State Mapping: PowerPC->PISA
  - Complications: Memory Operations
  - Solution: Speculative Decode
- **Translation Efficiency**

Instruction Expansion



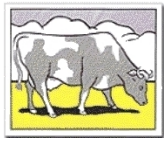
Dynamic Instruction Growth: 86% and 35% respectively

Memory Instr Growth < 1%



Future Work

- Simulation Infrastructure
  - Running more applications!
  - Integrating Translator into a Multiprocessor version of SimpleScalar (SimpleMP, Ravi Rajwar)
  - Integrating Translator/SimpleMP into SimOS-PPC full system simulator
- Speculative Decode



Questions?