Using Variable-MHz Microprocessors to Efficiently Handle Uncertainty in Real-Time Systems

Eric Rotenberg

Center for Embedded Systems Research (CESR)
Department of Electrical & Computer Engineering
North Carolina State University
www.tinker.ncsu.edu/ericro
Real-Time Embedded Processor Trends

• Need more performance for real-time tasks
  – More instructions per task
  – Tighter deadlines
  – More tasks

• Inherit high-performance microarchitecture techniques
  – Pipelining
  – Branch prediction
  – Caches
  – Dynamic scheduling
  – Multiple instruction issue (superscalar/VLIW)
Worst-Case Timing Analysis

• Find upper bound on number of cycles for task
  – Upper bound must be safe
    • Predicted Cycle Count > Actual Cycle Count
    • So that designer can guarantee deadline will never be missed
  – Upper bound should be accurate
    • Predicted Cycle Count ~ Actual Cycle Count
    • So that perceived frequency requirement is close to actual frequency requirement

\[
\text{frequency} \geq \frac{\text{cycle count}}{\text{deadline}}
\]
Problem: Uncertainty

• Worst-case timing analysis of complex pipelines
  – Ambiguous addresses ➞ ambiguous cache state
    • Assume certain loads always miss
  – Ambiguous control flow ➞ ambiguous predictor state
    • Assume certain branches always mispredict
  – Etc.

• Worst-case timing analysis underestimates microarchitecture performance to be safe
Symptom: Redundant Performance

• Designer must turn to *clock frequency* as a reliable source of performance

• Redundant performance
  – High-performance microarchitecture
    – Efficient source of performance
    – Unreliable (unpredictable performance)
  – High clock frequency
    – Inefficient source of performance
    – Reliable (predictable performance)

*We want these…*

*…but get these.*
Fault Tolerance Angle

• Redundancy methods
  – Spare always active
  – Spare swapped in

• Efficient performance redundancy
  – What is a “fault”?
    • Transient microarchitecture performance fault
  – What is the “spare”?  
    • Frequency reserves
  – What is the “sparing method”? 
    • Swap in
Efficiently Handling Uncertainty

- **Simulated-worst-case (SWC)**
  - Get “typical” worst-case timing via detailed microarchitecture simulation
  - Accurate but unsafe
  - The basis for a low speculative frequency

- **Worst-case (WC)**
  - State-of-the-art static worst-case timing analysis
  - Less accurate but safe
  - The basis for a high recovery frequency (“spare”)

Rotenberg © 2001
recovery frequency
(based on WC)

speculative frequency
(based on SWC)

frequency requirement

time (ms)

frequency (MHz)

transient performance fault
Transient Fault Detection/Recovery

• Straightforward detection method
  – Miss deadline
  – Cannot recover

• Conservative detection/recovery method
  – Divide tasks into sub-tasks [Mosse et al.]
  – Set up artificial interim deadlines for sub-tasks called checkpoints
  – Fault detection
    • Sub-task misses its checkpoint at the speculative frequency
    • Microarchitecture performed worse than simulation, somewhere in between SWC and WC
  – Fault recovery
    • Run all remaining sub-tasks at recovery frequency
Potential Benefits

• Power
  – Favoring microarchitectural sources of performance is better in terms of power

• Relax need for sophisticated worst-case timing analysis
  – Reliability: Simple analysis is less bug-prone than complex analysis (need reliability for the recovery frequency)
  – Increasing programmer productivity and software complexity: Re-introduce previously discouraged programming practices
Target Microprocessors

• Microprocessors with many frequency/voltage settings
  – E.g., Transmeta, Intel, AMD

• Custom-fit processors
  – Synthesize hardware specific for an embedded application
    (less flexible but highly optimized)
  – Examples:
    • Single pipeline, two frequency/voltage settings
    • Dual pipelines, each with single frequency/voltage setting
    • Novel microarchitectural support for variable frequency
Statically Deriving Frequencies

- Static worst-case timing analysis produces:
  - $T_{i,WC,f}$
  - Worst-case execution times (ms) for all sub-tasks $i$ at all supported frequencies $f$

- Microarchitecture simulation produces:
  - $T_{i,SWC,f}$
  - Simulated-worst-case execution times (ms) for all sub-tasks $i$ at all supported frequencies $f$
Statically Deriving Frequencies (cont.)

\[
\sum_{j=1}^{i-1} T_{j,SWC,f_{spec}} + T_{i,WC,f_{spec}} + \text{overhead} + \sum_{k=i+1}^{s} T_{k,WC,f_{rec}} \leq \text{deadline}
\]

- There is one equation for each sub-task \(i\)
- Solving method
  - Start with lowest \(f_{spec}\)
  - For each sub-task \(i\), find minimum \(f_{rec}\) that satisfies its eqn.
  - If a sub-task is reached where no \(f_{rec}\) can be found, start over with next higher \(f_{spec}\)
  - Output: minimized speculative and recovery frequencies
Frequencies for Comparison

• Frequency recommended by worst-case timing analysis
  \[ f_{wc} \sum_{i=1}^{s} T_{i,WC,f_{wc}} \leq \text{deadline} \]

• “Optimal” speculative frequency
  – What if we ideally know ahead of time that there won’t be a fault?
  \[ f_{opt} \sum_{i=1}^{s} T_{i,SWC,f_{opt}} \leq \text{deadline} \]
Experiments

• Processor
  – 7-stage pipeline
  – Single-issue with out-of-order execution
  – 16-entry ROB
  – 2K-entry bimodal predictor
  – 8KB direct-mapped instruction and data caches
  – 50 MHz – 300 MHz in 25 MHz increments
  – Memory access time (in nanoseconds) is constant

• Task = 16 FFT sub-tasks

• Static worst-case timing analysis
  – Currently, don’t have access to static timing analyzer
  – Mimic WC analysis
  – Over-estimate timing by injecting extra cache misses during simulation
    • WC10: 10% extra
    • WC30: 30% extra
    • WC50: 50% extra
Results (WC30)

- $f_{\text{rec}}$
- $f_{\text{wc}}$
- $f_{\text{spec}}$
- opt

Deadline (ms) vs. Frequency (MHz)
Results (WC50)

- f_rec
- f_wc
- f_spec
- opt
Trend #1

- More benefit with poorer timing analysis
  - E.g., 40 ms deadline
    - WC10: 25 MHz delta between speculative and worst-case freq.
    - WC50: 100 MHz delta between speculative and worst-case freq.
  - Reason
    - Speculative frequency depends on actual behavior (constant)
    - Worst-case frequency depends on quality of timing analysis
Trend #2

• More benefit with tighter deadlines
  – Tighter deadline requires more performance
  – Frequency gives diminishing performance returns due to irreducible main memory component
  – Need to increase frequency non-linearly to compensate for diminishing returns
  – *Effect is worse for WC than SWC due to larger memory latency component*
  – Worst-case frequency increases faster than speculative frequency
Trend #3

- Positive frequency trends
  - Speculative frequency
    - Insensitive to worst-case pessimism – no change among WC10, WC30, WC50
    - Closely tracks optimal speculative frequency
  - Recovery frequency
    - Sensitive to worst-case pessimism
    - But closely tracks the frequency produced by traditional worst-case design: “graceful degradation”
    - Effectively no downside to speculating
Summary

• Performance redundancy
  – High-perf. microarchitecture: efficient / unreliable
  – High Frequency: inefficient / reliable
  – Use frequency reserves (“swap-in-spare” approach): efficient / reliable

• Complementary timing approach
  – SWC ➔ speculative frequency (efficient / unreliable)
  – WC ➔ recovery frequency (inefficient / reliable)

• Significant frequency reduction, and:
  – Benefit increases with poorer timing analysis
  – Benefit increases with tighter deadlines
  – Speculative frequency nearly optimal, recovery frequency demonstrates graceful degradation