The Use of Multithreading for Exception Handling

Craig Zilles, Joel Emer*, Guri Sohi

University of Wisconsin - Madison

***Compaq - Alpha Development Group**

International Symposium on Microarchitecture - 32

November, 1999

Overview

Extensions to a multithreaded processor to reclaim lost performance during exception handling in a pipelined, out-of-order processor

- HARDWARE EXCEPTIONS
- PERFORMANCE IN TRADITIONAL IMPLEMENTATION
- IMPORTANT CHARACTERISTICS OF EXCEPTION HANDLERS
- EXPLOIT THEM WITH EXTENSION TO SMT PROCESSOR
- METHODOLOGY/PERFORMANCE
- AN OPTIMIZATION: QUICK-STARTING
- CONCLUSIONS

Hardware exceptions

COST-EFFECTIVE HARDWARE → UNCOMMON CASE HANDLED BY SOFTWARE

RECOVERABLE EXCEPTIONS (NOT SEGFAULTS)

- *TLB miss*
- *unaligned access*
- *emulated instructions*

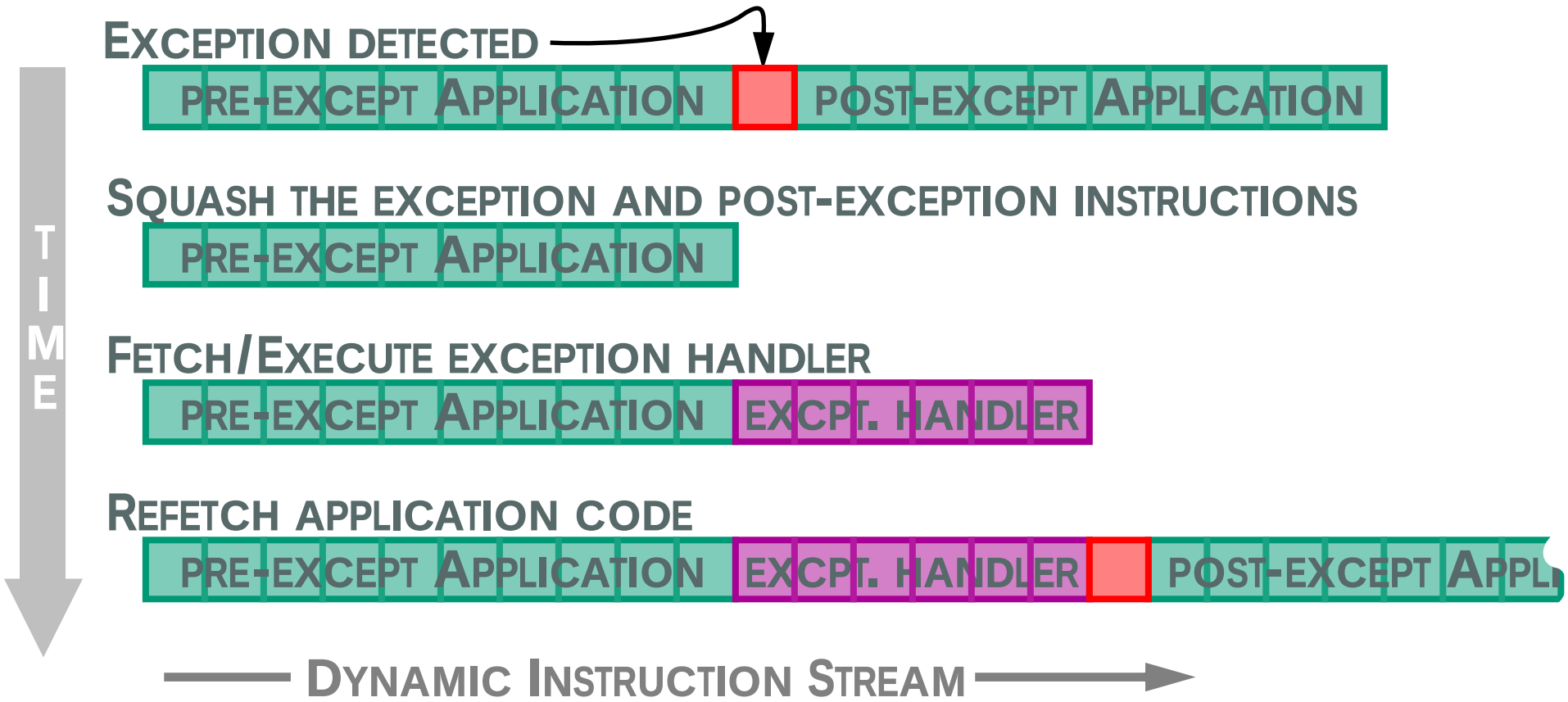
EVENT DETECTED BY HARDWARE, RESOLVED BY SOFTWARE

- *A short piece of code is executed*
- *Control is returned to the application at the exception*

Performance problem

MUCH LIKE BRANCH MISPREDICT

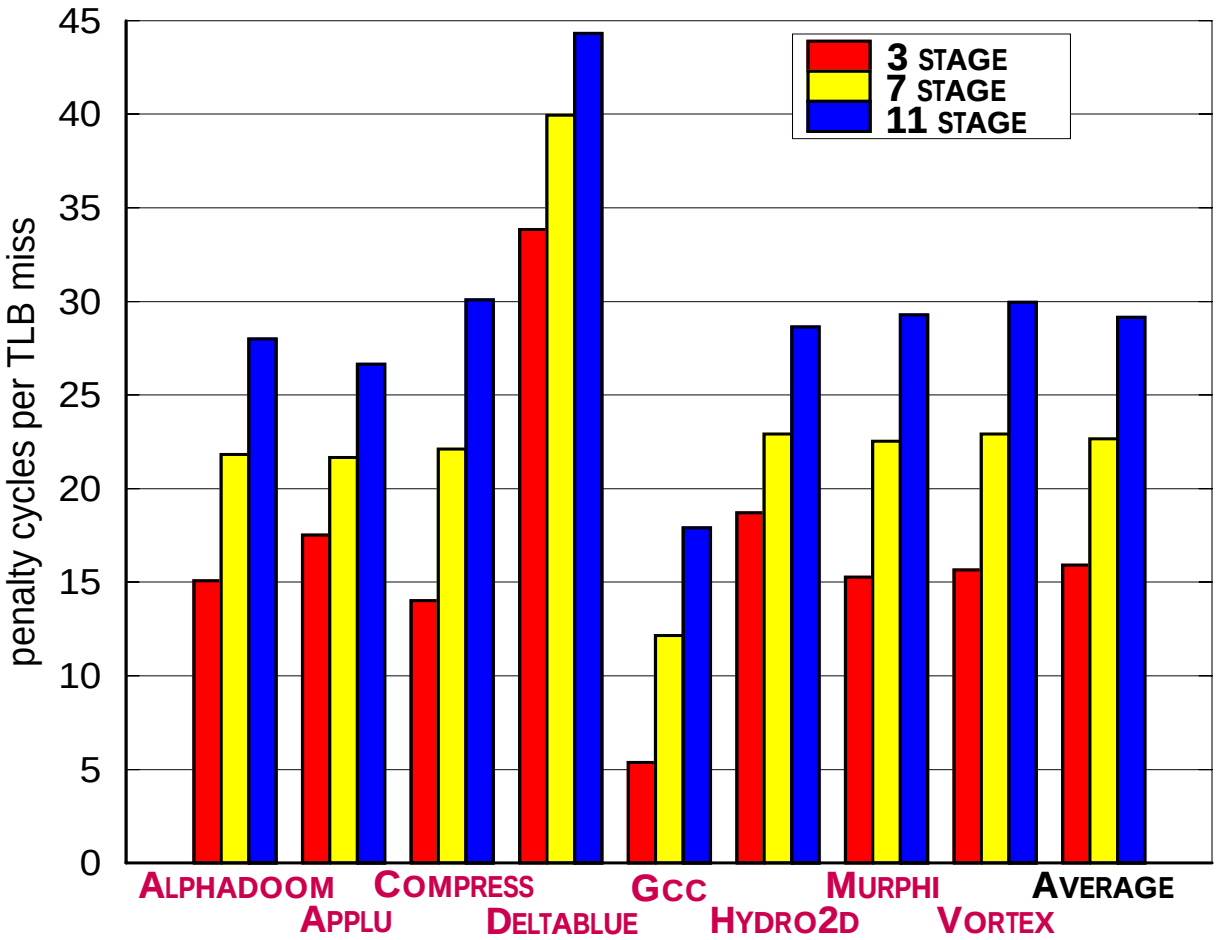
- *causes* **CHANGE IN CONTROL FLOW**
- *often* **DETECTED AT EXECUTE TIME**



Trends

With increased pipeline length, superscalar width, and window size

- **it only gets worse**



Structure of Exception Handler

RECONVERGENT CONTROL FLOW

The same application instructions are executed in the same order INDEPENDENT of the exception handler's execution

MINIMAL DATA DEPENDENCES *between application and exception handler*

typically only involving excepting instruction

Example: **TLB MISS HANDLER**

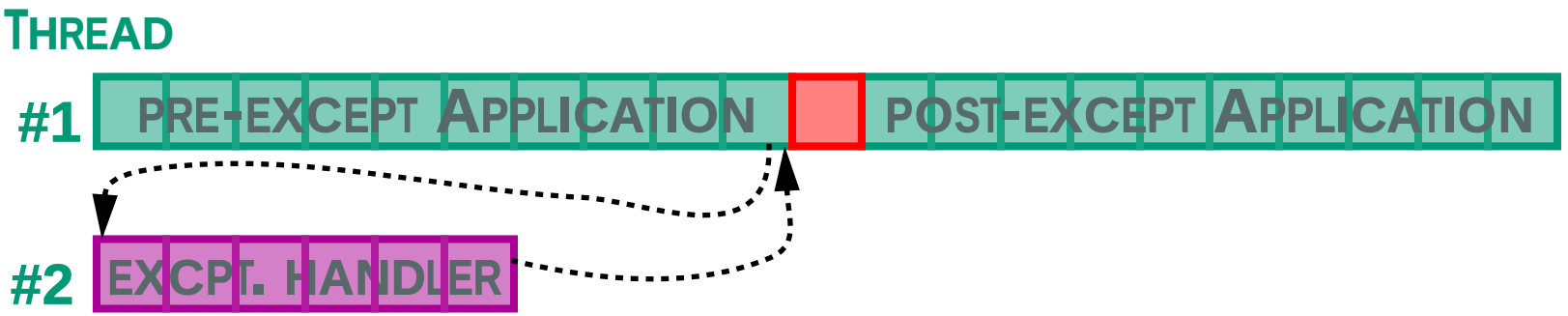
- *reads miss address from privileged register*
- *loads from page table*
- *writes TLB*

Extension to SMT processor

RECONVERGENT CONTROL FLOW → DON'T SQUASH

ALLOCATE THE HANDLER TO SEPARATE THREAD

- *FIFO management of window resources (within a thread)*
- *extra hardware required for ordering threads*



PROVIDE APPEARANCE OF SEQUENTIAL EXECUTION

- *Control thread retirement order*

Extension to SMT processor

MINIMAL DATA DEPENDENCES → USE SEPARATE REGISTER FILE

Avoids additional renamer complexity

UNCOMMON CASE (TLB MISS → PAGE FAULT → CONTEXT SWITCH)

- **REVERT TO NORMAL MECHANISM**

MEMORY DEPENDENCES

- **DETECT CONFLICTS, RECOVER (MUCH LIKE R10K, OR ARB)**

Methodology

Example implementation: software TLB miss handling

- Execution driven SMT simulator
- Built from alpha architecture SimpleScalar toolkit
- Supports enough of 21164 privileged architecture to run common-case TLB handler
  - speculative execution, multiple in-flight misses
- 8 wide, 128 window, 7 stage, big YAGS, 64K L1's, 1M L2
- Benchmarks with non-trivial TLB behavior (from SPEC and elsewhere)
- Scaled down (64 entry) data TLB

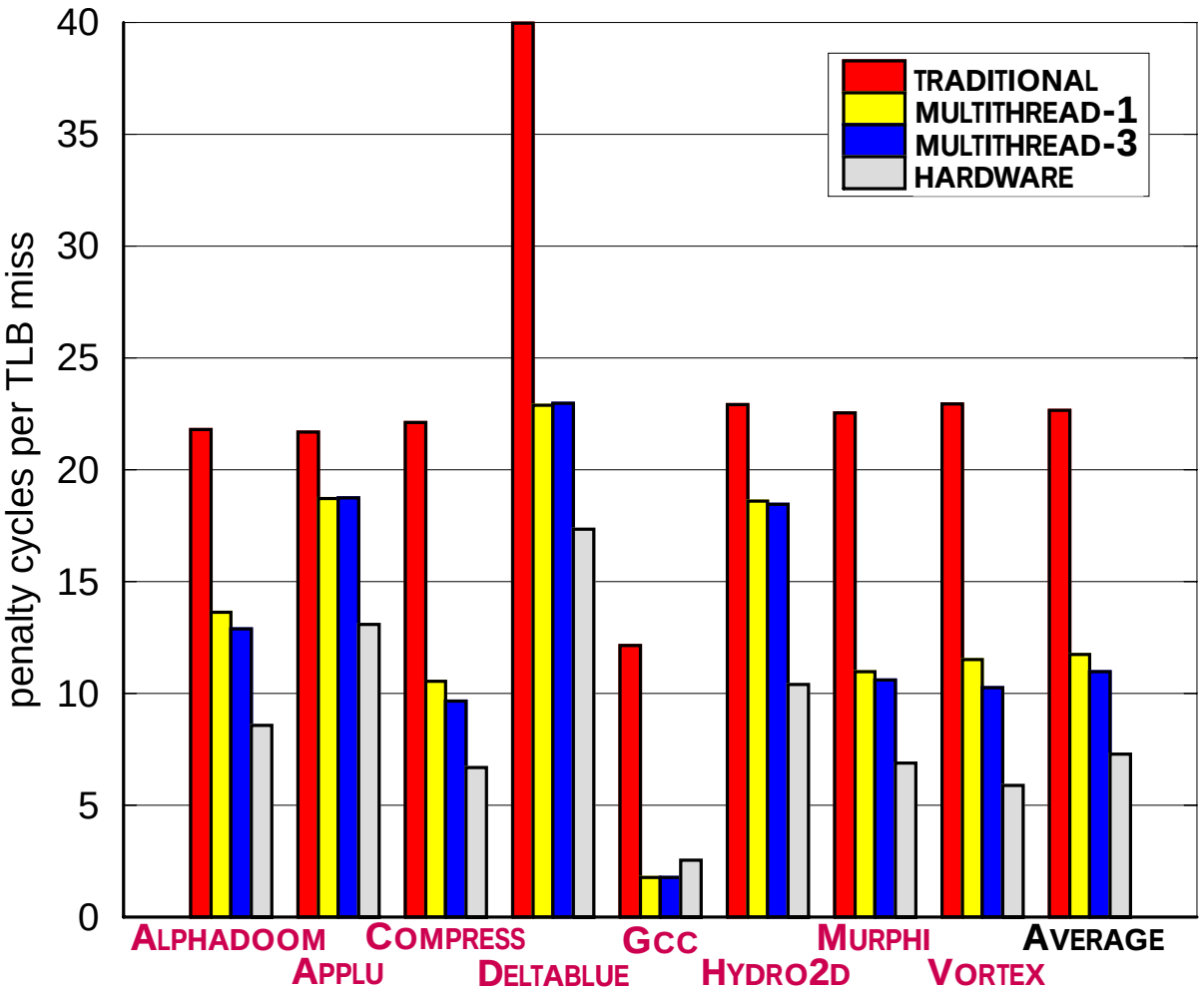
Metric: penalty per miss

(additional overhead vs. simulation with perfect TLB) / misses

Performance

DOES MUCH BETTER THAN TRADITIONAL SOFTWARE APPROACH

NOT AS GOOD AS AGGRESSIVE HARDWARE TLB MISS WIDGET



Optimization: Quick-starting

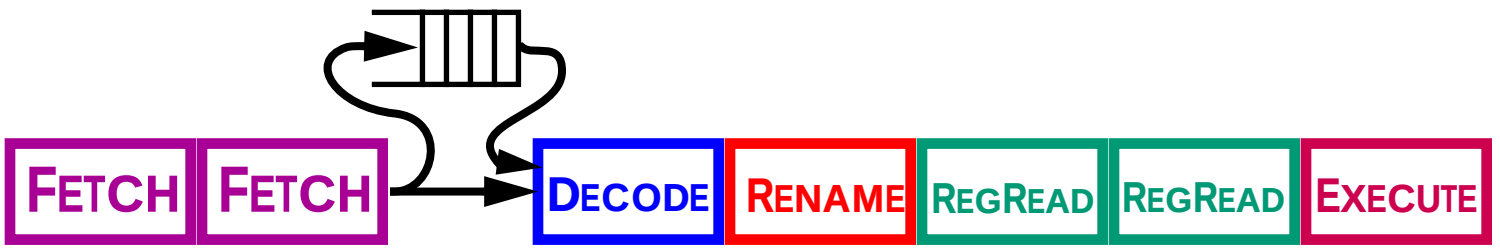
Performance gap between hardware and multi-threaded

- Fetch/Decode latency

Solution: Cache exception handler partway down pipeline

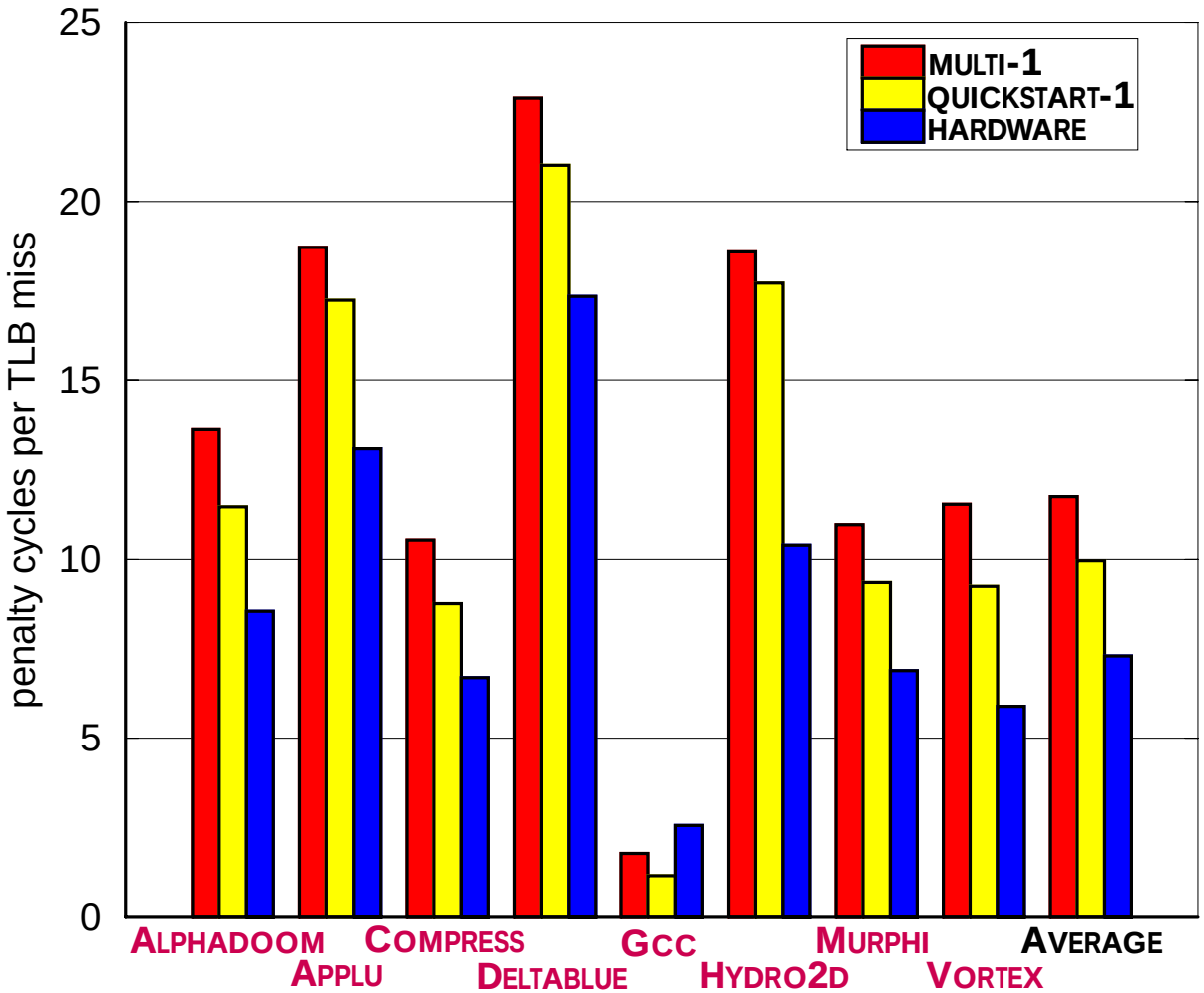
Our SMT Implementation:

- Per thread fetch buffers, idle resources when thread is idle
- Predict next exception, use idle fetch cycles to prefetch handler.
- Reduces multi-threaded exception latency.



Performance: Quick-starting

ALMOST CUTS PERFORMANCE GAP IN HALF



Single Thread Performance vs. Throughput

Single Application: (previous results)

Focus: Improve single thread performance

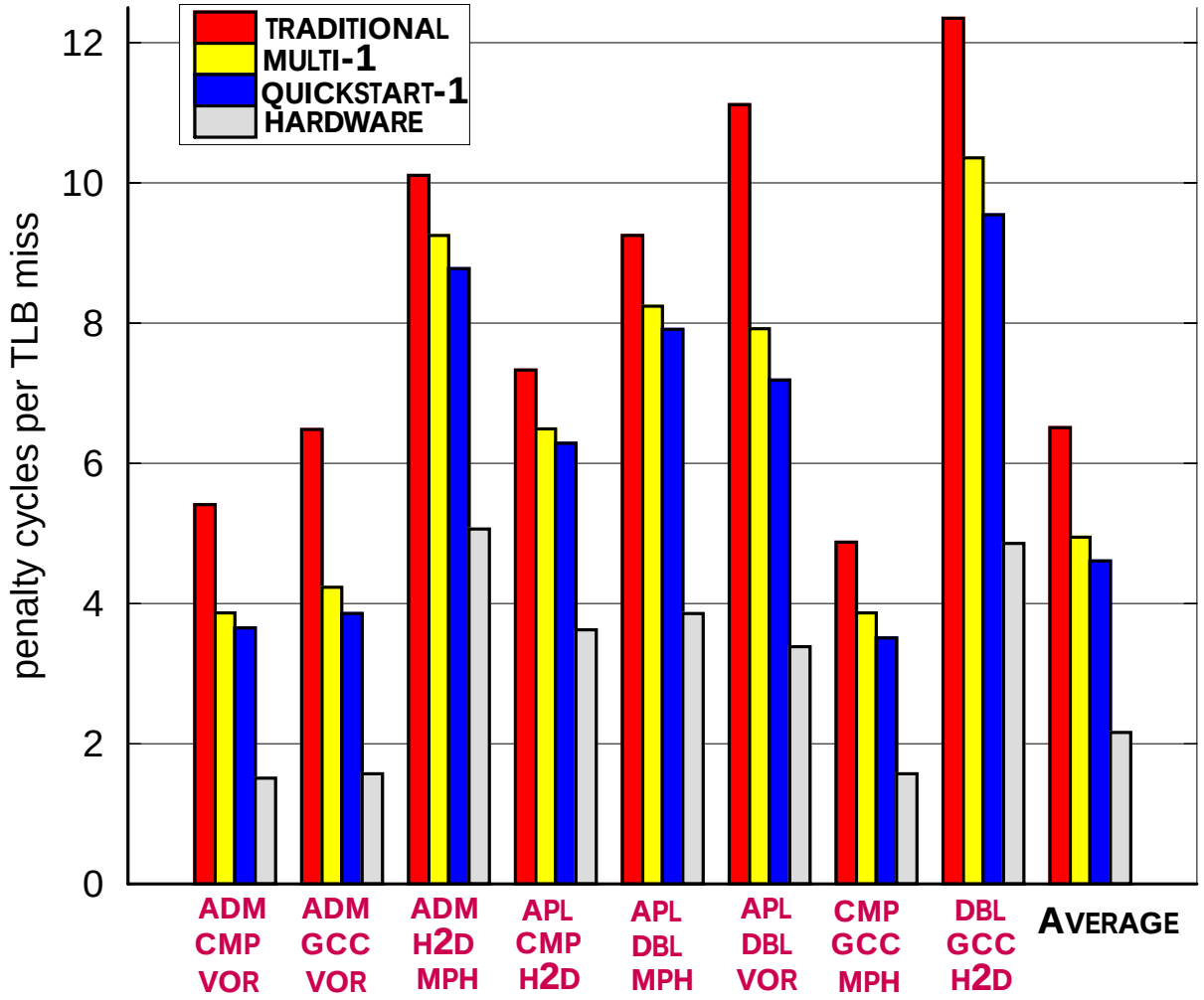
Multiprogramed/Multithreaded Workload:

Focus: Maximize throughput

Our experiment: (Not necessarily a fair comparison)

Run 3 applications, 1 idle thread for exception handling

Performance on Multiprogramed Workloads



PERFORMANCE IS MORE COMPLICATED

- **SMT IS MORE LATENCY TOLERANT**
- **SMT IS LESS TOLERANT OF WASTED BANDWIDTH**

Related Work

Architectures:

- M-Machine
  - Fillo, Keckler, Dally, Carter, Chang, Gurevich, Lee
  - Keckler, Dally, Chang, Lee, Chatterjee
- Multiscalar/Kestrel

Subordinate Multithreading:

- Chappel, Stark, Kim, Reinhart, and Patt
- Song and Dubois

Conclusions

SIGNIFICANTLY IMPROVES EXCEPTION HANDLING PERFORMANCE:

- *software TLB miss performance approaching that of an aggressive hardware TLB miss performance*

NOT ALL EXCEPTIONS CAN BE IMPLEMENTED IN HARDWARE

HIGH PERFORMANCE EXCEPTIONS ENABLE NOVEL SOFTWARE SYSTEMS

- *'a la* SOFTWARE **DSM** *or* CONCURRENT **GC**