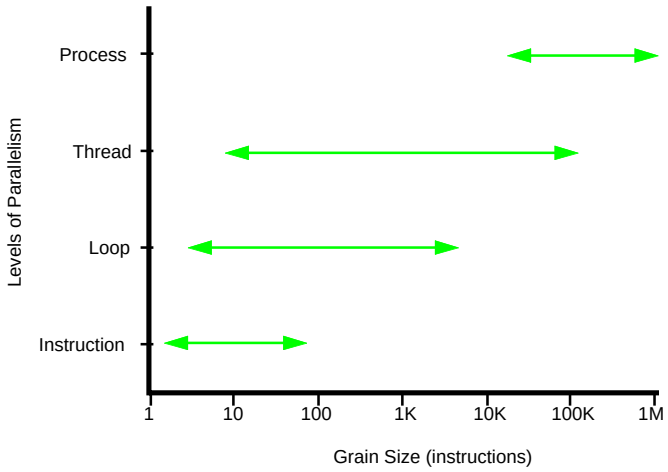
A New Approach to Speculation in the Hydra Single-Chip Multiprocessor

Kunle Olukotun
Computer Systems Laboratory
Stanford University

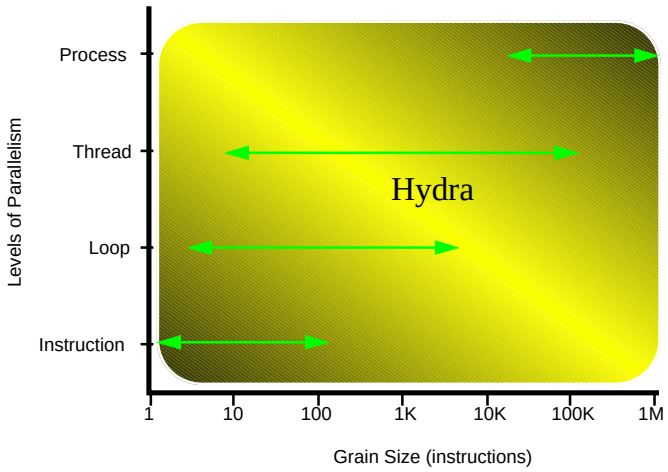
Program Parallelism



Hydra Approach

- Exploit all levels of program parallelism
- Develop a single-chip multiprocessor architecture that simplifies microprocessor design and achieves high performance
- Make the multiprocessor transparent to the average user
- Integrate use of parallelizing compiler technology in the design of microarchitecture

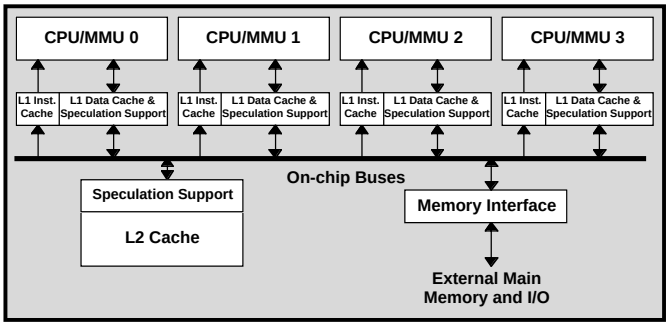
Goal of Hydra



Outline

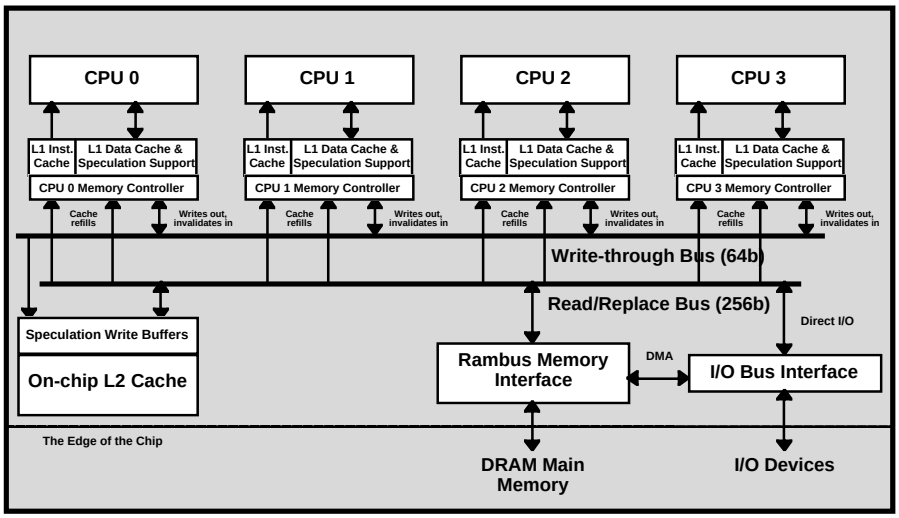
- Hydra overview
- Thread-level speculation
- Software support for speculation
- Hardware support for specualtion
- Preliminary performance of speculation
- Conclusions

The Hydra Design



- **Single-chip multiprocessor**
- **Four 2-way issue processors**
- **500 MHz ⇒ 4 gigaops**
- **Separate primary caches**
- **Write-through data caches to maintain cache coherence**
- **Shared 2nd-level cache**
- **Low latency interprocessor communication (10 cycles)**
- **Support for thread-level speculation**

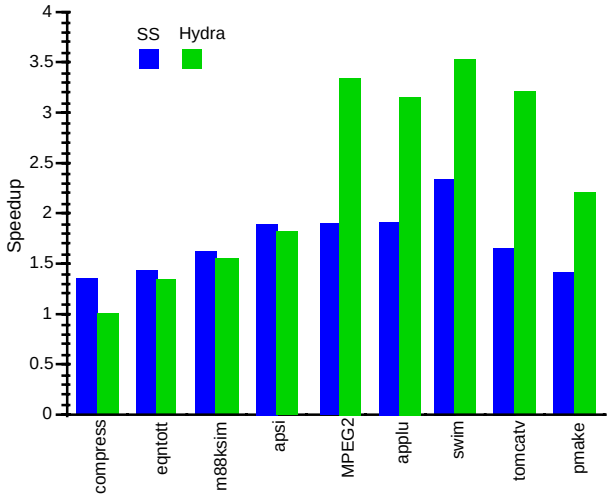
The Hydra Design (Details)



Cache Hierarchy Details

- **4 sets of single-ported L1 caches**
  - **Single cycle access**
  - **4-way Set Associative, writethrough**
  - **16K Instruction, 16K Data**
- **Shared, single-ported on-chip L2 cache**
  - **Fully pipelined, 2-cycle array time (~5 cycle access)**
  - **4-way Set Associative, writeback**
  - **512 KB, 32-byte lines**
- **Two data buses**
  - **Line-wide read bus for most purposes**
  - **Doubleword-wide write bus for writethroughs**

Hydra vs. Superscalar (SS)



- **ILP only**
  ⇒ SS 30% better than single Hydra processor
- **ILP & fine thread**
  ⇒ SS and Hydra comparable
- **ILP & coarse thread**
  ⇒ Hydra 1.5–2× better

Thread-Level Speculation

- Automatic parallelization of loops in C-programs is very difficult
  - **Loop iterations have data dependencies**
  - **Pointer disambiguation is difficult and expensive**
  - **Compile time analysis is too conservative**
- Speculation enables parallelization without regard for data-dependencies
  - **Oblivious loop parallelization**
  - **Speculation hardware ensures correctness**
  - **Add synchronization only for performance**

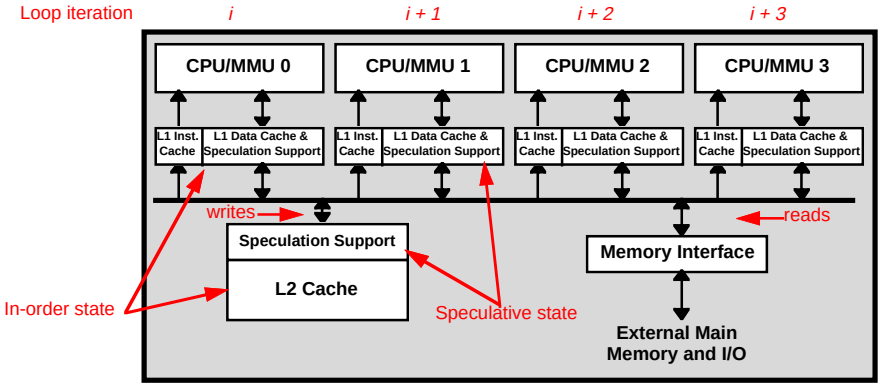
Speculation: Key Questions

- Can it expose more parallelism?
  - **sparse numerical applications**
  - **nonnumerical applications**
- What support mechanisms are required?
  - **software**
  - **hardware**
- How to implement hardware mechanisms cheaply?
  - **Sohi's multiscalar ARB approach is expensive**

Overview of Speculation

- Parallel regions (loops) are annotated by the compiler
- The hardware uses these annotations to run loop iterations in parallel
- Each CPU knows which loop iteration it is running
- CPUs dynamically prevent data dependency violations
  - **"later" iterations can't use data before write by "earlier" iterations (RAW)**
  - **"earlier" iterations never see writes by "later" iterations (WAW)**
- If a "later" iteration has used data that an "earlier" iteration writes, it is restarted
  - **all following iterations are halted and restarted, also**
  - **all writes by the later iteration are discarded**

SCMP with Speculation Support



One processor is always nonspeculative

- This is the "earliest" iteration currently running
- This iteration cannot violate, so it can never be restarted
- Makes OS interaction easier

Speculation Operations

Operations	Initiated by	Semantics
Begin_Speculation	Software	Start processor in speculative mode of execution
Map	software	Map CPU to sequence # Begin buffering speculative state
Read	Software	Read from speculative and in-order state Mark data that is read
Write	Software	Write to primary cache and speculative buffer Check for RAW hazards
Restart	Hardware	Discard speculative buffer for all future threads Restart this thread and all future threads
Commit	Software	Wait for thread to become in-order thread Copy speculative state to in-order state
End_Speculation	Software	Commit Discard speculative state for all future threads Stop speculation mode for all processors

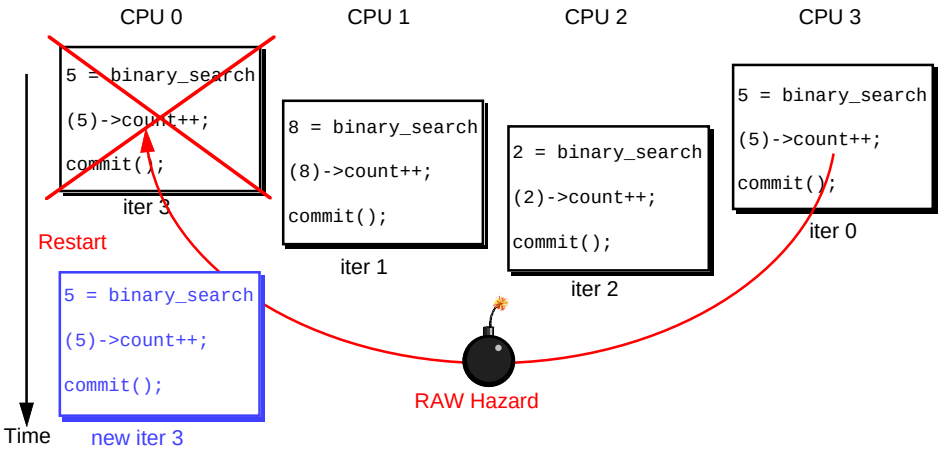
Compiling for Speculation

```
main()          /* count C keywords */
{
        ...
        while (getword(word) != EOF)
                if ((p = binary_search(word, keytab) >= 0)
                        p->count++;
        ...
}
```

Loop parallelization by SUIF compiler

```
begin_speculation();
while (1) {
    map(seq_num), seq_num += 4;
    if (getword(word) != EOF) {
        if ((p = binary_search(word, keytab) >= 0)
            p->count++;
        commit();
    }
    else {
        end_speculation();
        break;
    }
}
```

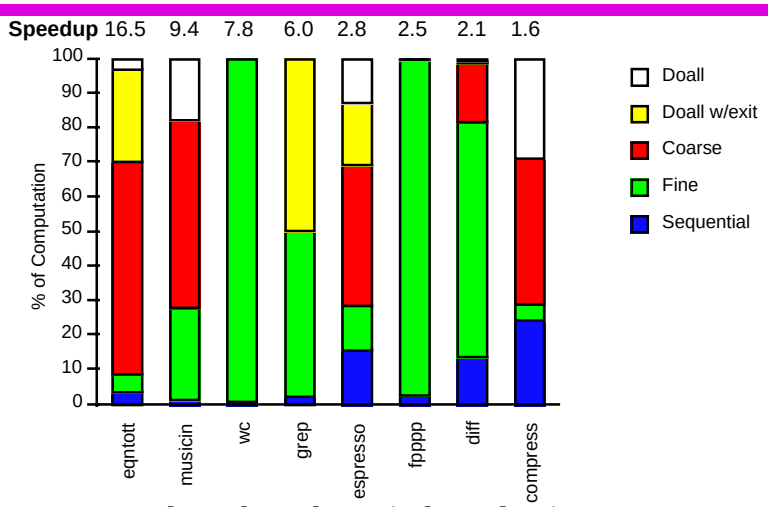

Executing with Speculation



Speculation Performance Potential

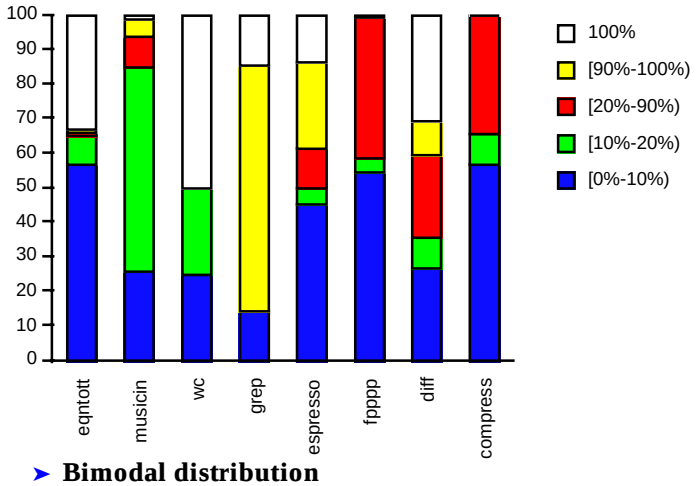
- Perfect speculation machine
  - **Infinite number of threads**
  - **Speculates on one loop at a time**
  - **Chooses best loop to speculate dynamically**
  - **No restarts, reads are delayed by minimum amount**
- Trace-driven simulation

Speculative Execution Modes



- **Most loops have dynamic dependencies**
- **Speculation hardware is needed**

Dependence Frequency



- **Bimodal distribution**
- **Synchronize high frequency dependencies**

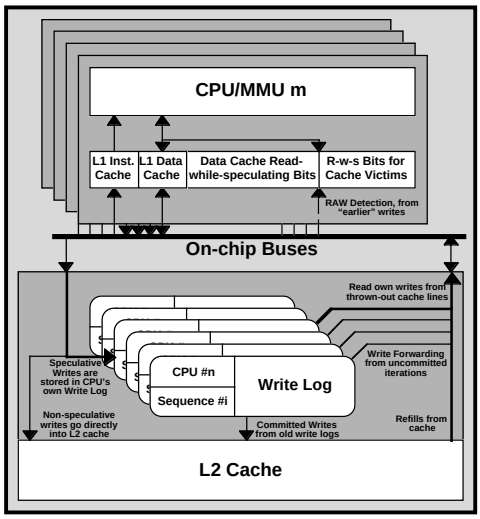
Compiler Optimizations

- Compile time analysis
  - **Code scheduling**
  - **Induction variable elimination**
- Use dynamic information to guide further compiler optimizations
  - **Where to speculate?**
  - **Where to synchronize?**
- Integrated in the SUIF compiler system

Hardware Support for Speculation

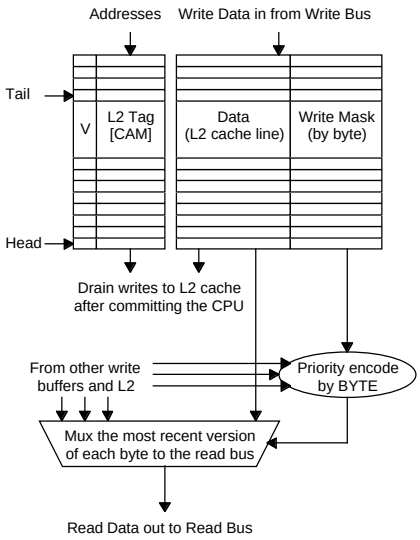
- Buffer speculative state
- Detect when true data-dependencies are violated
  - **Discard speculative state**
  - **Re-execute loop iteration**
- Safely commit speculative state

Speculation Hardware



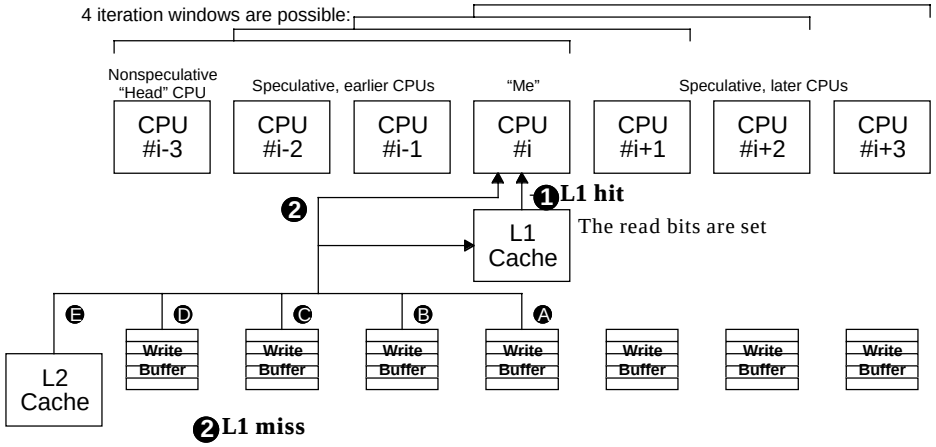
- **Speculative state in L1 cache**
  - **read**
  - **modified**
  - **pre-invalidate**
  - **gang clear**
- **Speculative buffers for writes**
- **L1 read misses fetch from speculative buffers and L2 cache**

Speculative Write Buffers



- **Not on L1 cache critical path**
- **Works with L2 reads and writes**
- **Double buffering**
  - **"Old" buffer drains to L2 cache**
  - **"New" buffer used for next speculative iteration**

Speculative Reads

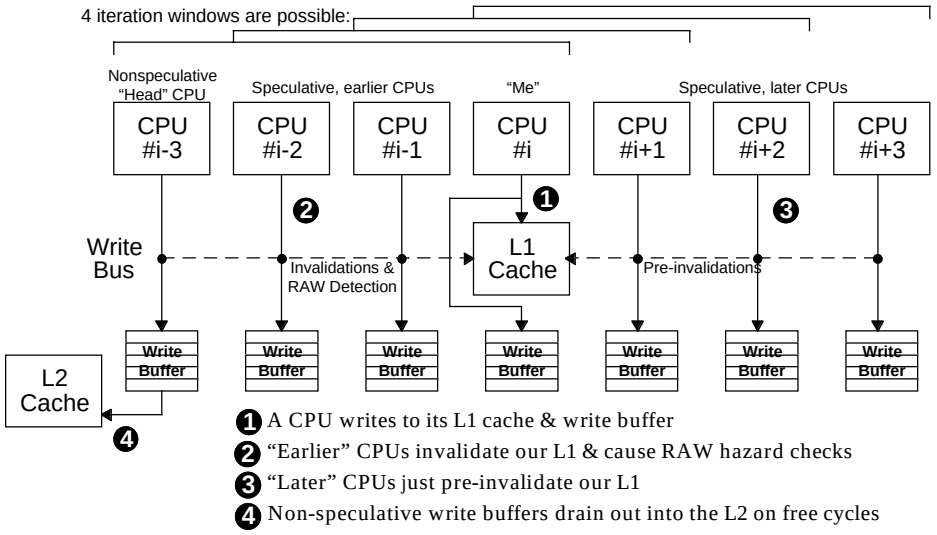


❷**L1 miss**

L2 and write buffers are checked in parallel

The newest bytes written to a line are pulled in by priority encoders on each byte (priority A-E) and modified bits set

Speculative Writes



Speculation and the OS

- OS does not run in speculation mode
- Need to manage user/kernel transitions
  - **Syscalls**
  - **Exceptions (e.g. page faults, divide by 0)**
  - **Interrupts**
- Only in-order CPU allowed to take synchronous exceptions
  - **Syscall: commit without advance of iteration number**
  - **Exception: hold exception untill in-order CPU**
- Deadlock prevention
  - **Must take all interrupts (e.g. disk I/O)**

Preliminary System Performance

- **Four processors**
- **All OS effects included using SimOS**

Application	Speedup	Restart (%)	Write State
wc	3.6	9.4	50 bytes
eqntott	2.7	40.7	25 bytes
grep	2.2	53.7	30 bytes
diff	1.2	85.5	30 bytes

- **Significant speedup**
- **Minimal write state**

Speculative Hardware Overheads

- Each CPU
  - **Control registers to manage speculation (~10 registers)**
  - **4 tag bits per store buffer entry**
  - **3 tag bits per cache line with gang clear**
  - **Mirror of L2 buffer tags**
- Secondary cache
  - **4–8 L2 buffers (256 B – 1KB each)**
  - **8 controls lines between CPU and L2 buffers (add to write-bus)**
  - **L2 buffer controller**
- Not that much extra hardware!

Hydra Conclusions and Status

- A new way to design microprocessors
  - **Single-chip MP exploits parallelism at all levels**
  - **Low overhead support for speculative parallelism**
  - **Provides up to 2× performance on applications with higher levels of parallelism**
  - **Promising performance on difficult to parallelize applications**
- Status
  - **Detailed C-level model complete**
  - **Refining speculative HW and SW**
  - **Working on Verilog model**
  - **Goal : complete Verilog model of Hydra design with compiler sytem for speculation**

Hydra Team

- Faculty
  - **Kunle Olukotun**
  - **Monica Lam**
- Students - Hardware
  - **Basem Nayfeh, Lance Hammond, Mike Chen, Maciek Kozyrczak**
- Students - Software
  - **Jeff Oplinger, Shih-Wei Liao, David Heine**