Single-Chip Multiprocessors: the Rebirth of Parallel Architecture

Guri Sohi

University of Wisconsin

Outline

- Waves of innovation in architecture
- Innovation in uniprocessors
- Lessons from uniprocessors
- Future chip multiprocessor architectures
- Software and such things

Waves of Research and Innovation

- A new direction is proposed or new opportunity becomes available
- The center of gravity of the research community shifts to that direction
  - SIMD architectures in the 1960s
  - HLL computer architectures in the 1970s
  - RISC architectures in the early 1980s
  - Shared-memory MPs in the late 1980s
  - OOO speculative execution processors in the 1990s

THE UNIVERSITY of WISCONSIN MADISON

Waves

- Wave is especially strong when coupled with a “step function” change in technology
  - Integration of a processor on a single chip
  - Integration of a multiprocessor on a chip

THE UNIVERSITY of WISCONSIN MADISON

Uniprocessor Innovation Wave: Part 1

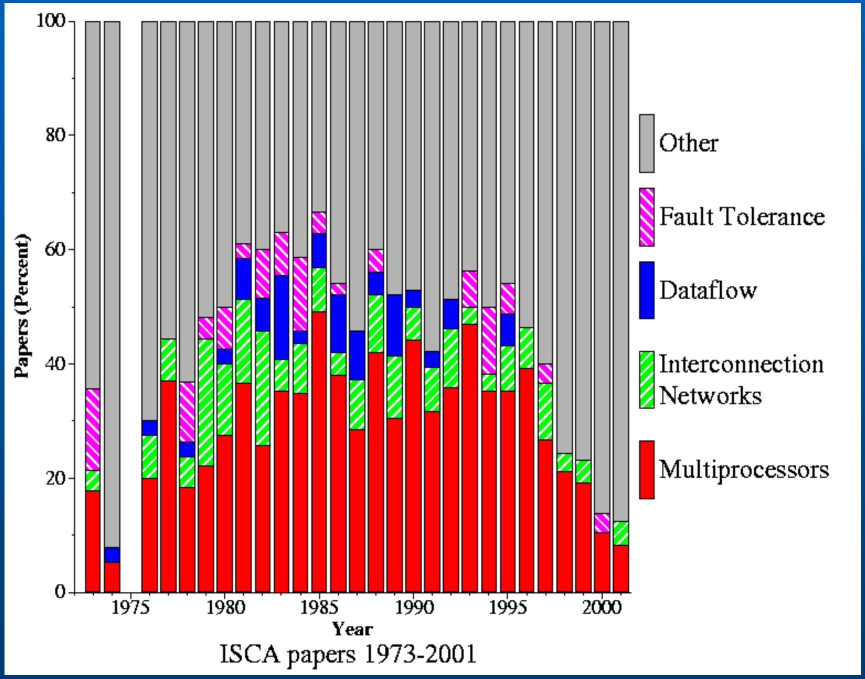
- Many years of multi-chip implementations
  - Generally microprogrammed control
  - Major research topics: microprogramming, pipelining, quantitative measures
- Significant research in multiprocessors

Uniprocessor Innovation Wave: Part 2

- Integration of processor on a single chip
  - The inflexion point
  - Argued for different architecture (RISC)
- More transistors allow for different models
  - Speculative execution
- Then the rebirth of uniprocessors
  - Continue the journey of innovation
  - Totally rethink uniprocessor microarchitecture
  - Jim Keller: “Golden Age of Microarchitecture”

Uniprocessor Innovation Wave: Results

- Current uniprocessor very different from 1980's uniprocessor
- Uniprocessor research dominates conferences
- MICRO comes back from the dead
  - Top 1% (NEC Citeseer)
- Impact on compilers



ISCA papers 1973-2001

Source: Rajwar and Hill, 2001

Why Uniprocessor Innovation Wave?

- Innovation needed to happen
  - Alternatives (multiprocessors) not practical option for using additional transistors
- Innovation could happen: things could be done differently
  - Identify barriers (e.g., to performance)
  - Use transistors to overcome barriers (e.g., via speculation)
  - Simulation tools facilitate innovation

Lessons from Uniprocessors

- Don’t underestimate what can be done in hardware
  - Doing things in software was considered easy; in hardware considered hard
  - Now perhaps the opposite
- Barriers or limits become opportunities for innovation
  - Via novel forms of speculation
  - E.g., barriers in Wall’s study on limits of ILP

Multiprocessor Architecture

- A.k.a. “multiarchitecture” of a multiprocessor
- Take state-of-the-art uniprocessor
- Connect several together with a suitable network
  - Have to live with defined interfaces
- Expend hardware to provide cache coherence and streamline inter-node communication
  - Have to live with defined interfaces

Software Responsibilities

- Have software figure out how to use MP
- Reason about parallelism
- Reason about execution times and overheads
- Orchestrate parallel execution
- Very difficult for software to parallelize transparently

Explicit Parallel Programming

- Have programmer express parallelism
- Reasoning about parallelism is hard
- Use synchronization to ease reasoning
  - Parallel trends towards serial with the use of synchronization

Net Result

- Difficult to get parallelism speedup
  - Computation is serial
  - Inter-node communication latencies exacerbate problem
- Multiprocessors rarely used for parallel execution
- Used to run threaded programs
  - Lower-overhead sync would help
- Used to improve throughput

The Inflexion Point for Multiprocessors

- Can put a basic small-scale MP on a chip
- Can think of alternative ways of building multiarchitecture
  - Don’t have to work with defined interfaces!
- What opportunities does this open up?
  - Allows for parallelism to get performance.
  - Allows for use of novel techniques to overcome (software and hardware) barriers
  - Other opportunities (e.g., reliability)

Parallel Software

- Needs to be compelling reason to have a parallel application
- Won’t happen if difficult to create
  - Written by programmer or automatically parallelized by compiler
- Won’t happen if insufficient performance gain

Changes in MP Multiarchitecture

- Inventing new functionality to overcome barriers
  - Consider barriers as opportunities
- Developing new models for using CMPs
- Revisiting traditional use of MPs

Speculative Multithreading

- Speculatively parallelize an application
  - Use speculation to overcome ambiguous dependences
  - Use hardware support to recover from mis-speculation
- E.g., multiscalar
- Use hardware to overcome limitations

Overcoming Barriers: Memory Models

- Weak models proposed to overcome performance limitations of SC
- Speculation used to overcome “maybe” dependences
- Series of papers showing SC can achieve performance of weak models

Implications

- Strong memory models not necessarily low performance
- Programmer does not have to reason about weak models
- More likely to have parallel programs written

Overcoming Barriers: Synchronization

- Synchronization to avoid “maybe” dependences
  - Causes serialization
- Speculate to overcome serialization
- Recent work on techniques to dynamically elide synchronization constructs

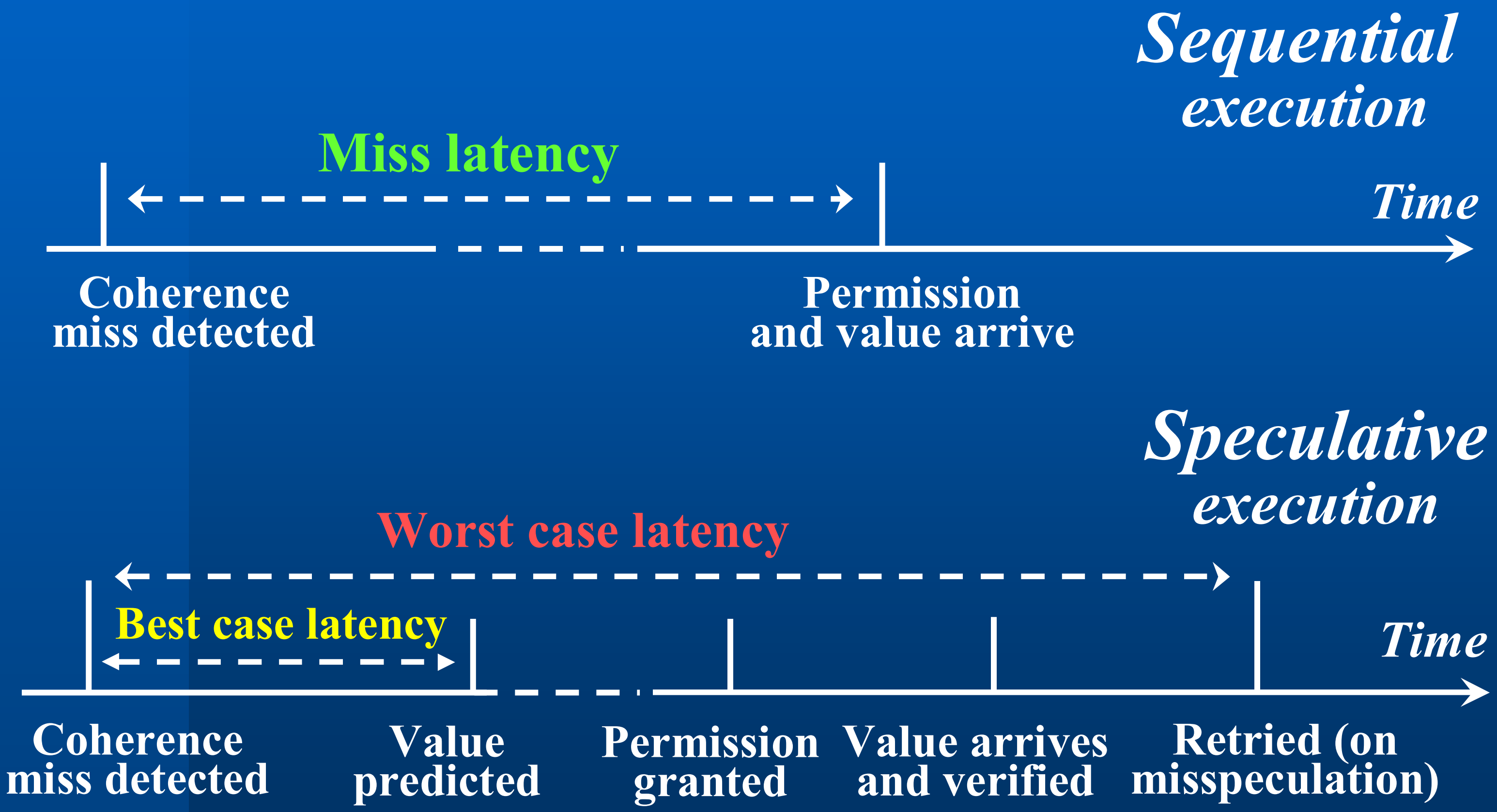
Implications

- Programmer can make liberal use of synchronization to ease programming
- Little performance impact of synchronization
- More likely to have parallel programs written

Overcoming Barriers: Coherence

- Caches used to reduce data access latency; need to be kept coherent
- Latencies of getting value from remote location impact performance
- Getting remote value is two-part operation
  - Get value
  - Get permissions to use value
  - Can separating these help?

Coherence Decoupling



Zeros/Ones in Coherence Misses

Load Value	**1**	**0**	**-1**	**Total**
OLTP	5.1%	32.3%	9.9%	47.3%
Apache	0.7%	27.8%	0.5%	29.0%
JBB	18.6%	12.6%	0.5%	31.7%
Barnes	1.7%	34.1%	1.7%	37.5%
Ocean	1.3%	25.9%	1.7%	28.9%
Store Value	**1**	**0**	**-1**	**Total**
OLTP	7.1%	29.1%	13.1%	49.4%
Apache	3.4%	17.0%	7.4%	27.7%
JBB	0.7%	21.3%	1.4%	23.4%
Barnes	1.8%	29.8%	17.2%	48.7%
Ocean	13.2%	4.9%	3.5%	21.5%

Preliminary Data, Simics (Ultrasparc/Solaris, 16P),
Cache (4MB 4-way SA L2, 64B lines, MOSI)

Other Performance Optimizations

- Clever techniques for inter-processor communication
  - Remember: no artificial constraints on chip
- Further reduction of artificial serialization

New Uses for CMPs

- Helper threads, redundant execution, etc.
  - will need extensive research in the context of CMPs
- How about trying to parallelize application, i.e., “traditional” use of MPs?

Revisiting Traditional Use of MPs

- Compilers and software for MPs
- Digression: Master/Slave Speculative Parallelization (MSSP)
- Expectations for future software
- Implications

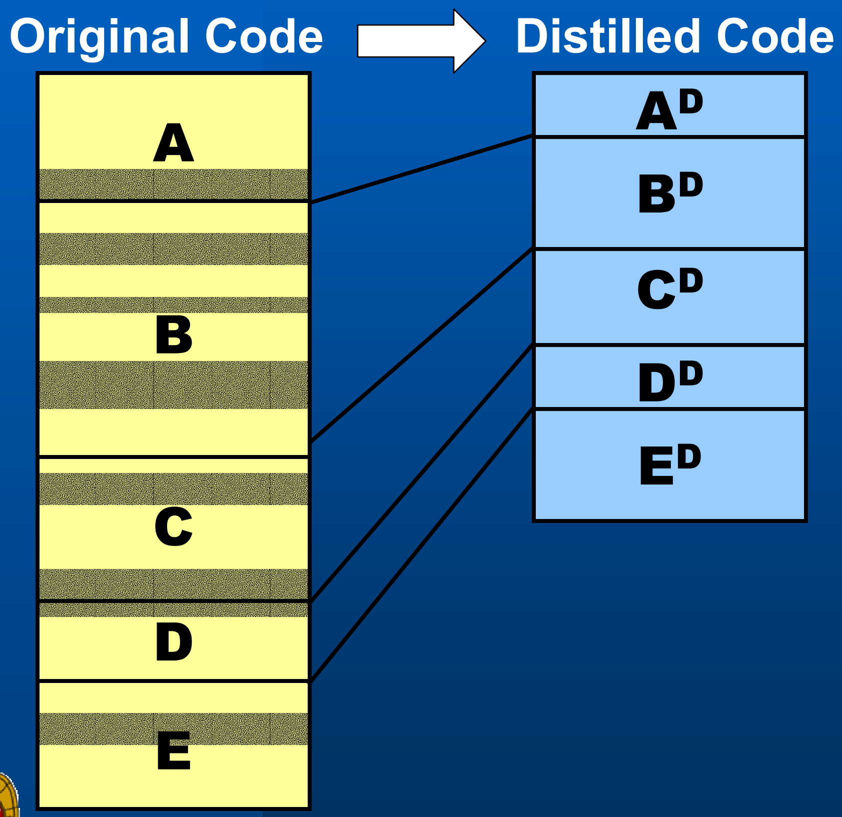
Parallelizing Apps: A Moving Target

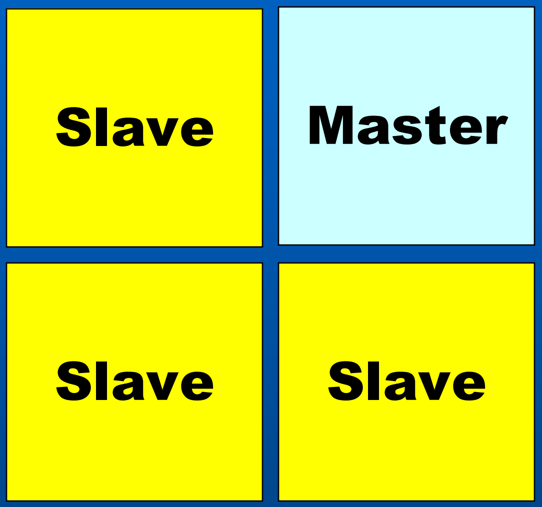
- Learned to reason about certain languages, data structures, programming constructs and applications
- Newer languages, data structures, programming constructs and applications appear
- Always playing catch up
- Can we get a jump ahead?

Master/Slave Speculative Parallelization (MSSP)

- Take a program and create program with two sub-programs: Master and Slave
- Master program is approximate (or distilled) version of original program
- Slave (original) program "checks" work done by master
- Portions of the slave program execute in parallel

MSSP - Overview



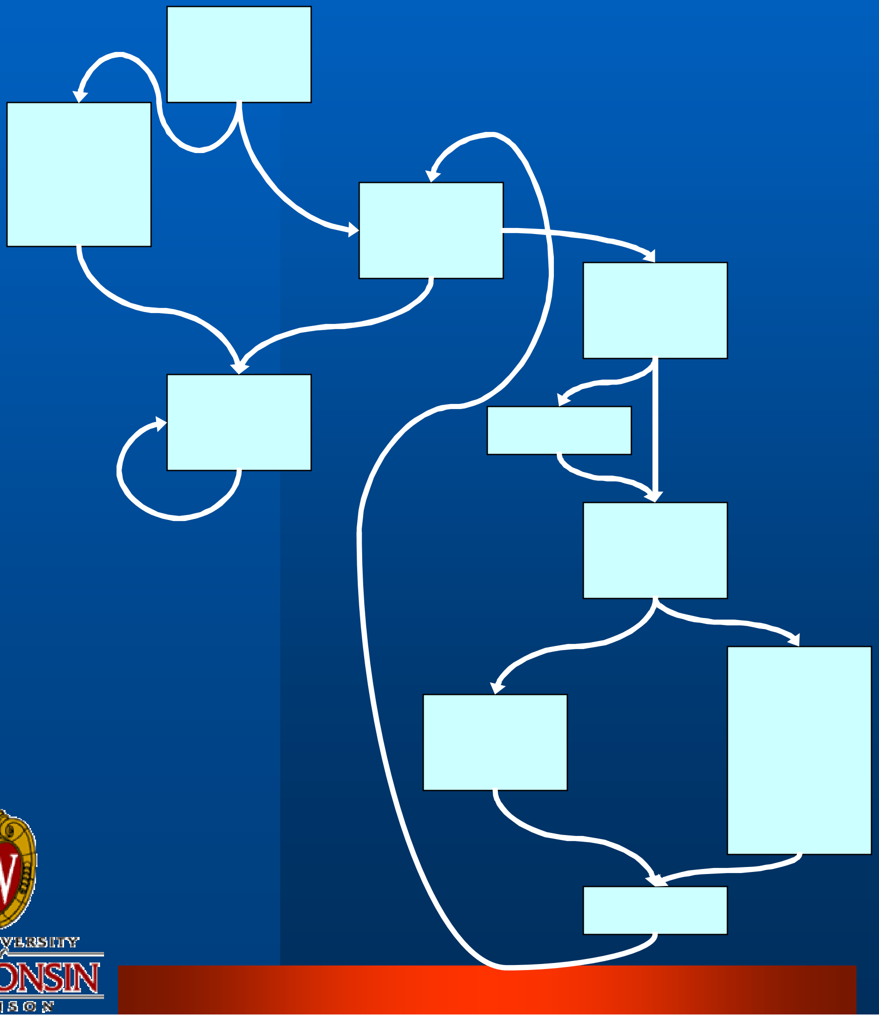


Distilled Code on Master

Original Code concurrently on Slaves verifies Distilled Code

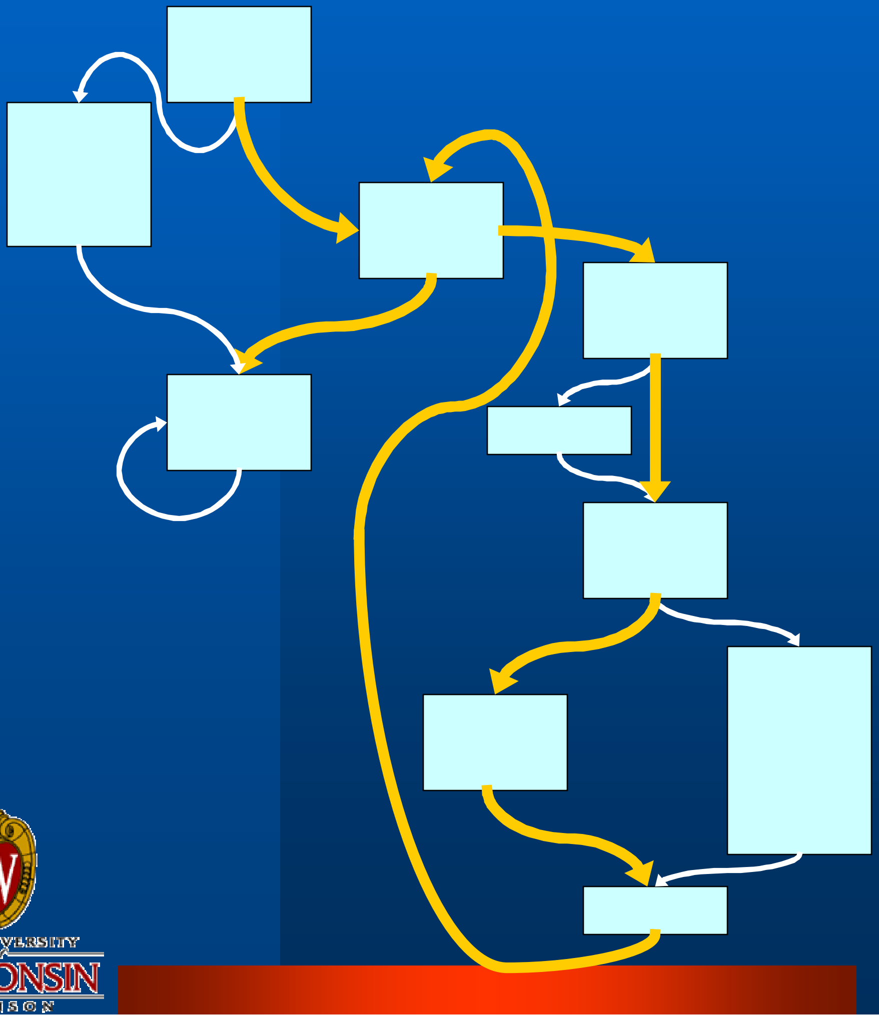
Use checkpoints to communicate changes

MSSP - Distiller



Program with many paths

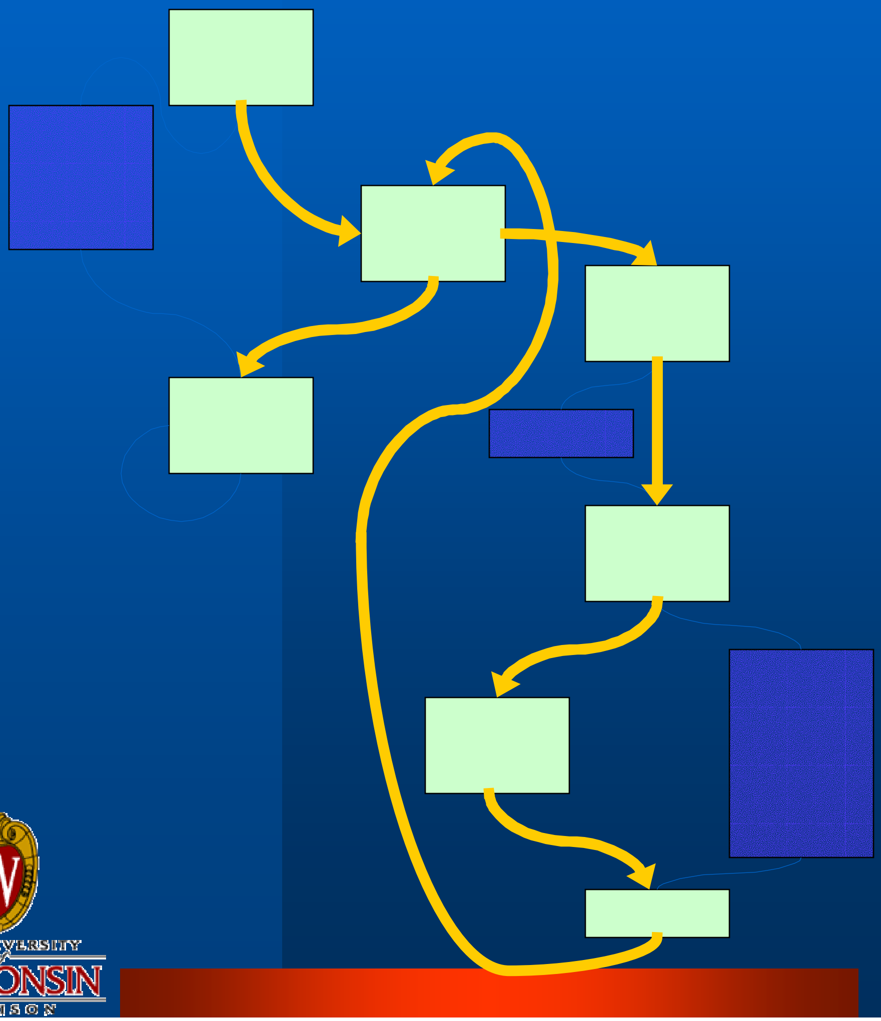
MSSP - Distiller



Program with many paths

Dominant paths

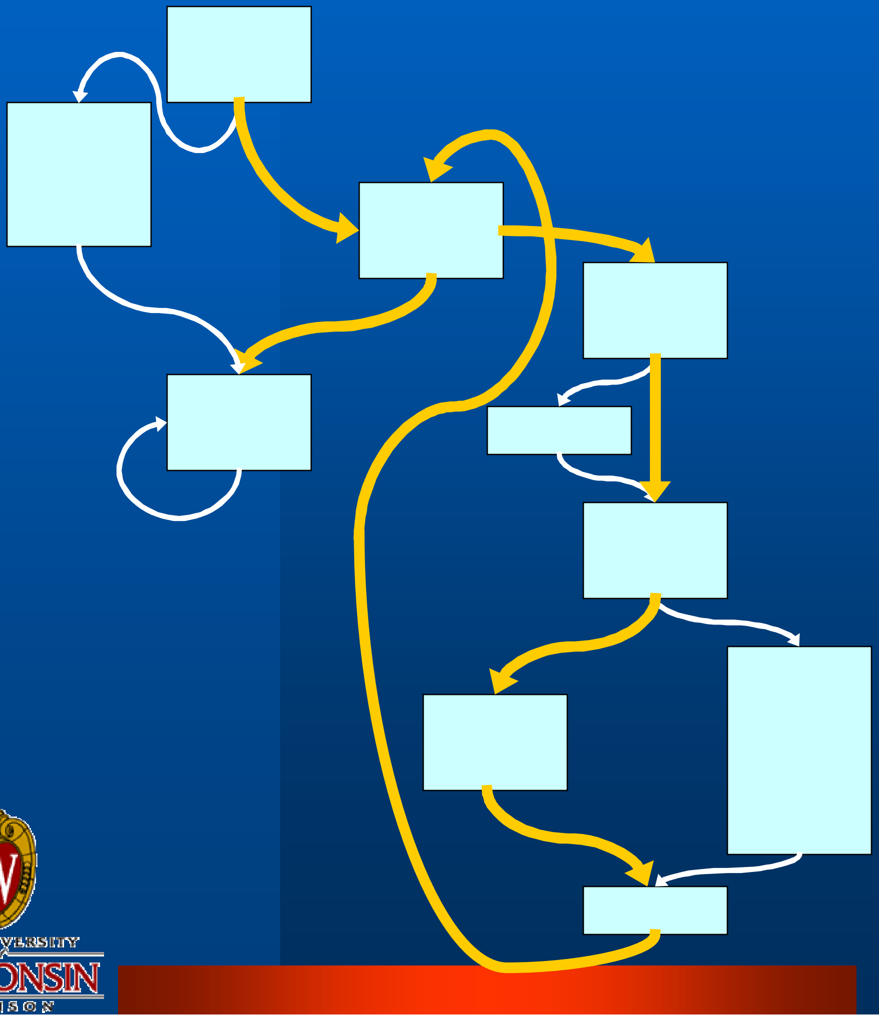
MSSP - Distiller



Program with many paths

Dominant paths

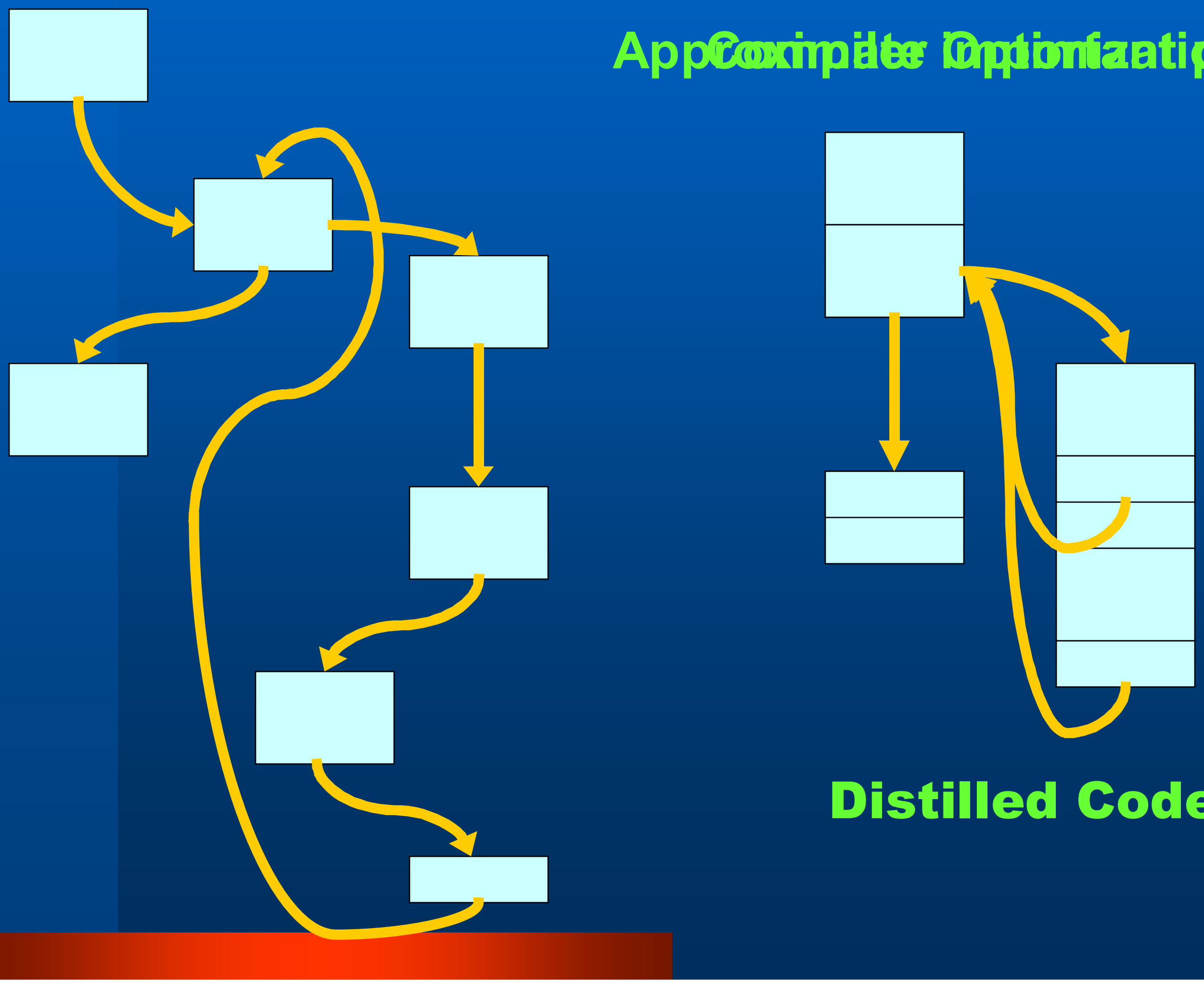
MSSP - Distiller



Program with many paths

Dominant paths

MSSP - Distiller

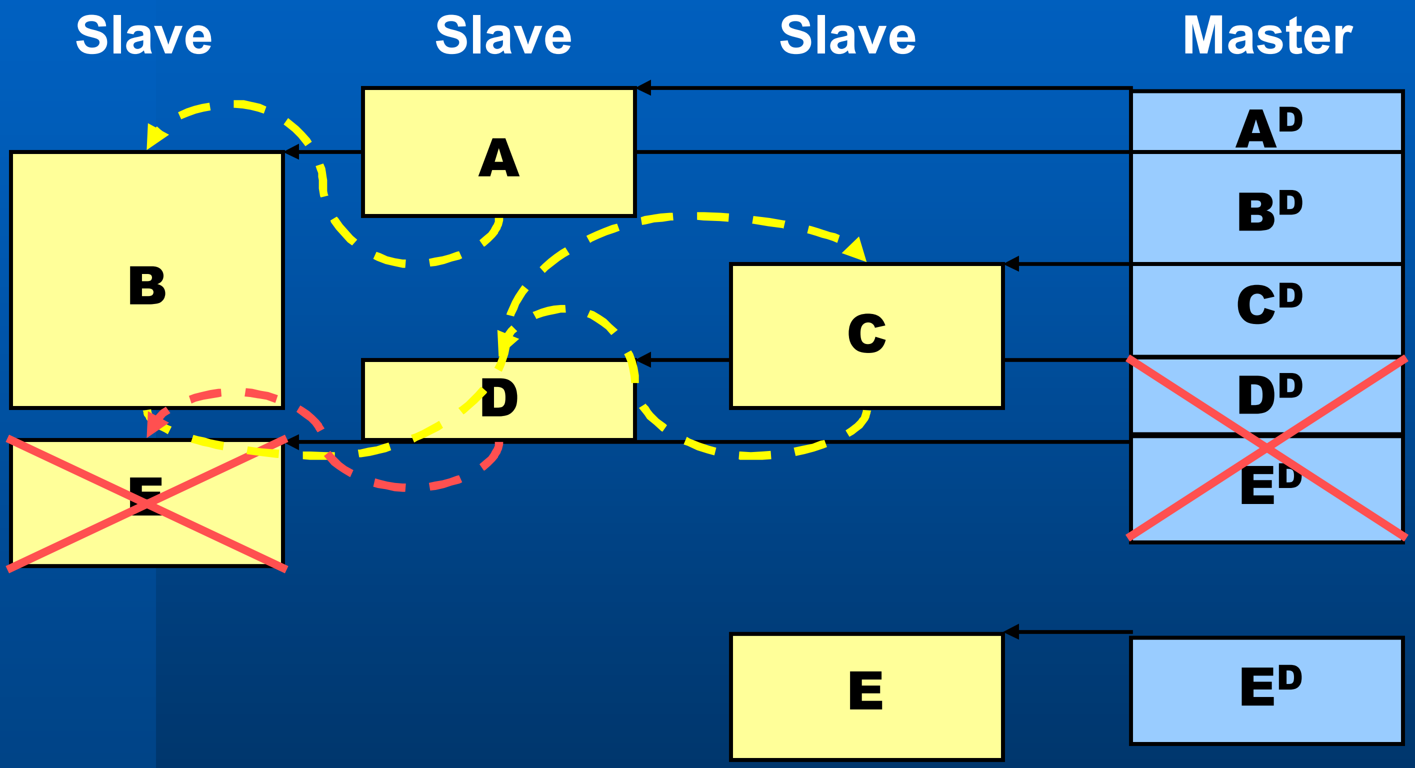


Approximate paths

Compiler Optimizations

Distilled Code

MSSP - Execution



Verify checkpoint of D^E Correct Wrong

MSSP Summary

- Distill away code that is unlikely to impact state used by later portions of program
- Performance tracks distillation ratio
  - Better distillation = better performance
- Verification of distilled program done in parallel on slaves

Future Applications and Software

- What will future applications look like?
  - Don’t know
- What language will they be written in?
  - Don’t know; don’t care
- Code for future applications will have “overheads”
  - Overheads for checking for correctness
  - Overheads for improving reliability
  - Overheads for checking security

THE UNIVERSITY of WISCONSIN MADISON

Overheads as an Opportunity

- Performance costs of overhead have limited their use
- Overheads not a limit; rather an opportunity
- Run overhead code in parallel with non-overhead code
  - Develop programming models
  - Develop parallel execution models (a la MSSP)
  - Recent work in this direction
- Success at reducing overhead cost will encourage even more use of "overhead" techniques

Summary

- New opportunities for innovation in MPs
- Expect little resemblance between MPs today and CMPs in 15 years
  - We need to invent and define differences
  - Not because uniprocessors are running out of steam
  - But because innovation in CMP multiarchitecture possible

Summary

- Novel techniques for attacking performance limitations
- New models for expressing work (computation and overhead)
- New parallel processing models
- Simulation tools