Co-Scheduling in CMP
(Chip Multicore Processors)
Slides Credits

• David Tam (University of Toronto)

• Figures clipped from this week’s papers
  
  • *Thread Clustering: Sharing-Aware Scheduling on SMP-CMP-SMT Multiprocessors*, David Tam, Reza Azimi, Michael Stumm, European Conf. in Computer Systems (EuroSys), Mar 2007
  
  
  • *Analysis and Approximation of Optimal Co-Scheduling on Chip Multiprocessors*, Yunlian Jiang, Chen Jie, Xipeng Shen, and Rahul Tripathi, PACT 2008
Multicore Scheduling

*What we are covering*

- Contention-aware scheduling – how to keep competing applications apart
- **Co-scheduling** – how to group complimentary applications together
- Next week: Work-stealing – load balancing
- Student Presentations today
  - Quantum damping – adjust CPU quantum (Kofi)
  - Hyperthread-aware Scheduling (Shahab)
Co-Scheduling on CMP

• Scheduling compatible threads together

• Taxonomy
  – Reactive at runtime v. pro-active (predictive) before execution
  – HW driven v. software-based
  – Fine-grained v. coarse-grained
  – Wide range of techniques, metrics, algorithms
“Symbiotic Job Scheduling” is not a new idea

- Seminal paper by Allan Snavely and Dean Tullsen at UCSD proposed this idea for SMT architectures in 2000 before the multicore craze.

- SOS (Sample, Optimize, Symbios)
  - Uses runtime sampling of thread behavior via hardware counters
  - Applies a heuristic to guess at optimal schedule
  - Adjusts the sets of co-scheduled jobs to match the ‘optimal’ schedule
SOS Scheduling
(Snavely and Tullsen, cont.)

• Co-scheduling threads increases ILP since more instructions are available for the pipeline.

• Performance metric:
  – \( WS(t) = \text{Weighted Speedup in interval } t = \sum_{i}^{n} (\text{realized IPC } job_i / \text{single-threaded IPC } job_i) \)
  – Interval \( t \) starts and ends on a certain cycle at a specific point in job execution
SOS Scheduling (briefly)

Low level predictors:
  FP queue conflicts
  FP unit conflicts
  Dcache hit rate

Higher level predictors
  Combined HW predictors (sum a bunch)
  Instruction diversity
  Consistent IPC across timeslices
SOS Scheduling (briefly)

Improvements close to 18% over random

Issues
• ratio of time for sampling: time for symbiosis
• overhead
  - rate of resampling – want to
  - hardware monitoring

Enhancements
• hierachical SOS
• SOS with priorities

Figure 5: Response time improvements obtained by SOS over a random jobscheduler for various levels of multithreading.
Compatible Phase Co-scheduling
El-Moursy, Garg, Albonesi, and Dwarkadas
University of Rochester and Cornell, 2006

• **Goal:** Scheduling of thread phases based on “compatibility” to improve overall throughput

• Compatibility w.r.t. sharing of processor and cache resources

• **Two Phase Method:**
  – Runtime monitoring of low level hardware contention;
  – Periodic thread migration to form compatible groups

• **Assumptions:**
  – independent threads (no a-priori knowledge of thread behavior);
  – dual-core CMP, dual-threaded SMT (4 contexts)
Compatible Phase Co-scheduling

Which hardware resources to monitor?

- Register File
- L1 D-cache
- Functional Units
- L1 I-cache

Use HW performance counters
Compatible Phase Co-scheduling

Simulation using Simplescalar 3.0 Simulator
  – Simplescalar is a 1997, classic ILP simulator
  – Would someone like to report on MIT’s Graphite Multicore Simulator?

• Benchmarks from SPEC2000 (SPEC2006 not out yet)
• SMT similar to Pentium 4 Hyperthreading
• Alpha 21264 architecture (they mimic multicore in the simulator)
• 10 job mixes of 4 benchmarks each
• Metric: geometric mean of IPC
  $\sqrt[4]{\prod\frac{IPC_{new}}{IPC_{old}}}$. 
Compatible Phase Co-scheduling

Co-Scheduling Strategies based on inter-thread contention for HW

• Data Cache Conflict Scheduling (DCCS)
• Register File Utilization Scheduling (RFUS)
• Register File Conflict Scheduling (RFCS)

Co-Scheduling Strategies based on sensitivity of thread’s need for HW resources

• IPC-based Scheduling (IPCS)
• Ready Inflight Ratio Scheduling (RIRS)
Compatible Phase Co-Scheduling

Data Cache Conflict Scheduling (DCCS)

Measuring D-cache conflicts with activity vectors and paired conflict vectors
Compatible Phase Co-Scheduling
Data Cache Conflict Scheduling (DCCS)

- Count hits to each set during an interval
- At end of interval, capture each thread’s cache activity
  - Copy AND of MSB and overflow bit for each set in the cache

Per thread
Per cache set
hardware counters
Compatible Phase Co-Scheduling

Combining activity vectors to find best pairings

- For each pair of threads, AND the activity vectors
- Compute conflict score as sum of 1 bits
- Create 2x2 conflict cores
- Choose most compatible 2x2 mix for the next scheduling interval

![Diagram showing activity and conflict vectors](image)
Compatible Phase Co-Scheduling

In-Class Exercise (handout): Using El-Moursy’s activity vectors for compatible co-scheduling

(1) Find the best pairings

(2) Can you find a way to scale this to m cores with n-way SMT on each core. Assume and n are even numbers.
Compatible Phase Co-Scheduling

Best scheme was RIRS (white, rightmost)  
• Consistent over mixes  
• Best case: 7% over best static mix

• DCCS (3rd bar)  
Performance very sensitive to mix  
• Best case: 9% over best static mix

Both did best on mix10

Figure 12. Performance of the different scheduling policies relative to the best static thread grouping.
Compatible Phase Co-Scheduling

Fine grain (100K cycles) vs. Coarse grain intervals (100M cycles)

Very fine grain co-scheduling cycle has high overhead and may interfere with OS activities

Overall similar performance
Thread Clustering: Sharing-Aware Scheduling on SMP-CMP-SMT Multiprocessors

David Tam, Reza Azimi, Michael Stumm
University of Toronto, Eurosys 2007

Goals:

• Co-locate threads that share data heavily on the same chip: decrease cache latencies
• Place non-computing threads with high memory footprints on different chips: reduce cache capacity problems

Approach: monitor cache sharing patterns, then cluster through migration; runtime

Assumptions:

• Applications are commercial server apps
• Implemented in Linux kernel on 8-way IBM Power5
Thread Clustering: Sharing-Aware Scheduling on SMP-CMP-SMT

Utilizes HW performance monitoring units that contain hardware performance counters (HPCs)

HPC

- counts cache misses
- interrupts the CPU upon exceeding threshold
- gives information about addresses that cause the cache miss

HPCs are limited in number → they use logical HPCs through an HPC multiplexing technique.

HPC cache miss address hard to extract → indirect method to get the miss address
Other approaches to co-scheduling

SMP application thread analysis
Intel Hyperthreading specific
Cache contention or cache sharing focus
Processor HW focus

We will compile a high level taxonomy as an in-class exercise next week.