An Integrated Hardware/Software Approach to On-Line Power-Performance Optimization

Sandhya Dwarkadas

University of Rochester

Collaborators at UR: David Albonesi, Chen Ding, Eby Friedman, Michael L. Scott, UR Systems and Architecture groups
Collaborators at IBM: Pradip Bose, Alper Buyuktosunoglu, Calin Cascaval, Evelyn Duesterwald, Hubertus Franke, Zhigang Hu, Bonnie Ray,Viji Srinivasan

Outline

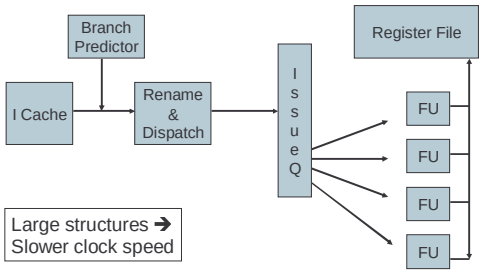
- Framework: Dynamically Tunable Clustered Multithreaded Architecture
- Motivation: Workload characterization
- Architectural support for adaptation
- Role of program analysis
- Resource-aware operating system support

Emerging Trends

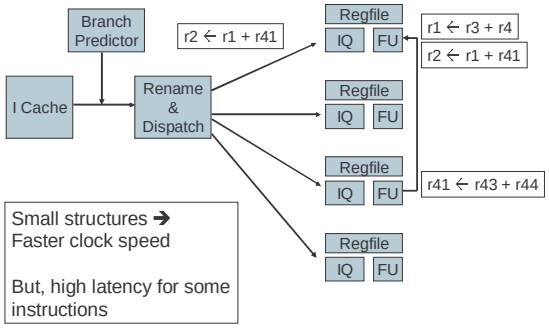
- Wire delays and faster clocks will necessitate aggressive use of clustering
- Larger transistor budgets and low cluster design costs will enable addition of more clusters incrementally
- There is a trend toward multithreading to exploit the transistor budget for improved throughput by combining ILP and TLP

→ Combine clustering and multithreading?

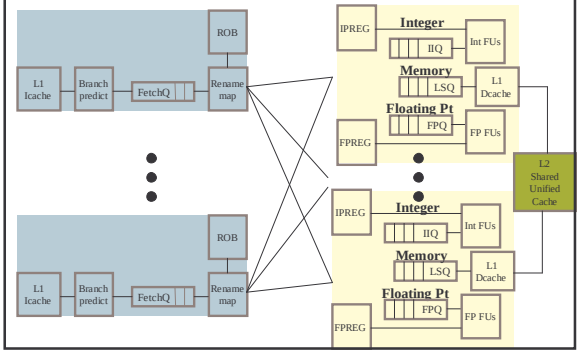
Conventional Processor Design



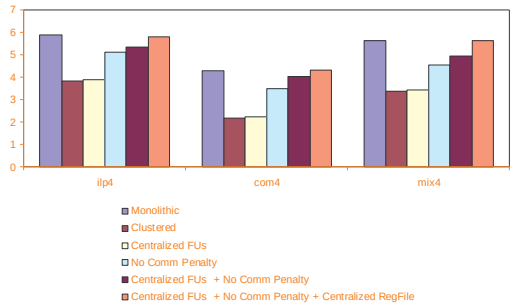
A Clustered Processor



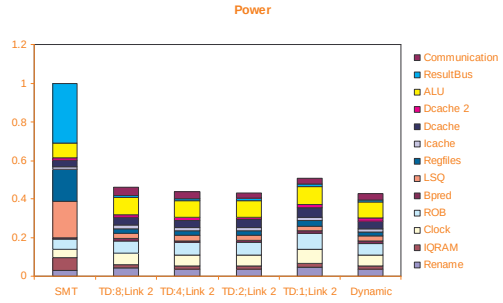
A Clustered Multithreaded (CMT) Architecture



Components of IPC Degradation



Overall Energy Impact



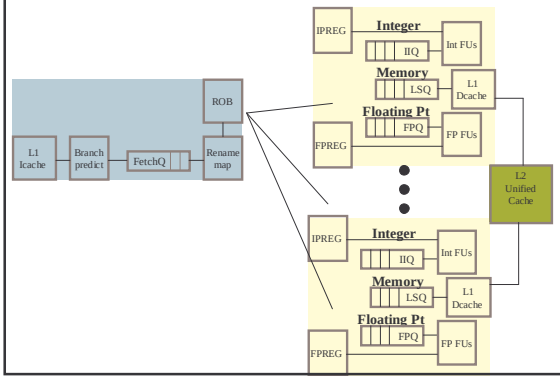
Problems

- Tradeoff in communication vs. parallelism for a single thread
- Increased communication delays and contention when employing multiple threads
  - Reduced performance
  - Increased energy consumption

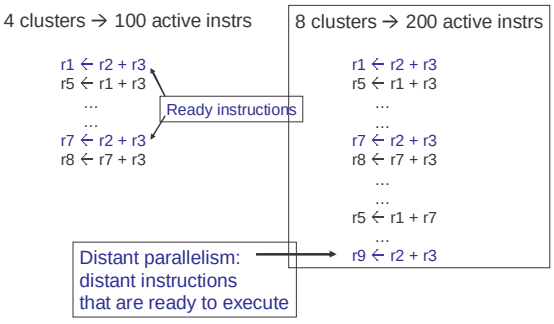
Goal:

Intelligent mapping of applications to resources for improved throughput and resource utilization as well as reduced energy

Single Thread Execution



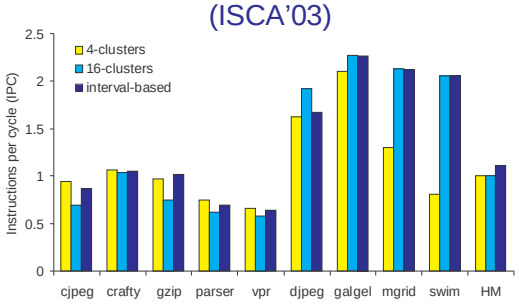
Communication vs Parallelism



Single-Thread Adaptation [ISCA'03]

- Dynamic interval-based exploration can adapt to available instruction-level parallelism in a single thread
  - Determine when communication can no longer be tolerated in exploiting additional clusters
- Allow remaining clusters to be turned off to reduce power consumption or to be used by a different thread/application

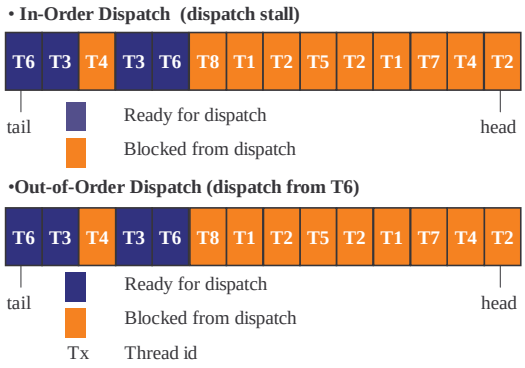
Results with Interval-Based Scheme (ISCA'03)



An Integrated Approach to Dynamic Tuning of the CMT

- Architectural design and dynamic configuration for fine-grain adaptation
- Program analysis to determine application behavior
- Runtime support to match predicted application behavior and resource requirements with available resources
  - Resource-aware thread scheduling for maximum throughput and fairness
  - Runtime support for balancing ILP with TLP in parallel application environments

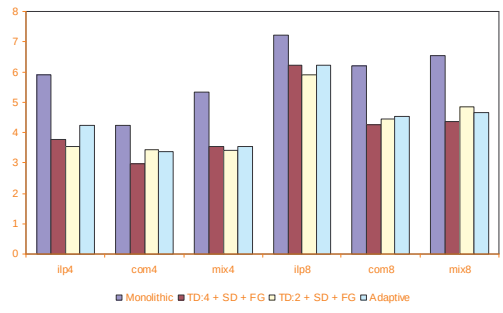
Out-of-order Dispatch & Fetch Gating



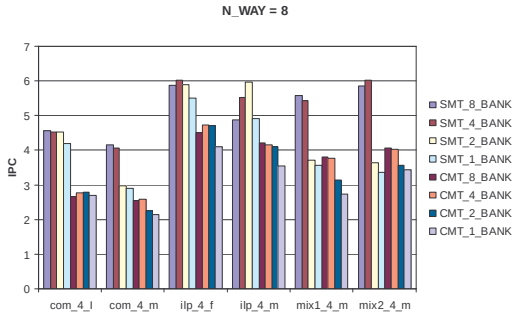
Multithreaded Adaptation

- Basic scheme
  - Interval-based
  - Fixed 100,000 cycles
  - Exploration-based
  - Hysteresis to avoid spurious changes

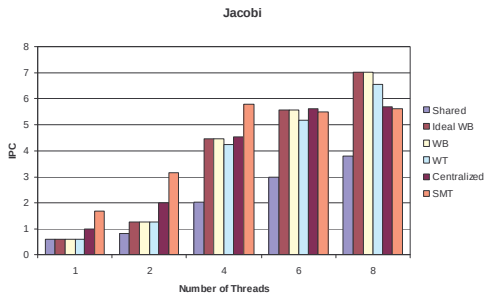
Thread to Cluster Assignment



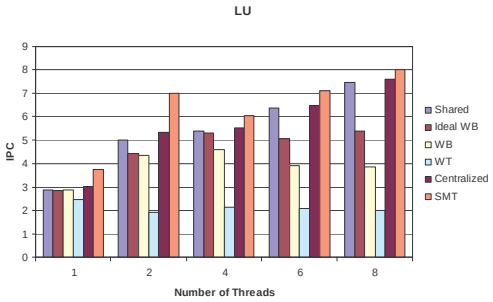
Thread to Cache Bank Assignment



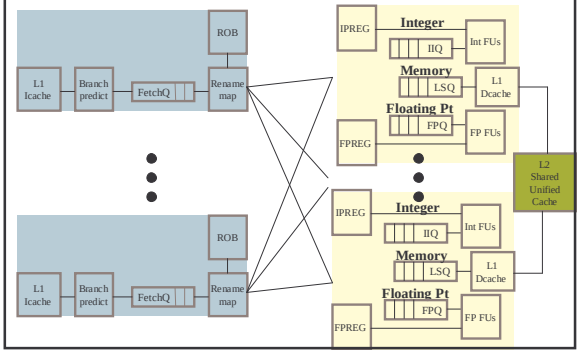
ILP vs. TLP



ILP vs. TLP

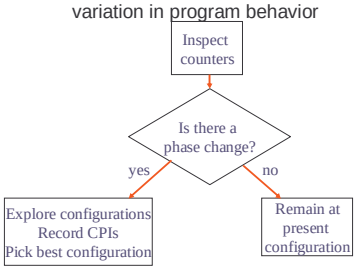


A Dynamically Tunable Clustered Multithreaded (DT-CMT) Architecture



Current Approaches to Adaptation

Reactive → Adaptive change is triggered after observed variation in program behavior



→ Success depends on ability to repeat behavior across successive intervals

Interval Length

Problem:

- Unstable behavior across intervals

Solution:

- Start with minimum allowed interval length
- If phase changes are too frequent, double the interval length – find a coarse enough granularity such that behavior is consistent
- Periodically reset interval length to the minimum
- Small interval lengths can result in noisy measurements

Varied Interval Lengths

Benchmark	Instability factor for a 10K interval length	Minimum acceptable interval length and its instability factor
gzip	4%	10K / 4%
vpr	14%	320K / 5%
crafty	30%	320K / 4%
parser	12%	40M / 5%
swim	0%	10K / 0%
mgrid	0%	10K / 0%
galgel	1%	10K / 1%
cjpeg	9%	40K / 4%
djpeg	31%	1280K / 1%

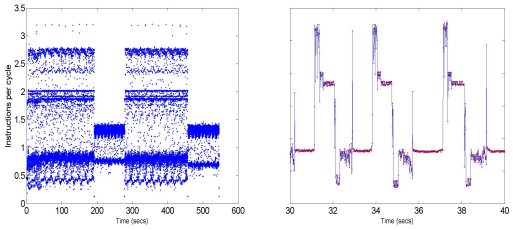
Instability factor: Percentage of intervals that flag a phase change

Characterizing Program Behavior Variability

- Whole program instrumentation (currently SPEC2k)
- Periodic hardware performance counter sampling using *Ticker*
  - Dynamic Probe Class Library (DPCL) to insert a timer-based interrupt in the program
  - Performance Monitoring API (PMAPI) to read the hardware counters
  - AIX-based
- Sampling interval of 10 msec
- Examination of IPC, L1D cache miss rate, instruction mix, branch mispredict rate
- Statistical analysis – correlation, frequency analysis, behavior variation

Example IPC Plots

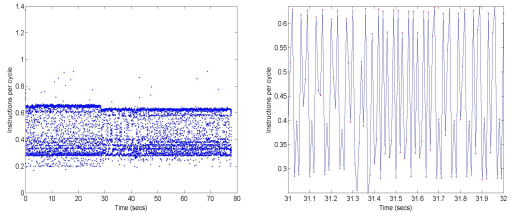
SPEC2k:bzip2



- Existence of macro phase behavior
- Significant behavior variation even at coarse granularities
- Strong frequency components/periodicity across several metrics

Example IPC Plots

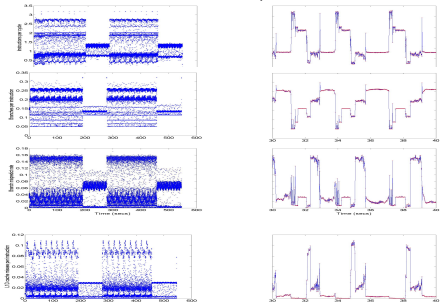
Spec2k:art



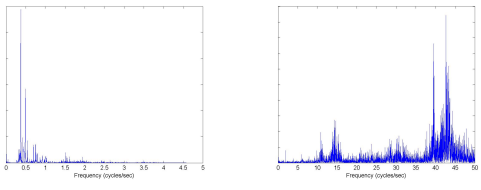
- High rate of behavior variation from one measurement to the next

Similarity Across Metrics

SPEC2k:bzip2



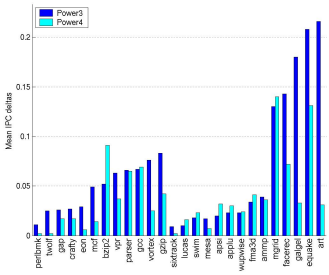
Comparing Frequency Spectra



bzip2 art

- Strong low (bzip2) and high (art) frequency components, indicating high rate of repeatability

Program Behavior Variability



- Variation in behavior, while different, persists across different sampling interval sizes

Important Behavior Characteristics

- Programs exhibit high degrees of repeatability across all metrics
- Rate of behavior repeatability (periodicity) across metrics is highly similar
- Variation in behavior from one interval to the next can be high
- Variation in behavior, while different, persists across different sampling interval sizes

➡ *On-line power-performance optimization needs to be predictive rather than reactive*

On-Line Program Behavior Prediction

- Linear (statistical) predictors to exploit behavior in the immediate past
  - Last value
  - Average(*N*)
  - Mode(*N*)
- Table-based predictors to exploit periodicity (non-linear)
  - Run-length encoded
  - Fixed-size history
- *Cross-metric* predictors to exploit similarity across metrics
  - Use one metric to predict several potentially different metrics
  - Efficiently combine multiple predictors

Table-Based Predictors

- E.g. table-based and asymmetric predictor –

  $a_{t-4}a_{t-3}a_{t-2}a_{t-1} \rightarrow a_t, b_t \quad a_{vote}, b_{vote}$

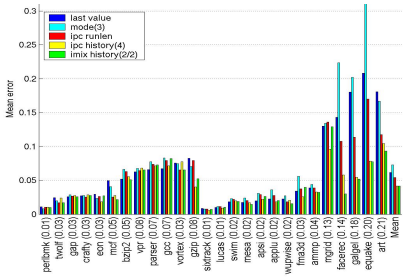
- Default to last value during learning period
- Use a voting mechanism to update table entries
  - Prediction (a_t or b_t) is updated with the mode of the actual value (*vote*) the last time this history was encountered, the current prediction(*t*), and the measured value at the end of the interval
- Encoding and length of history (index) can be varied
  - Fixed size or run-length encoded

➡ *Trade-off between noise tolerance, learning period, and prediction accuracy*

Design Trade-offs

- Precision
  - Too coarse a precision implies insensitivity to fine-grained behavior
  - Too fine a precision implies sensitivity to noise
- Size of history
  - Too long a history implies a potentially long learning period
  - Too short a history implies inability to distinguish between common histories of otherwise distinct regions
- Both precision and history have table size implications

Mean IPC Prediction Error (Power3)



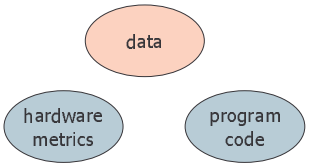
Program Behavior Predictability

- Variations in program behavior are predictable to within a few percent
- Table-based predictors outperform any others for programs with high variability
- Cross-metric table-based predictors make it possible to predict multiple metrics using a single predictor
- Microarchitecture-independent metrics allow stable prediction even when the predicted metric changes due to dynamic optimization

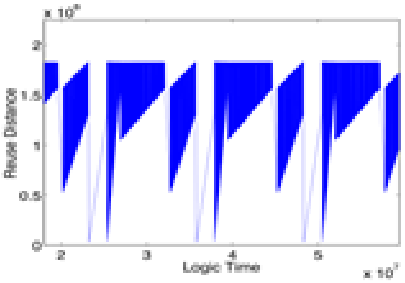
Problems

- High variability in program behavior
- Interval length hard to determine
  - Too small → measurement noise
  - Too large → missed opportunities for adaptation
- Interval and actual phase boundaries do not match

Information Space for Workload Analysis

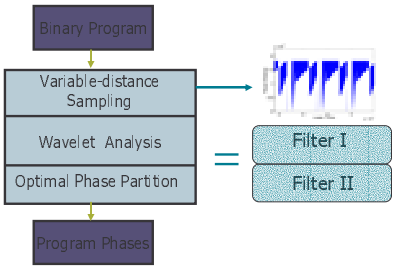


Data Locality Analysis (Shen and Ding)



Reuse Distance Signature Of TOMCATV

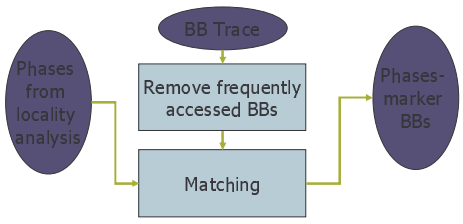
Program Phase Detection



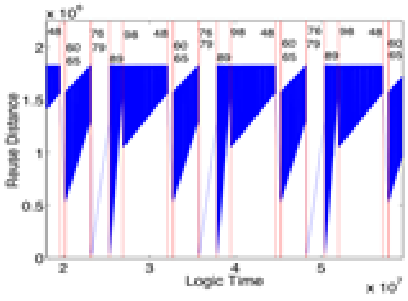
Phase Marker Insertion

-----Basic Block Trace Analysis

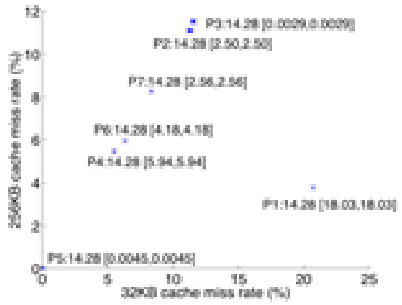
- Objective: to find basic blocks marking unique phase boundaries.



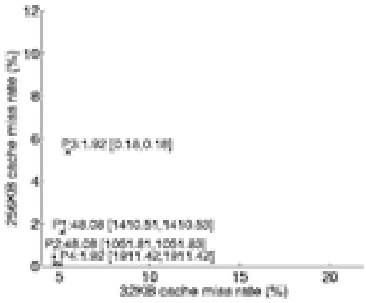
TOMCATV RD Signature with Phase Boundaries



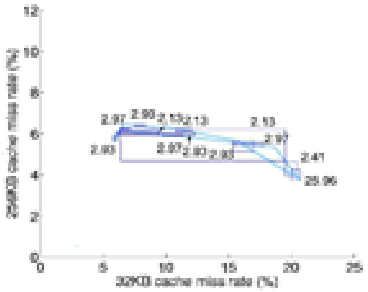
Similarity of Locality Phases: TOMCATV (5250 Phases)



Similarity of Locality Phases: COMPRESS (52 Phases)



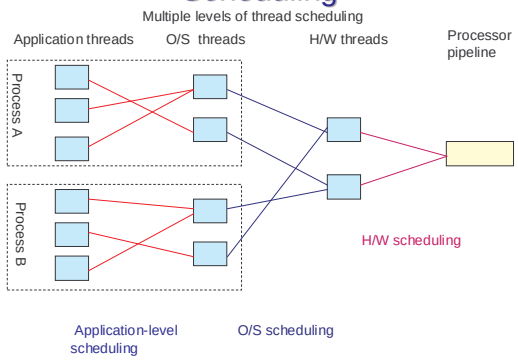
Similarity of Phases with BBV TOMCATV (2493 Intervals)



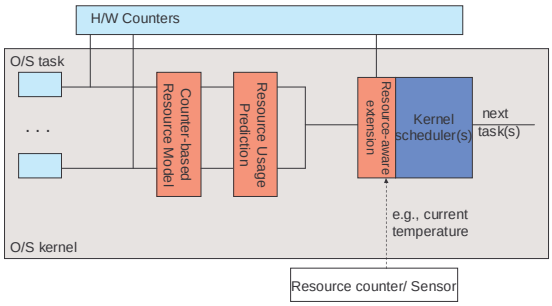
Bringing It All Together

- Locality analysis for phase detection and marking of macro phases
- Linear or non-linear (table-based) prediction within each phase for improved learning

Resource-Aware Thread Scheduling



Resource-Aware O/S Scheduler



Fair Cooperative Scheduling [PPoPP 2001]

- Each process is allocated a piggy-bank of time (set to 1 quantum) from which it can borrow and to which it can add
- The piggy-bank is used to boost a process's priority (with the original purpose of responding to a communication request when notified by a wakeup signal)
- A process can add to the piggy-bank whenever it relinquishes the processor

→ Adapt the above so the piggy-bank is used to schedule a process earlier than according to priority and replenished, for example, when reconfiguring at a phase marker

→ Coordinate among schedulers for multiple hardware contexts

Application-Level Scheduling

- Provide a framework for
  - trading ILP for TLP based on application characteristics and available resources
  - Specifying cache and cluster sharing configurations at appropriate points

At the JVM level

- Target server workloads

At the level of an API such as OpenMP

- Target scientific/parallel applications

Resource-Aware Thread Scheduling: Other Applications

Power/Thermal management

- Temperature-aware process/thread scheduling to avoiding temperature hotspots
  - characterize threads based on expected temperature contribution
  - schedule based on a thread's predicted heat contribution and current temperature

Performance

- Improving L2 bandwidth utilization on a multiprocessor (e.g., the two cores of a Power4)
  - Characterize threads based on expected L2 cache accesses
  - avoid scheduling different threads with high L2 access concurrently

Resource-Aware Thread Scheduling (cont'd)

Performance and Power

- Resource (memory, FU, and temperature) aware thread scheduling for simultaneous multithreaded processors (e.g., the Power5, and the hyper-threads of the Pentium IV) or our proposed clustered multithreaded architecture

Summary: An Integrated Hardware/Software Approach to DT-CMT

- Aggressive clustering and multithreading requires a whole-system integrated view in order to maximize resource efficiency
  - Architectural configuration support (while carefully considering circuit-level issues)
  - Program analysis
  - Runtime/OS support

On-Going Projects at UofR

- CAP: Dynamic reconfigurable general-purpose processor design
- MCD: Multiple Clock Domain Processors
- DT-CMT: Dynamically Tunable Clustered Multi-threaded architectures
- InterWeave: 3-level versioned shared state (predecessors: InterAct and Cashmere)
- ARCH: Architecture, Runtime, and Compiler Integration for High-Performance Computing

See http://www.cs.rochester.edu/research and http://www.cs.rochester.edu/~sandhya