NC STATE UNIVERSITY

Virtual Simple Architecture (VISA): Exceeding the Complexity Limit in Safe Real-Time Systems

Aravindh Anantaraman*, Kiran Seth*, Kaustubh Patil†,
Eric Rotenberg*, Frank Mueller†

Center for Embedded Systems Research (CESR)
* Electrical & Computer Eng. / † Computer Science
North Carolina State University
www.tinker.ncsu.edu/ericro
ericro@ece.ncsu.edu

Safe Planning in Real-Time Systems

- Need worst-case execution time (WCET) of tasks
  - Guarantee hard deadlines
  - Schedulability analysis (will all tasks fit in schedule?)
- Static worst-case timing analysis
  - Derive WCETs of tasks independent of input data and considering cycle-accurate microarchitecture timing

Complexity Limit

- Capabilities of static timing analysis
  - In-order scalar pipeline with static branch prediction and split I/D caches
- Contemporary processors
  - Out-of-order, multiple issue, dynamic branch prediction, caches, deep speculation, etc.
- Analyzability fundamental to design of safe systems
  - Need for analyzability excludes contemporary microarchitecture techniques from many embedded systems
  - Long-term implications for extending scope of embedded systems

VISA Framework

- Virtual Simple Architecture (VISA)
  - Pipeline timing specification of hypothetical simple processor
  - Within capabilities of static worst-case timing analysis
- WCET derived assuming the VISA
- Speculatively attempt task on complex processor
  - Continuously gauge progress to confirm complex processor is as timely as hypothetical simple processor
  - If not as timely, reconfigure pipeline to simple mode of operation that directly implements the VISA
    - E.g., disable OOO execution, revert to static branch prediction, etc.

Gauging Progress

- Divide task into multiple sub-tasks
  - Each sub-task assigned soft deadline (checkpoint)
  - Checkpoint based on latest allowable completion time of sub-task on hypothetical simple processor
- Monitor checkpoints
  - Continued safe progress confirmed for as long as checkpoints are met
  - If a checkpoint is missed, fall back to simple mode to explicitly bound execution time

Exploiting Higher Performance

- Complex processor typically much faster
  - Task finishes very early
  - Finishing early unimportant from safety standpoint
  - Benefit comes from significant slack in schedule
- Exploit newly-created slack
  - Dynamic voltage scaling
    - Complex processor can meet checkpoints at lower frequency
  - Conventional concurrency
    - Schedule more soft- and non-real-time tasks
  - Simultaneous multithreading
    - Execute soft- and non-real-time tasks at same time as critical tasks

Key Contribution

- Eliminate need to do explicit WCET analysis of complex processor, by dynamically confirming its execution time is bounded by simple proxy
  - Enables unrestricted use of arbitrarily complex processors in safe real-time systems

Previous Alternatives

- *Bar complexity altogether*
  - VISA relaxes this restriction
- *Allow complexity, but disable at onset of hard real-time task*
  - VISA defers disabling until actually necessary
- *Continue research in aggressive WCET analysis*
  - VISA fully leverages this body of work, and benefits from continued advances (tighten WCET of proxy)

Outline

- ✓ VISA contribution
- General methodology for safe operation
- System (co-)design
  - VISA specification
  - VISA-compliant complex pipeline
  - Static worst-case timing analysis tool
- Exploiting slack for power savings
  - Frequency speculation on VISA framework
- Results
- Summary and Future Work

Setting Checkpoints

- Missed checkpoint called a misprediction
- Need enough time between checkpoint and deadline to recover from misprediction
  - Pipeline switching overhead
  - Remaining time of unfinished sub-task
    - Cannot know how much of mispredicted sub-task got done
    - Assume none of mispredicted sub-task got done
  - WCETs of remaining sub-tasks

$$\text{checkpoint}_i = \text{deadline} - \text{overhead} - \sum_{k=i}^{s} \text{WCET}_{k,f}$$

Enforcing Checkpoints

- Watchdog cycle counter
  - Initialized by code snippet at start of first sub-task
    - $checkpoint_1$*f
  - Counter value augmented by each new sub-task i
    - ($checkpoint_i$ – $checkpoint_{i-1}$)*f
    - Advances the checkpoint enforced by watchdog
  - Decremented by one each cycle
  - Missed-checkpoint exception raised if reaches 0 before next sub-task has chance to advance it

Outline

- ✓ VISA contribution
- ✓ General methodology for safe operation
- System (co-)design
  - VISA specification
  - VISA-compliant complex pipeline
  - Static worst-case timing analysis tool
- Exploiting slack for power savings
  - Frequency speculation on VISA framework
- Results
- Summary and Future Work

VISA Specification

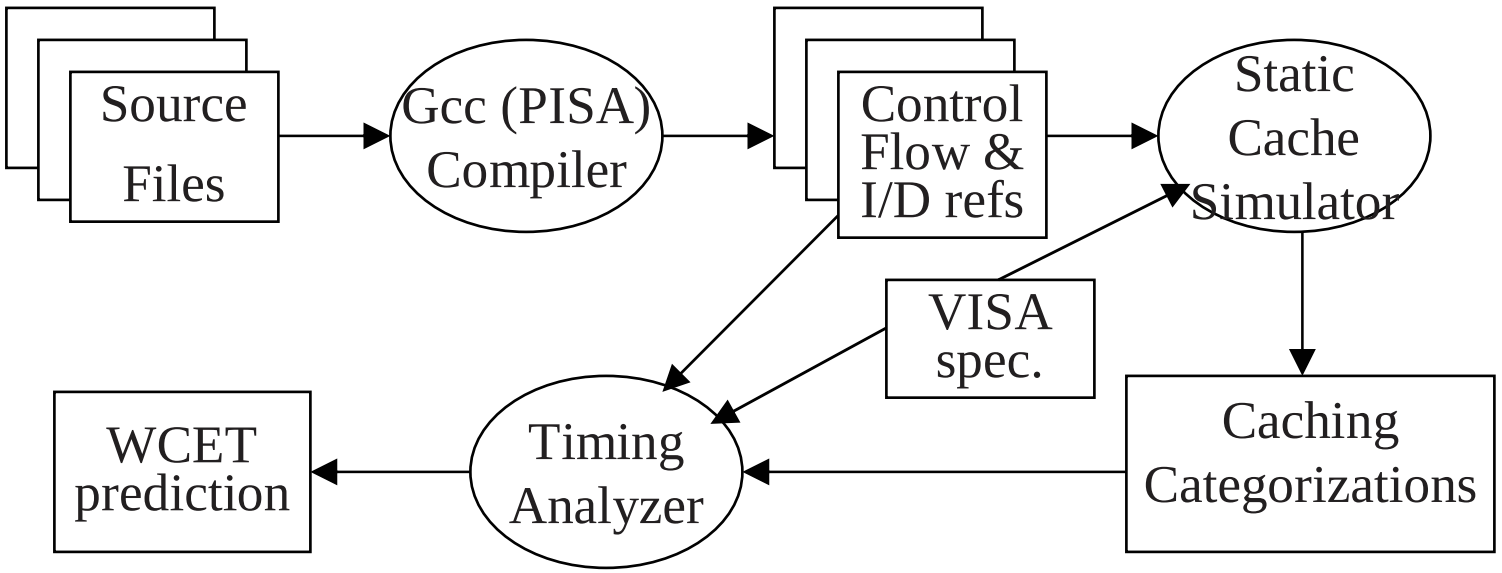
- Considerations
  - What are capabilities of current timing analysis tools?
  - How is simple mode likely to be accommodated in contemporary pipeline?
- Hypothetical simple processor
  - Six-stage, scalar, in-order pipeline
    - Fetch, decode, register read, execute, memory, writeback
    - Single unpipelined universal function unit
  - Static branch prediction: BT/FNT heuristic
  - L1 I-cache & D-cache: 64KB 4-way, 64B block, 1-cycle
  - Execution latencies: MIPS R10K latencies

VISA-Compliant Complex Pipeline

- Complex pipeline
  - Seven-stage, 4-way superscalar, out-of-order pipeline
    - Fetch, dispatch, issue, register read, execute or agen/memory, writeback, retire
    - 4 pipelined universal function units
  - Dynamic branch prediction: gshare
  - Same memory hierarchy configuration and execution latencies as VISA
- Coarse-grain reconfiguration technique
  - Override dynamic predictor with FT/BNT
  - Pipestages throttle instruction delivery to 1 instr./cycle
  - Bypass the issue queue, load/store queue
  - May optionally disable physical register file management (active list, freelist, maps, etc.) although not strictly needed

Static Worst-Case Timing Analysis Tool

- Tool derived from FSU / NCSU framework
- Ported to VISA pipeline model and SimpleScalar ISA

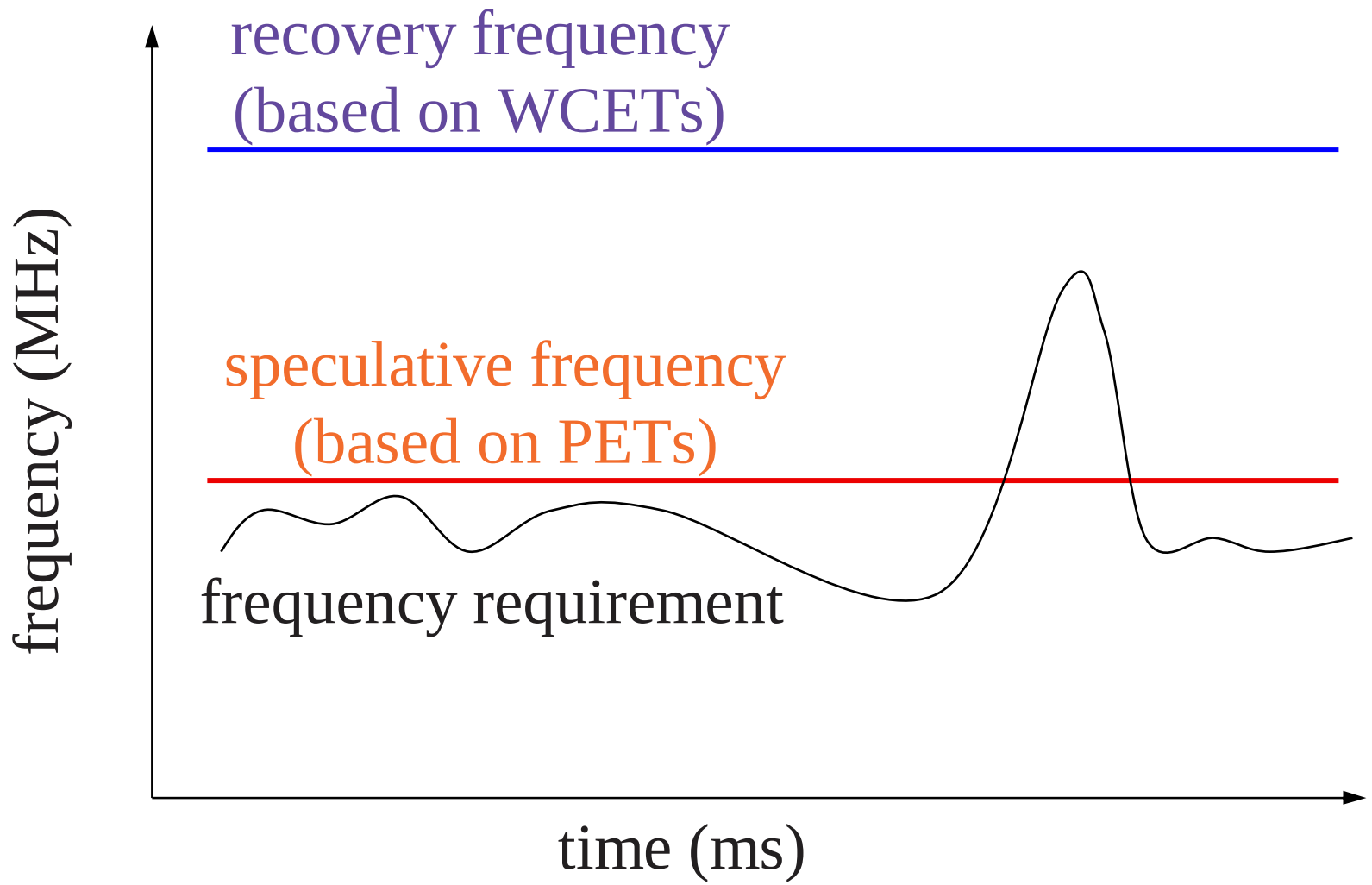


Outline

- ✓ VISA contribution
- ✓ General methodology for safe operation
- ✓ System (co-)design
  - VISA specification
  - VISA-compliant complex pipeline
  - Static worst-case timing analysis tool
- Exploiting slack for power savings
  - Frequency speculation on VISA framework
- Experiments
- Summary and Future Work

Frequency Speculation

[Rotenberg, MICRO-34]



Frequency Speculation on VISA

- Frequency speculation uses same gauging mechanism (sub-tasks & checkpoints)

$$\sum_{j=1}^{i-1} PET_{j,f_{spec}} + PET_{i,f_{spec}} + ovhd + WCET_{i,f_{rec}} + \sum_{k=i+1}^{s} WCET_{k,f_{rec}} \leq deadline$$

sub-task i misses checkpoint

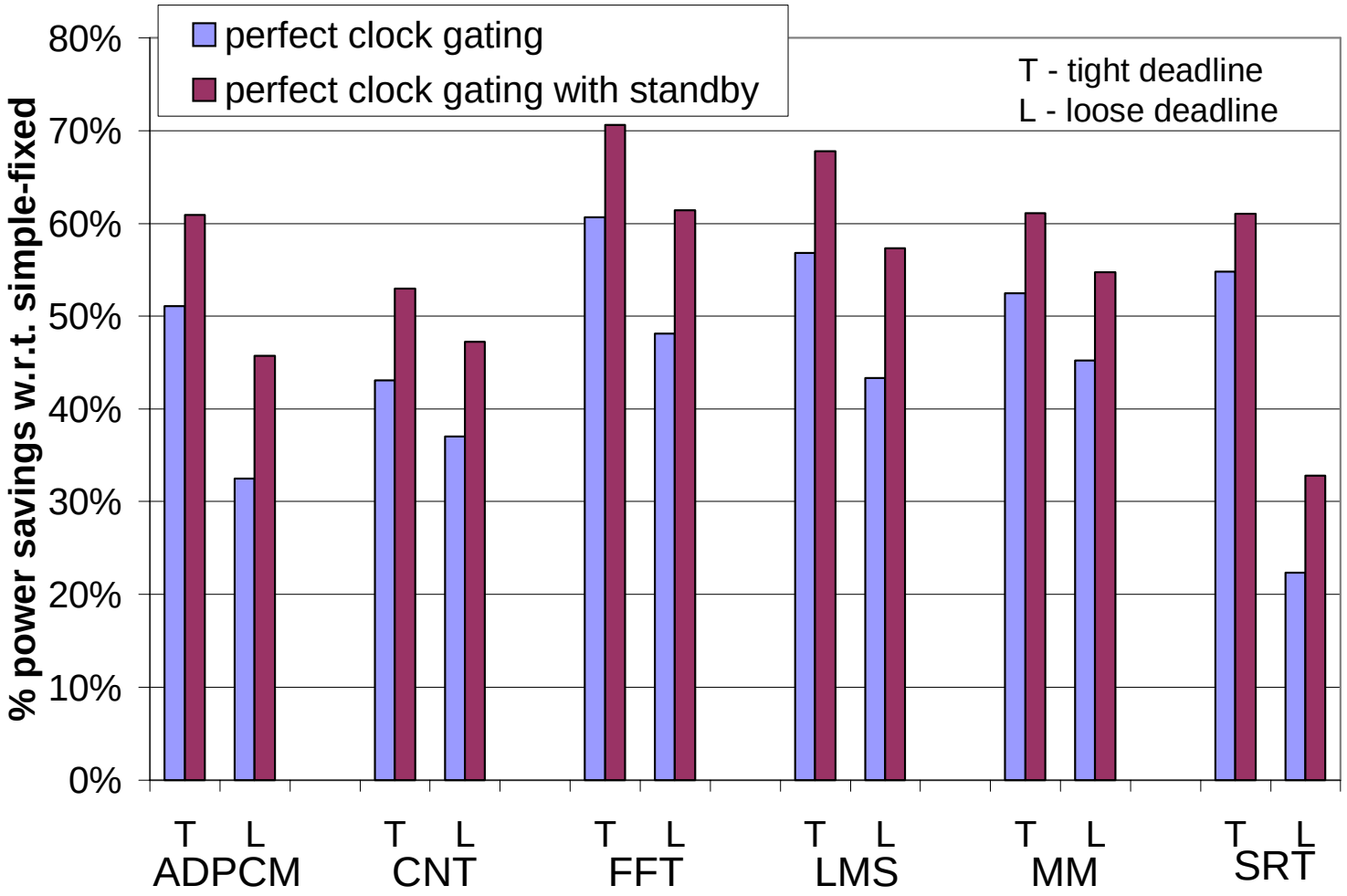
Conventional frequency speculation:
- Switch to recovery frequency

Frequency speculation on VISA:
- Switch to recovery frequency
- **Switch to simple mode**

Experimental Method

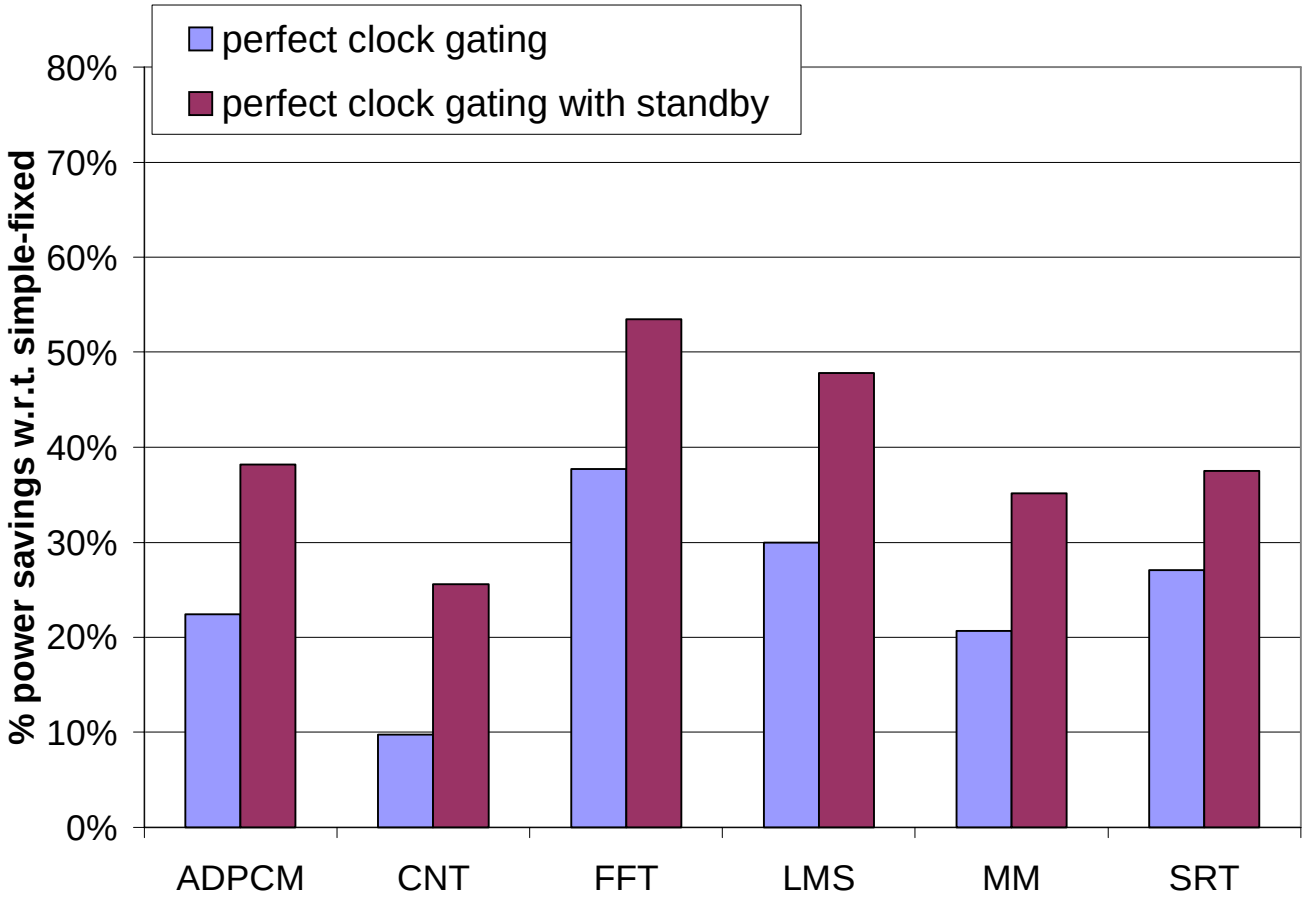
- Cycle-accurate simulator with two processor models
  - *complex*: VISA-compliant complex processor
  - *simple-fixed*: Explicitly-safe simple processor
    - Literal implementation of VISA specification
    - More power-efficient than simple mode of *complex*
- Power modeling
  - Integrated Wattch models, modified for contemporary μarch
  - Added DVS support with 37 frequency/voltage settings
  - Minimum frequency setting (100 MHz) during idle time
- 6 tasks from C-lab real-time benchmarks
  - Straightforward manual sub-task selection
  - Run single task 200 times (periodic hard-real-time task)
  - Consider both tight and loose deadlines

Results


perfect clock gating
perfect clock gating with standby
T - tight deadline
L - loose deadline
% power savings w.r.t. simple-fixed
80%
70%
60%
50%
40%
30%
20%
10%
0%
T L
ADPCM
T L
CNT
T L
FFT
T L
LMS
T L
MM
T L
SRT

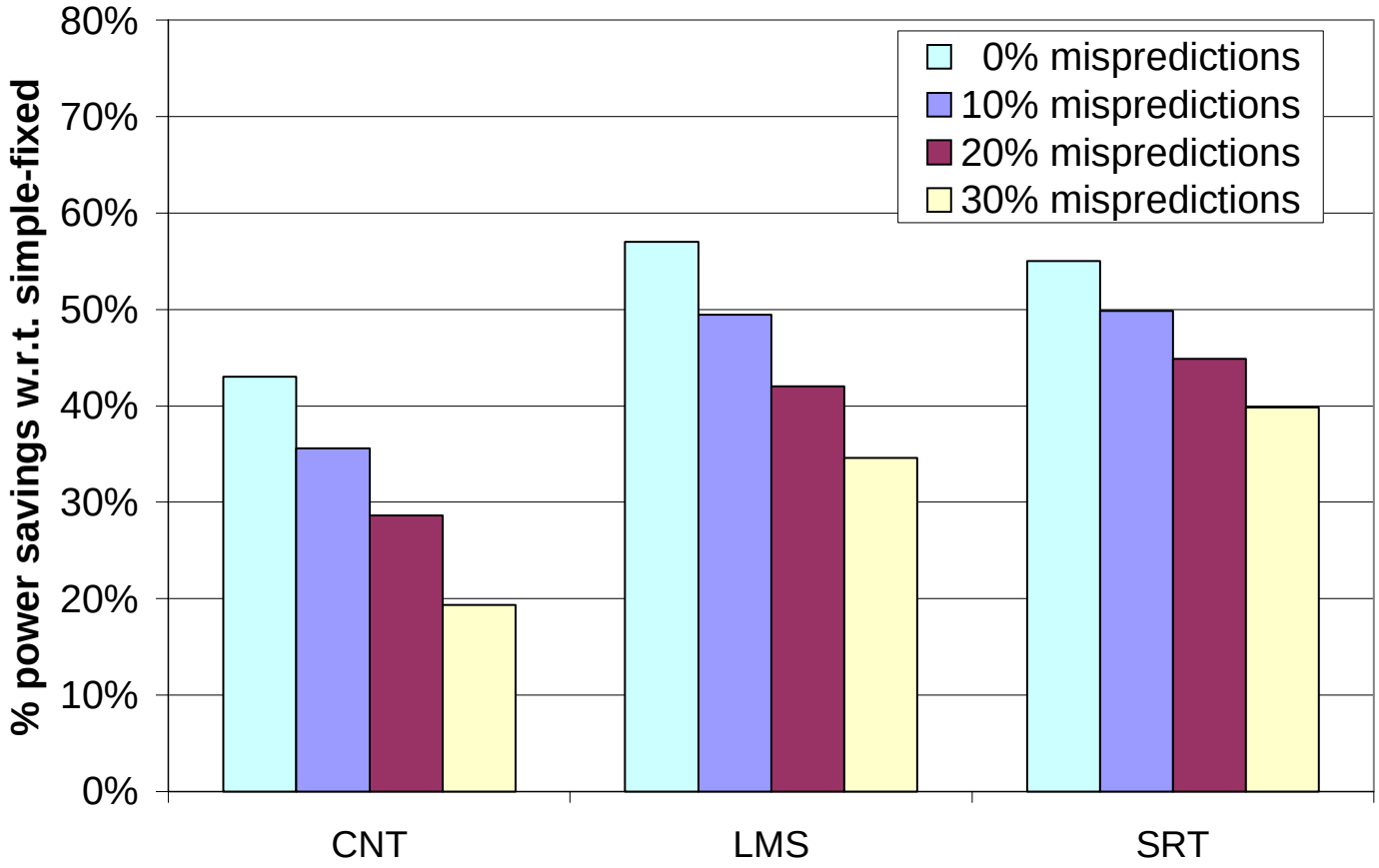
Results (cont.)

- Assume simple-fixed can operate at 1.5x frequency for same voltage



Results (cont.)

- Induce certain percentage of missed checkpoints
  - Set f_{spec} lower than recommended



Summary

- VISA shields worst-case timing analysis from underlying microarchitecture
  - Expedites introduction of contemporary processors in safe real-time systems
- Higher performance creates slack
  - Power/energy savings via DVS
  - Higher concurrency in mixed-task systems
  - Arbitrarily complex SMT processors feasible
- 43-61% less power than explicitly-safe simple processor

Summary (cont.)