

Data Speculation Support for a Chip Multiprocessor

Lance Hammond, Mark Willey, and Kunle Olukotun

Computer Systems Laboratory
Stanford University
<http://www-hydra.stanford.edu>

A Chip Multiprocessor

ASPLOS 1998

MOTIVATION

- Implementation benefits
 - High-speed signals localized within CPUs
 - Simple design replicated over die
 - ASPLOS-VII, IEEE Computer (9/97)
- Great when parallel threads are available
- Problem: Lack of parallel software!

Parallel Software

- Traditional auto-parallelization is limited
 - Works for dense matrix Fortran applications
- Many applications only hand-parallelizable
 - Parallelism exists in algorithm
 - One can't always statically verify parallelism
 - *Pointer disambiguation* is a major problem!
- Some applications just not parallelizable
 - True data dependencies may be present

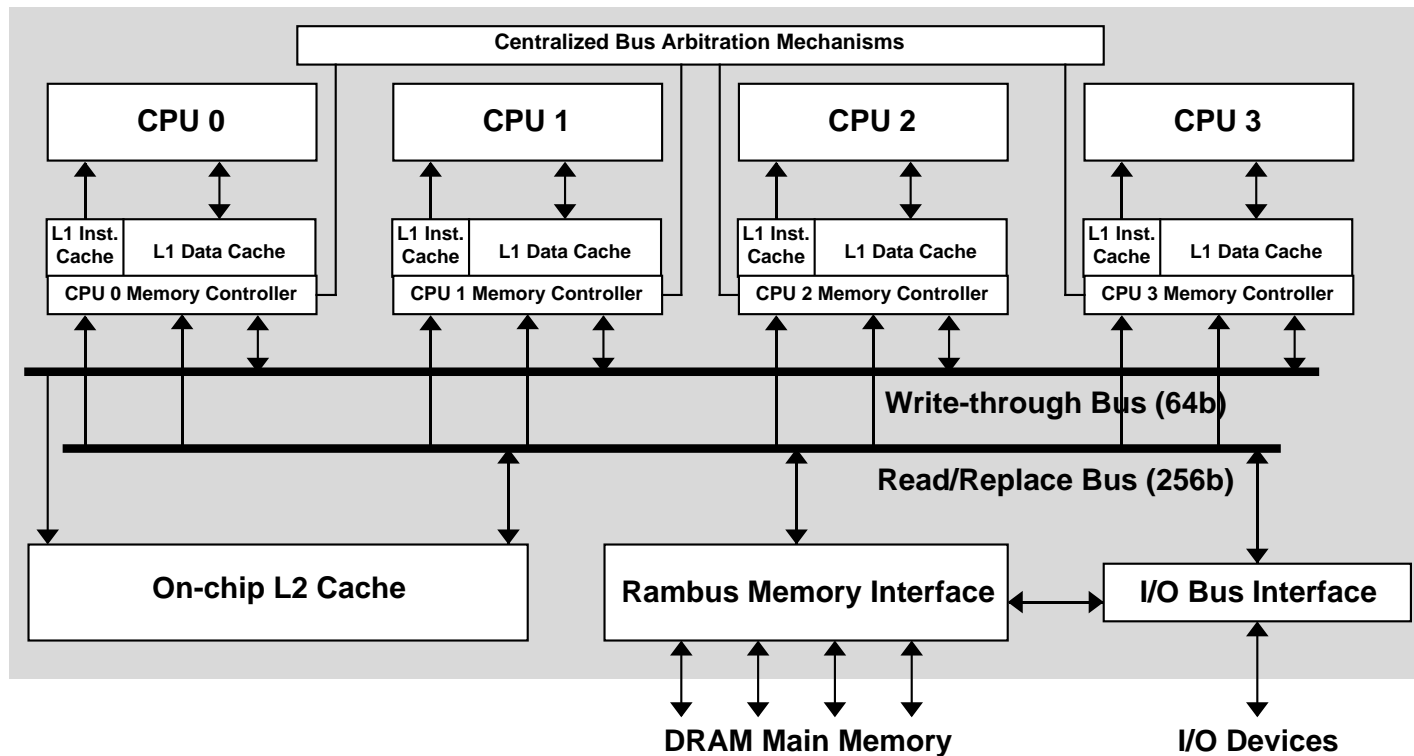
Data Speculation

- Previous work: Knight, Multiscalar, TLDS
- Eases compiler parallelization of loops
 - Hardware can protect against pointer aliasing
 - Synchronization isn't *required* for correctness
 - Parallelization of loops can easily be automated!
- Allows more ways to parallelize code
 - HW can break up code into “arbitrary” threads
 - One possibility: speculative subroutines

Hydra CMP

ASPLOS 1998

OUR SPECULATION DESIGN

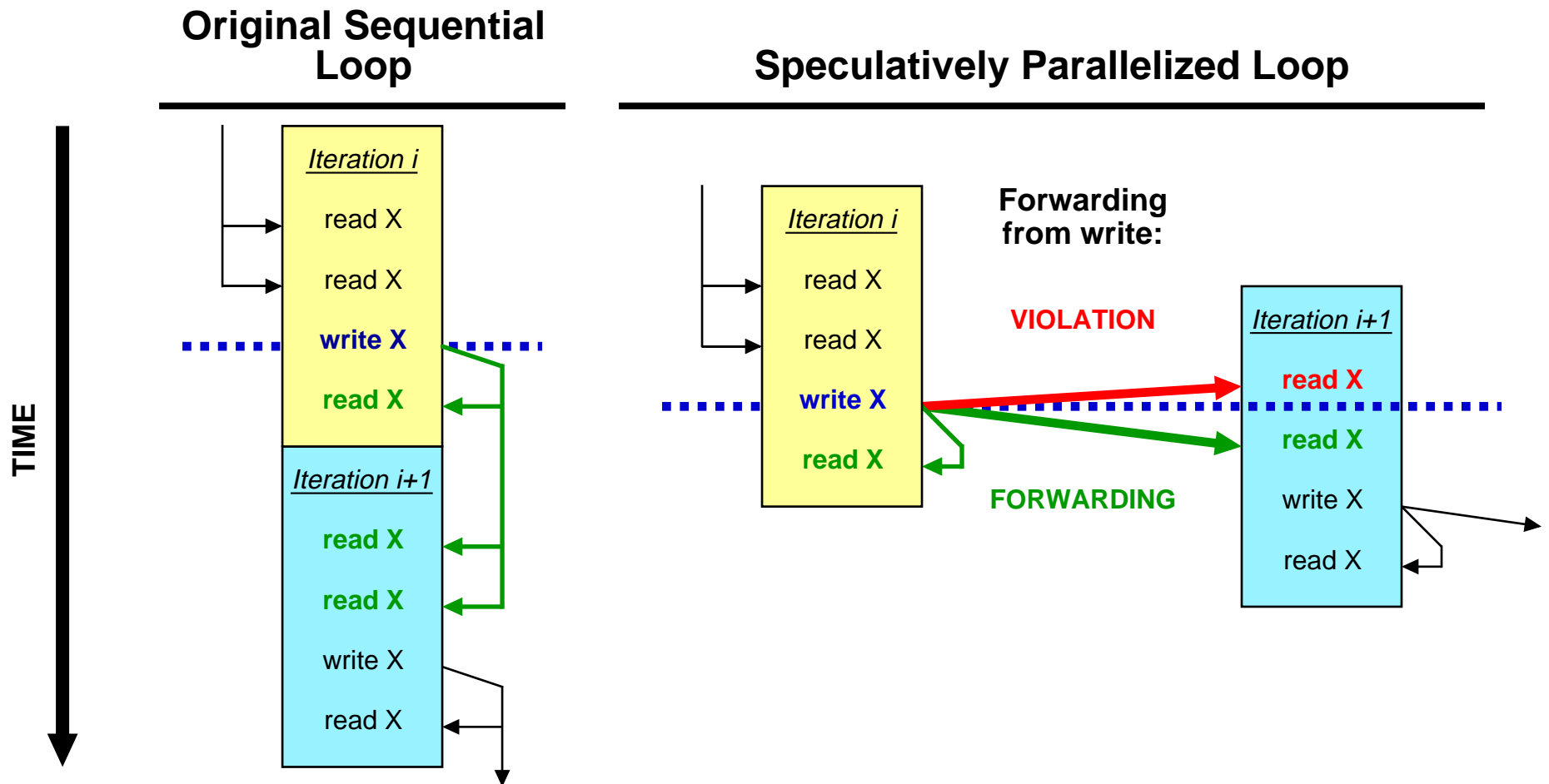


- 4 processors and secondary cache on a chip
- 2 buses connect processors and memory
- Coherence: writes are broadcast on write bus

Speculative Memory I

ASPLOS 1998

OUR SPECULATION DESIGN

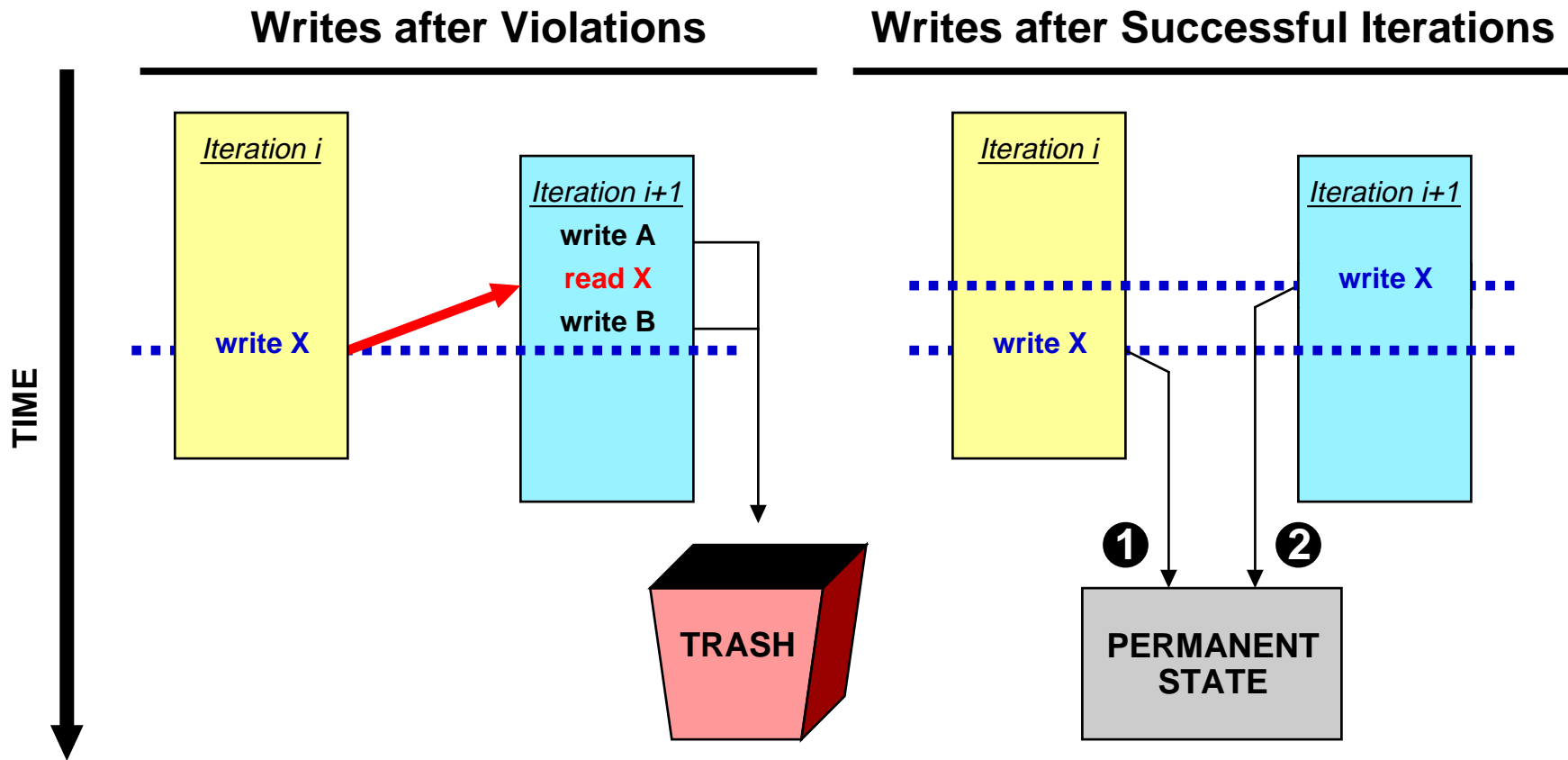


1. Forward data between threads
2. Detect violations

Speculative Memory II

ASPLOS 1998

OUR SPECULATION DESIGN

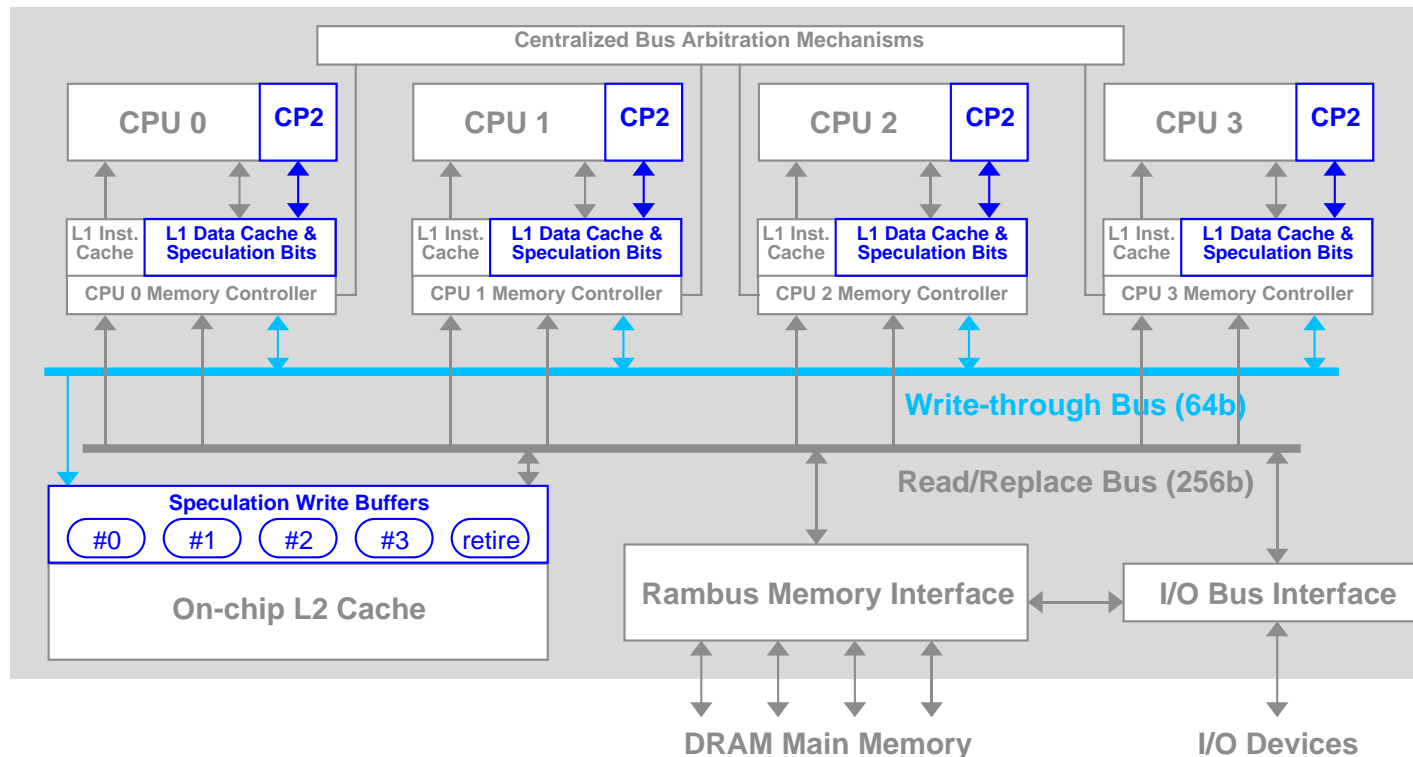


3. Safely back up after violations
4. Retire speculative state in the correct order

Hydra Speculation Support

ASPLOS 1998

OUR SPECULATION DESIGN



1. Write bus & L2 buffers provide forwarding
2. “Read” L1 tag bits detect violations
3. “Dirty” L1 bits & L2 buffers allow backup
4. L2 buffers reorder & retire speculative state

Speculative Threads

- Try to run post-subroutine code speculatively in parallel with subroutines
 - Requires return-value prediction
- Try to execute loop iterations in parallel
- Explicit synchronization still allowed during speculation
 - *Only* needed if it *helps* performance!

Speculation Control

ASPLOS 1998

OUR SPECULATION DESIGN

- Software control simplifies implementation
 - Hand-coded assembly for speed
 - Initiated with quick, vectored exceptions
- Essential control functions
 - Control L1 cache with coprocessor instructions
 - Send messages to other processors & L2 with stores
 - Pass registers through memory
 - Control speculation and value prediction logic
 - Manage threads (a small runtime system)
- Overheads
 - Subroutine control: 70-100 inst. at start and end
 - Loop control: 70 instructions at start and end
16 instructions per iteration

Simulation Methodology

ASPLOS 1998

SIMULATION RESULTS

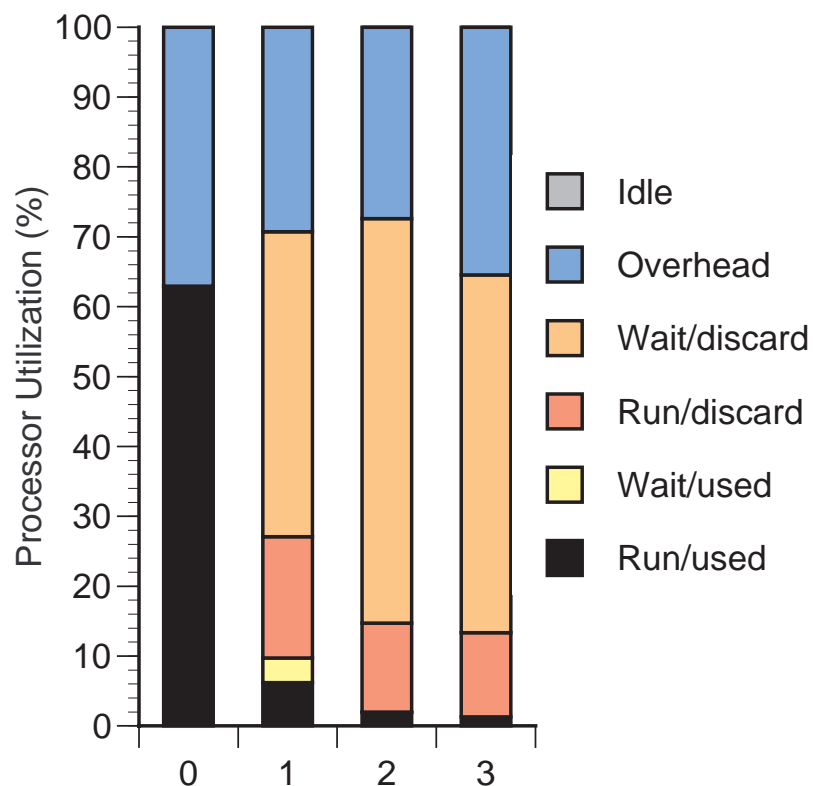
- A simple, automatic loop pre-translator
 - Only for speculative code
- Compiled with a commercial compiler
 - Using O2 optimization
- 4 single-issue pipelined MIPS processors
- Fully simulated memory system
 - 1 cycle instruction and data caches
 - 5+ cycle on-chip secondary cache

vortex Results

ASPLOS 1998

SIMULATION RESULTS

- Speedup: 0.58



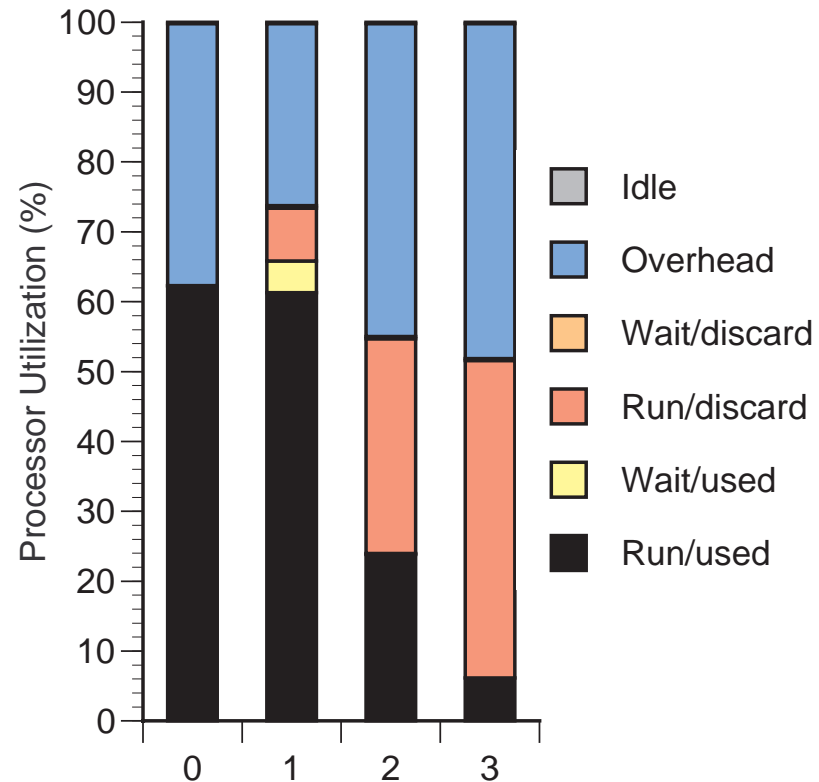
- Subroutine speculation is difficult
 - Lots of overhead in control routines
 - Load imbalance limits parallelism
 - Many subroutines are poor speculation targets

wc Results

ASPLOS 1998

SIMULATION RESULTS

- Speedup: 0.62
— 0.66 with extra delay



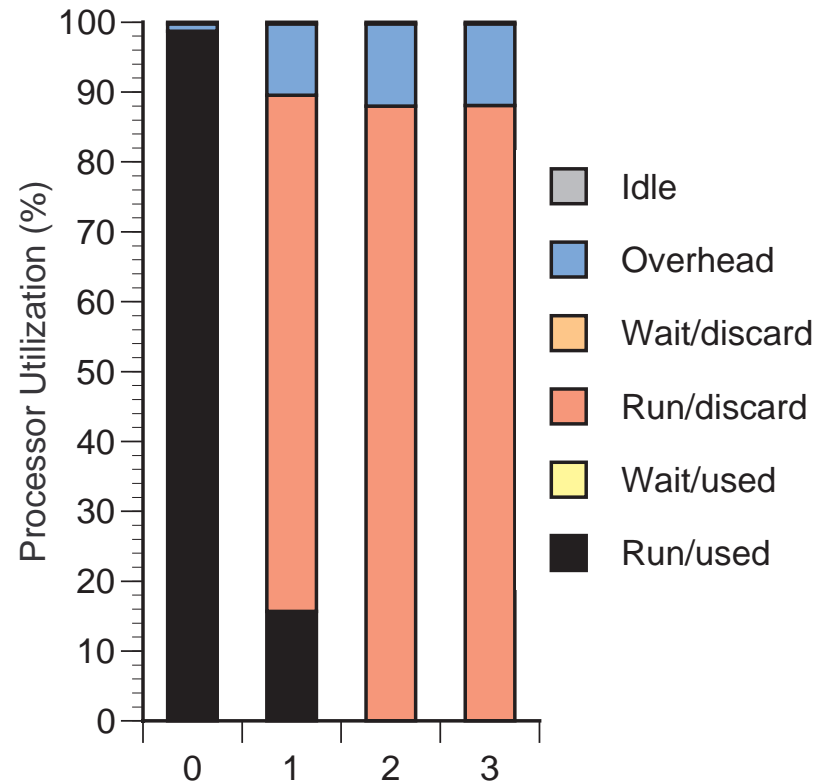
- Overheads can be significant (small regions)
 - Software control handlers
 - Additional load/store instructions needed
 - Interprocessor communication delays

m88ksim Results

ASPLOS 1998

SIMULATION RESULTS

- Speedup: 1.04



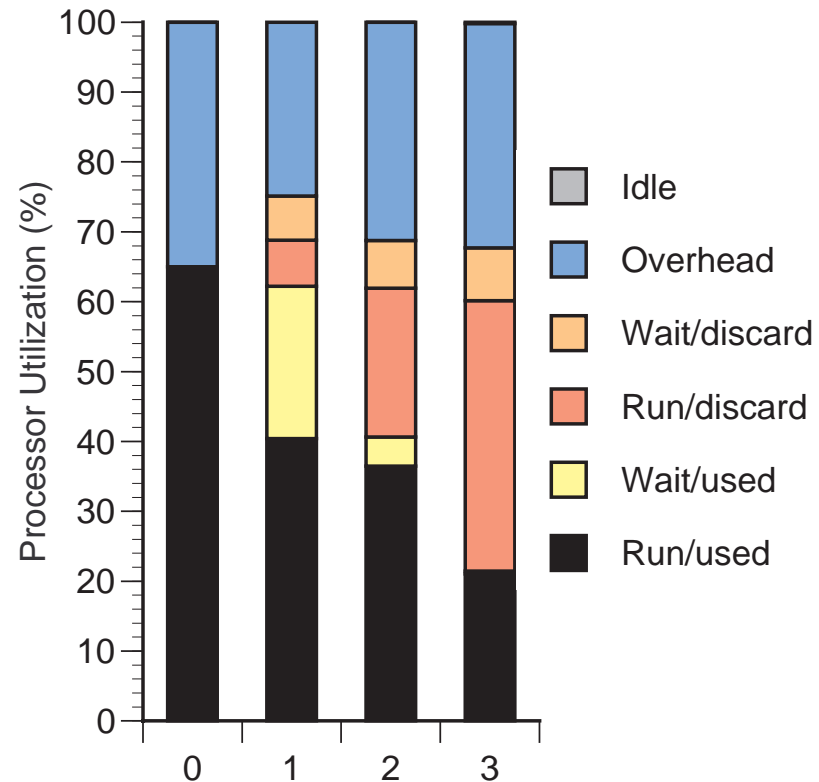
- Dependencies are a problem (large regions)
 - One dependency can force serialization

compress Results

ASPLOS 1998

SIMULATION RESULTS

- Speedup: 1.00
 - 1.09 with explicit synchronization



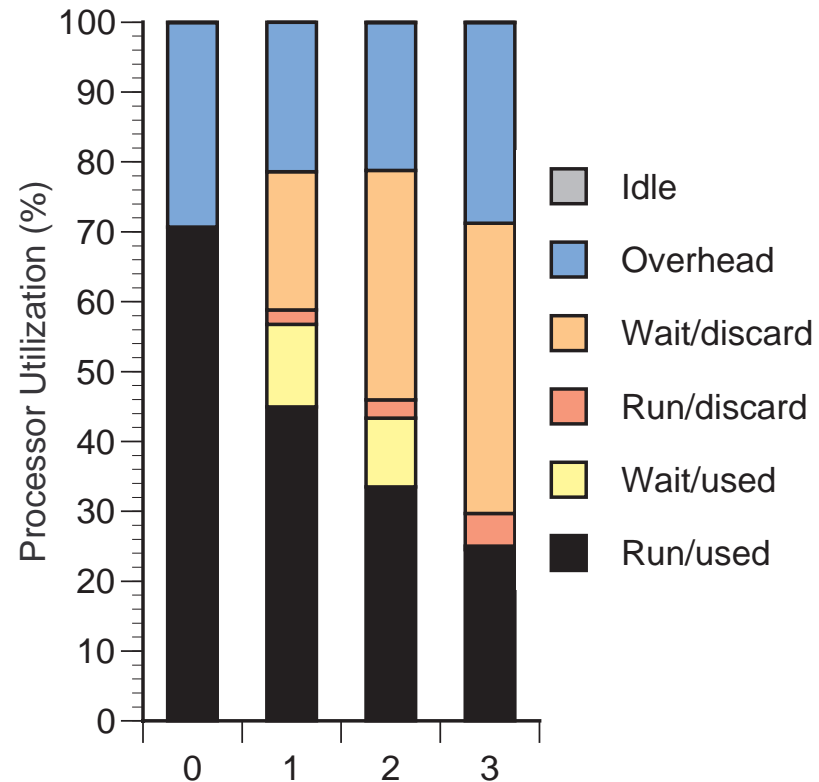
- Explicit synchronization can help
 - Key dependencies can be protected
 - But most synchronization may be omitted

ijpeg Results

ASPLOS 1998

SIMULATION RESULTS

- Speedup: 1.51



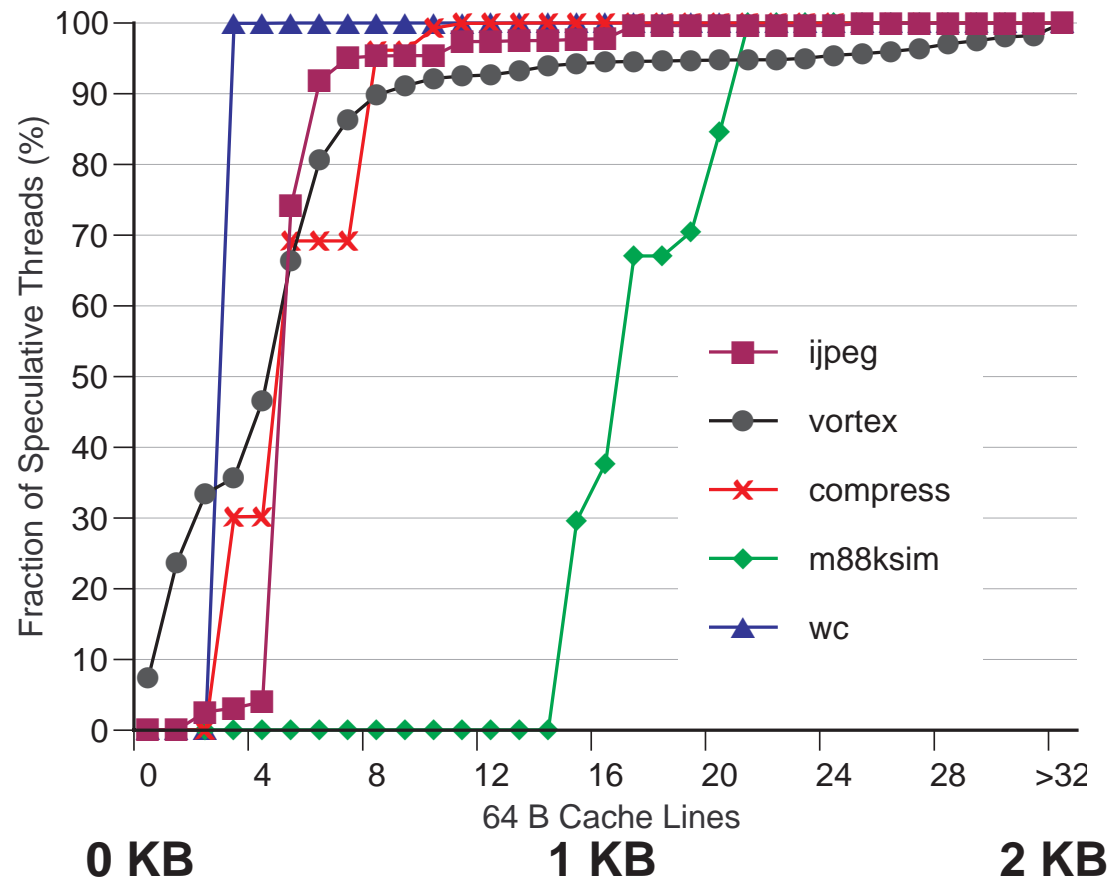
- Good speedup can occur on some loops
 - Reasonable size (250-2,500 instructions)
 - Limited dependencies = parallelism exists

L2 Data Buffering

ASPLOS 1998

SIMULATION RESULTS

- Small buffers are sufficient
 - We used a fully associative line buffer
 - 1 – 2 KB per thread captures most writes



Conclusions

- Reasonable cost/performance
 - Small gain, but small investment
 - Allows extraction of some parallelism that compilers can't normally find
 - Just turn off if limited by dependencies or grain size
- Normal MP performance not impacted
 - Multiprogrammed and explicitly-parallelized applications still get more speedup, when available
 - Flexible: can freely mix speculative and MP threads

Future Work

- Hardware modifications
 - Update data cache protocol
 - Special instructions to lower overhead
 - Hardware thread control
- Compiler analysis of routines
 - “Pruning” of poor routines at compile time
 - Profile directed compilation
- Compiler control of variable access (TLDS)
 - Move loads from shared variables as late as possible
 - Move stores to shared variables as early as possible
 - Adding synchronization where useful