Real Time Systems
EEE6595 2/14/2013

Prateek Thakyal

Worst-Case Execution Time Analysis for Parallel Run-Time Monitoring

Daniel Lo and G. Edward Suh, Cornell University
DAC 2012, June 3-7, 2012, SF, CA, USA

Outline

1. Real Time Systems
2. Real Time Systems Space
3. Reliability and Security
4. Impact of Parallel Monitoring on WCET
5. Parallel Monitoring Challenges
   a. WCET (Basics)
6. Parallel Monitoring Advantages
7. Parallel Monitoring Architecture
8. Integer Linear Programming
9. Formulation
10. Results
11. Future Work

Real Time Systems

A system is said to be *real-time* if the **total correctness** of an operation depends upon :

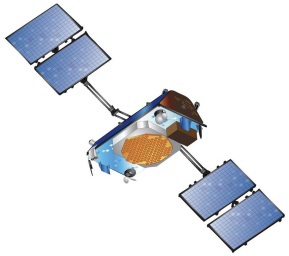
1. logical correctness, and
2. **the time in which it is performed.**

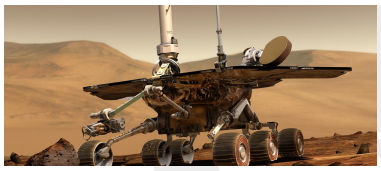
Operations are like Fire brigades - arrival and arrival time both are equally important

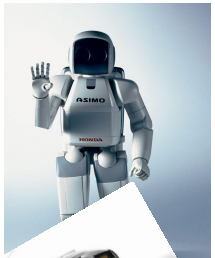
Real Time Systems Space (1/3)

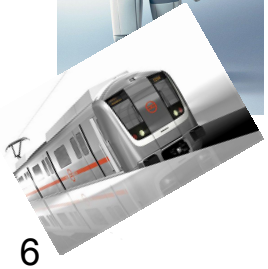
- Home Appliances
- Transport: Cars/ Aeroplanes etc.
- Personal Electronics
- Robots : Mars rover etc
- Medical Appliances
- Buildings

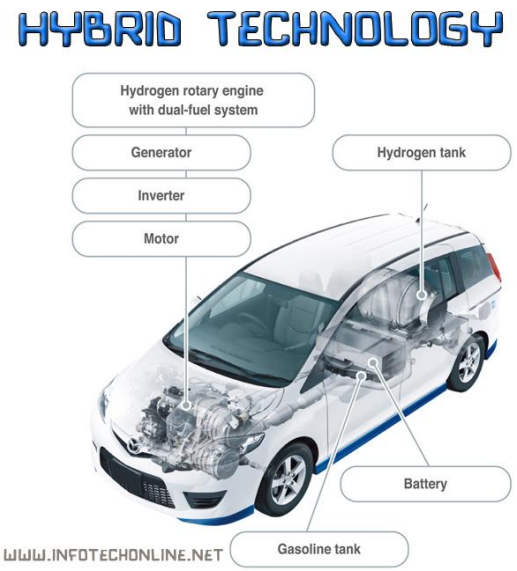
Real Time Systems Space (2/3)











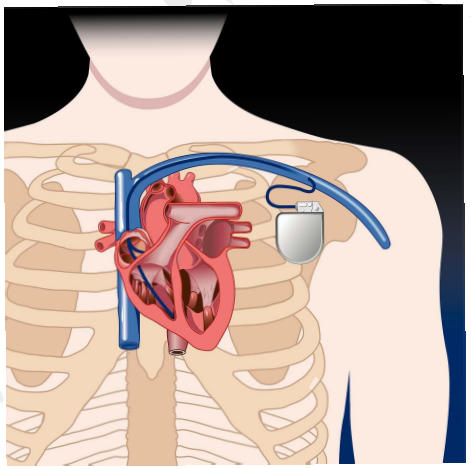
Real Time Systems Space (3/3)

Space and Complexity of the Real Time Systems is Growing More and More...

Reliability and Security

Reliability and Security

- a key concern for Emerging Technologies.
- Reliability or Security breach in critical applications (e.g. Medical Applications, may cause physical damage loss of life)



Growing need for Monitoring tasks in parallel to hunt down Security/ Reliability Issues

Reliability & Security Issues

Untrusted I/Os Operation
Uninitialized Memory Read
Control Flow Corruption

Monitoring of the task is known to significantly improve Reliability and Security

Parallel Monitoring Challenges

1. Run time overload
   - Incorporating Monitoring sequential to the main tasks an absolute killer for the Designers to add new applications
   - Parallel Monitoring helps but may still issues stalls to the Main compute
2. Power
   - Continuous Parallel Monitoring comes at the cost of power
3. Area
   - Parallel Monitoring requires additional hardware/ area
4. Scheduling Overhead for the RTOS
5. Lack of timing guarantees
   - Forbids inclusion in Critical RTS
   - Need for estimate of the WCET of the monitoring processes
   - RTOS needs to know the WCET

WCET

(Worst Case Execution Time)

Definition: an Upper bound on the execution time of a task [Peter1]

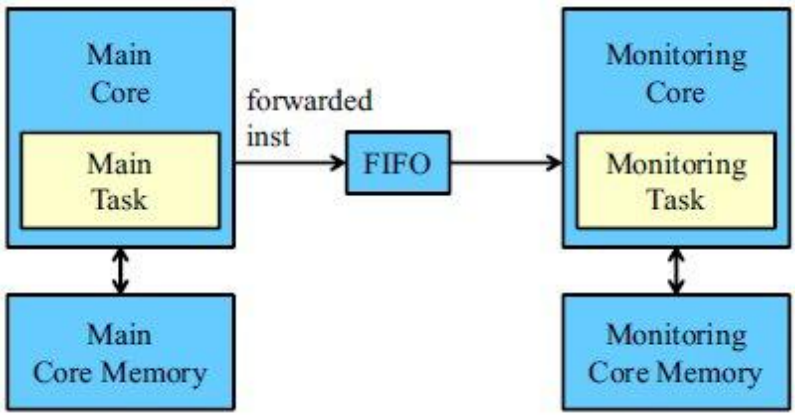
Required by the Operating system to schedule talks and provide *real Time guarantees*.

"Mr. Barnes is expecting you, but he's currently in a chess game. So, he'll be with you in a few minutes, or several hours."

Parallel Monitoring Advantages

1. Enables many new capabilities:
   a. fine-grained memory protection
   b. error bound checks
   c. hardware errors
2. Protection against large class of software attacks
3. High reduction (orders of tens of percent) in monitoring run time compared to single core monitoring

Parallel Monitoring Architecture Mod



- Main and Monitoring core loosely coupled through a FIFO buffer
- Forwarded Instruction:
  - determined based on monitoring technique
  - sent transparently (no explicit inst in main task)
  - triggers series of monitoring instructions
- If FIFO full:
  - Main core needs to wait/**stall** on forwarded inst till FIFO available. Referred to as Monitoring stall

Monitoring Technique

UMC(Uninitialized Memory Check)

a. Monitoring Core detects bugs that read memory location before being written.
b. Load/ Store instructions forwarded by Main core to Mon core
c. On store Mon sets a tag bit corresponding to the location
d. On load, mon core checks the tag bit and raises exception if not set

Paper focus and Assumptions

1. Analyses focuses on Main core and Monitoring core interactions through the FIFO
2. Monitoring core assumed to have separate memory (no shared resource cycle loss)
3. No timing anomalies in the main core - required to assume that monitoring stalls produce WCET on the main core
4. WCET of a main task and a monitoring task on the different cores may be estimated individually
5. Enough loop iterations for the FIFO to become full.

Worst Case Execution Time Analysis

Classic WCET Analysis: Implicit Path Enumeration

1. Convert the program into a control flow graph (CFG)
2. Formulate ILP to maximize

$$t = \sum_{B \in \mathcal{B}_{CFG}} N_B \cdot c_{B,max}$$

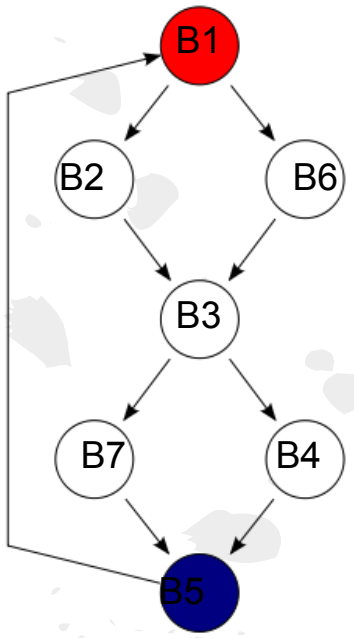
where, B_{CFG} :set of basic blocks in the CFG

N_B :# of times block B is executed

$C_{B,max}$: Max cycles to execute B

3. In case of branches take only one branch
4. Put constraints on N_B to account for only certain paths getting executed

<u>Maximum Value of "t" gives WCET</u>



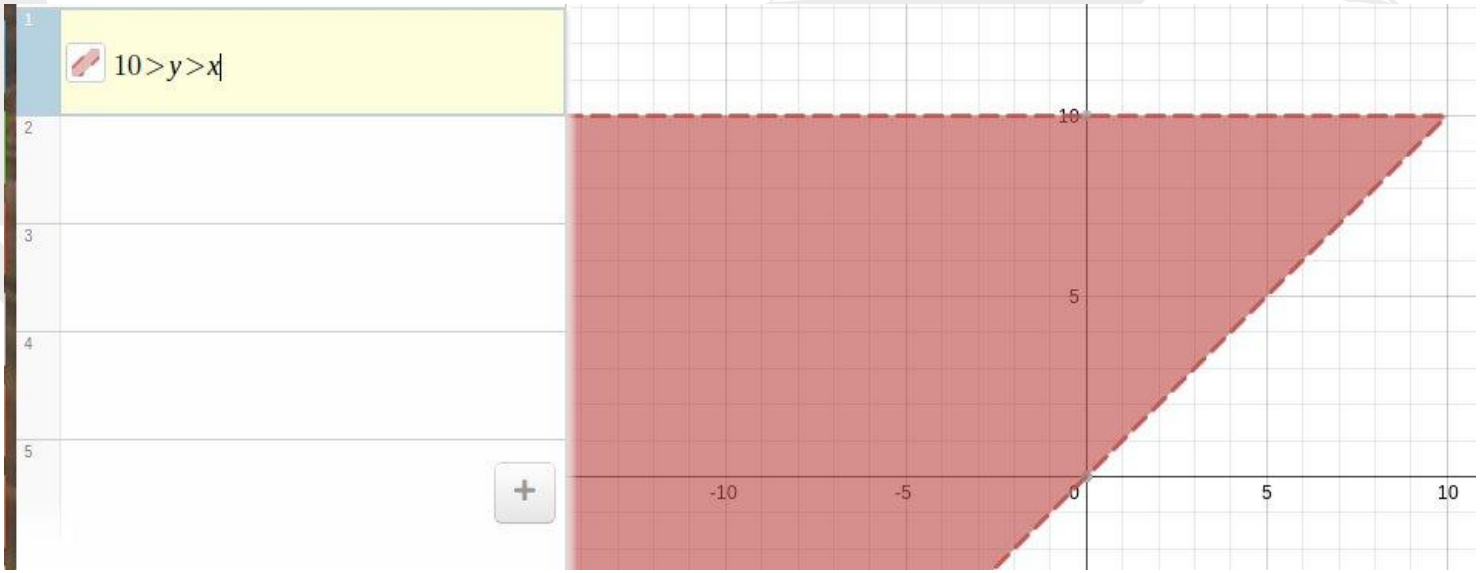
Parallel Monitoring WCET Analysis

Classic ILP formulation may be extended to account for the Monitoring stalls per block:

$$t = \sum_{B \in \mathcal{B}_{CFG}} N_B \cdot (c_{B,max} + s_{B,max})$$

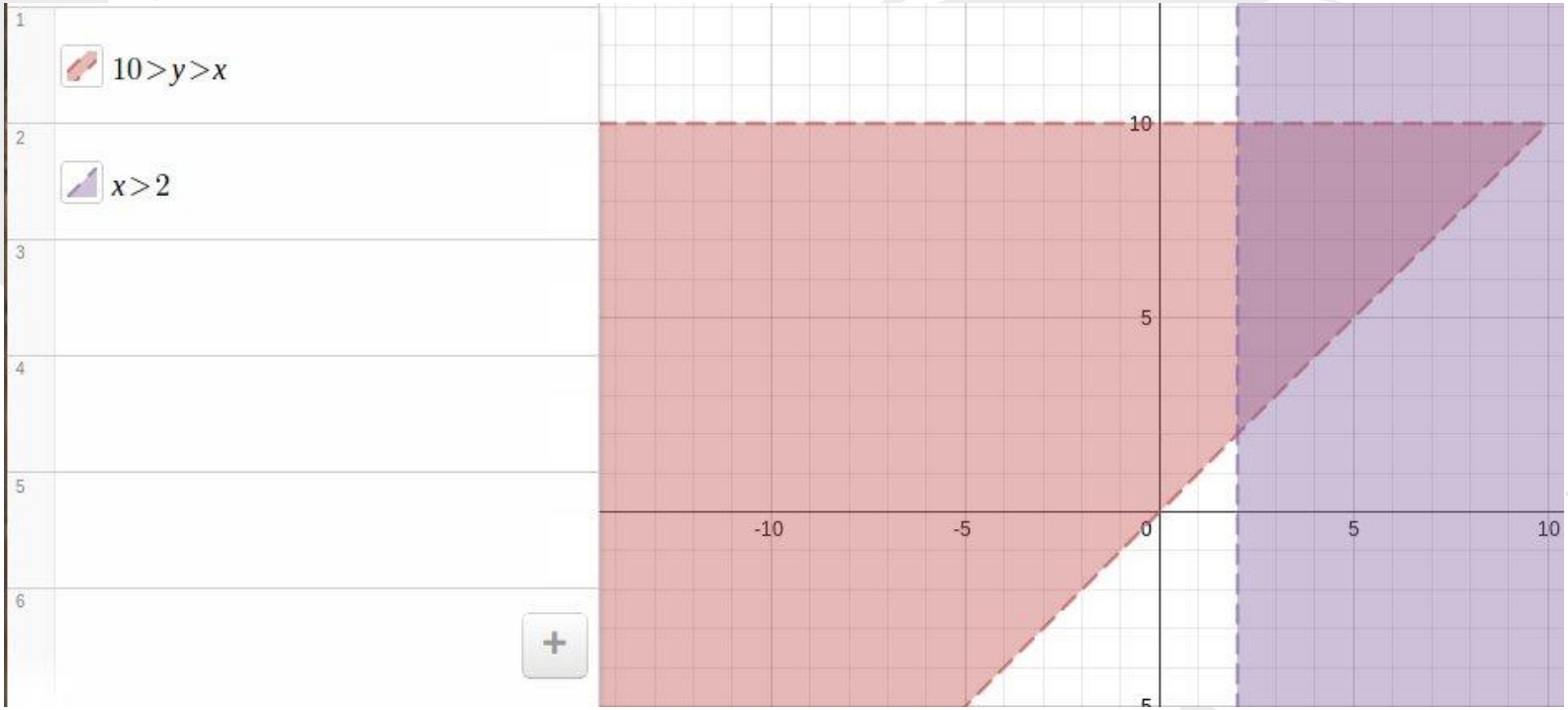
where, $s_{B,max}$: max # of cycles that B is stalled due to monitoring

Integer Linear Programing: Basics



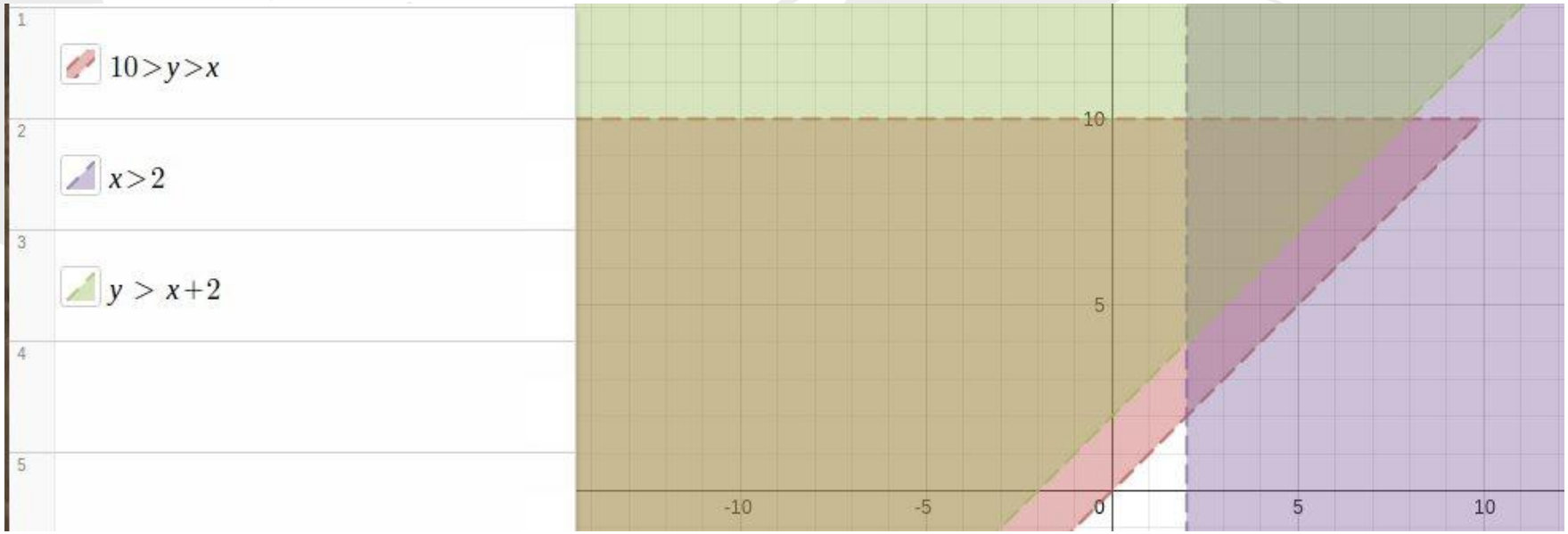
1. Boundary conditions provide the Area of interest
2. In the Area of interest, we may choose a find the points for a maxima of a function

Integer Linear Programing: Basics



1. Boundary conditions provide the Area of interest
2. In the Area of interest, we may choose a find the points for a maxima of a function

Integer Linear Programing: Basics

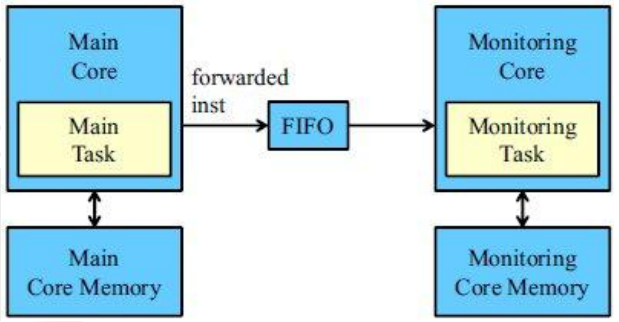


1. Boundary conditions provide the Area of interest
2. In the Area of interest, we may choose a find the points for a maxima of a function

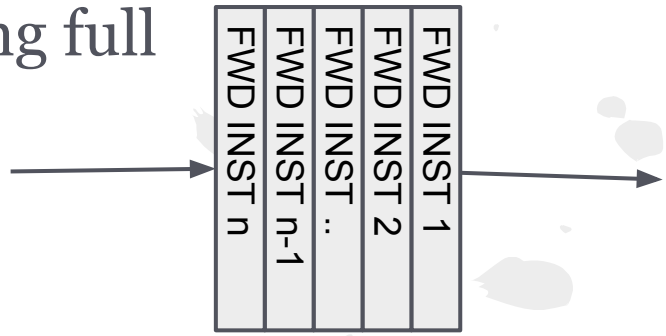
Sequential Monitoring Bound

- Conservative Bound on the worst-case monitoring stalls cycles
  - monitoring task run in line with the main task on the same core
  - WCET may be attained in the traditional way, by having a single program for monitoring and main execution
  - may causes a monitoring stall for every instruction
  - Extremely conservative compared to parallel execution of monitoring task by coupling through a FIFO

FIFO Model (1/2)



- Main task
  - continues as long as FIFO entry available
  - stalls when FIFO full
- WCET model needs to capture
  - the worst-case (maximum) number of entries in the FIFO at each forwarded instruction
  - determine how many cycles the main task may be stalled due to the FIFO being full



FIFO Model (2/2)

Monitoring Flow Graph (MFG)

CFG is transformed so that each node contains at most one forwarded instruction

a. forwarded inst to be located at the end of the code represented by the node

Monitoring Load

# of cycles required for the monitoring core to process all outstanding entries in the FIFO at a given point in time

Monitoring Load

Challenge:

Mathematical Modeling of FIFO at entry by entry level

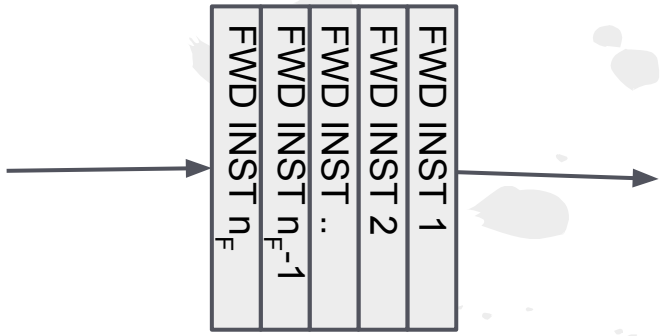
Simplification:

$t_{M,max}$: Increase in monitoring load for any forwarded instruction = max(worst case monitoring task execution time for any forwarded instruction)

Bound: 0 <= Monitoring Load <= Maximum monitoring load FIFO can handle lmax

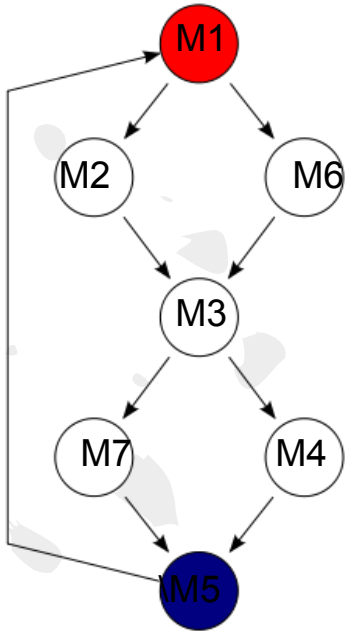
Monitoring Load = n_F x $t_{M,max}$

Where n_F : # of FIFO entries



Monitoring Flow Graph

Node Mx Represents Monitoring Graph Node



Worst case Stall Cycles

Change in Monitoring Load at node M

li_M : Monitoring Load coming into the Node M

lo_M : Monitoring Load going out of the node M

Delta l_M : Change in the Monitoring load "M"

$t_{M,max}$ = WCET of Monitoring Task

$C_{M,min}$ = Minimum cycles to execute Mon

$$\Delta l_M = \begin{cases} t_{M,max} - c_{M,min}, & \text{forwarded inst.} \in M \\ -c_{M,min}, & \text{no forwarded inst.} \in M \end{cases}$$

Output Monitoring Load at node M

$$lo_M = \begin{cases} 0, & li_M + \Delta l_M < 0 \\ li_M + \Delta l_M, & 0 \leq li_M + \Delta l_M \leq l_{max} \\ l_{max}, & li_M + \Delta l_M > l_{max} \end{cases}$$

$$l_{max} = n_F \cdot t_{M,max}$$

Input Load of node M due to Previous nodes

$$li_M = \max_{M_{prev} \in \mathcal{M}_{prev}} lo_{M_{prev}}$$

Stall occurs when forwarded instruction is executed but still no entery in FIFO is free

S_M : Number of cycles stalled

$$s_M = \begin{cases} 0, & li_M + \Delta l_M < l_{max} \\ (li_M + \Delta l_M) - l_{max}, & li_M + \Delta l_M \geq l_{max} \end{cases}$$

Worst case monitoring stall cycle for a given node M

$$\max \sum_{M \in \mathcal{M}_{MFG}} s_M$$

Results

Monitoring	Experiment	Benchmark						
		cnt	expint	fdct	fibcall	insertsort	matmult	ns
None	wcet-none	64531	3483	1805	245	598	133668	5951
	sim-none	62931	2293	1805	245	598	133668	5951
UMC	sequential-umc	103052	3591	4382	257	2489	357453	10338
	wcet-umc	64550	3498	3035	245	2083	256120	5953
	sim-umc	62931	2297	2564	245	1864	235120	5951
CFP	sequential-cfp	151732	11669	1976	794	1174	231507	18623
	wcet-cfp	93544	8984	1805	547	677	133668	13614
	sim-cfp	72540	5247	1805	382	598	133668	9824

Estimated and Observed WCET (clock cycles) with and without monitoring

Results (Ratio)

Ratio			Benchmark						
			cnt	expint	fdct	fibcall	insertsort	matmult	ns
wcet-none	:	sim-none	1.03	1.52	1.00	1.00	1.00	1.00	1.00
wcet-umc	:	sim-umc	1.03	1.52	1.18	1.00	1.12	1.09	1.00
wcet-cfp	:	sim-cfp	1.29	1.71	1.00	1.43	1.13	1.00	1.39
sequential-umc	:	wcet-umc	1.60	1.03	1.44	1.05	1.19	1.40	1.74
sequential-cfp	:	wcet-cfp	1.62	1.30	1.09	1.45	1.73	1.73	1.37
wcet-umc	:	wcet-none	1.00	1.00	1.68	1.00	3.48	1.92	1.00
wcet-cfp	:	wcet-none	1.45	2.58	1.00	2.23	1.13	1.00	2.29

Ratios Comparing Results from different Experiments

Conclusion

- Parallel Monitoring an attractive solution for improving the safety and reliability of future real-time systems.
- WCET of the P Mon techniques needs to be analyzed before they may be applied
- Method for estimating the WCET for tasks running on a PMon system presented
- Non-linear FIFO behavior modeled as an MILP problem to produce the worst-case monitoring stall cycles
- WCET monitoring stall cycles may be incorporated into traditional IPET methods for WCET estimation.
- Evaluation shows significant improvements over an estimate assuming sequential execution of the monitoring.
- Amount of overestimation is comparable to the overestimation for a system without parallel monitoring.

Future work and Improvements

1. Tighten the WCET bound
   a. Improve by incorporating more info about the main task
   b. Incorporate loop bounds and infeasible paths
2. Improve the time needed to solve the linear programming Problem
3. Architectural features
   a. Shared memory analysis
4. Non-linear programming Techniques

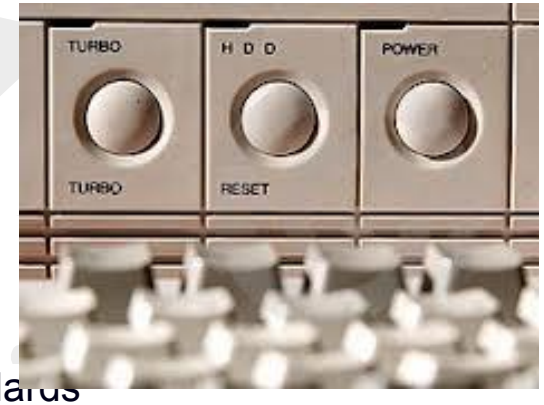
Leakage-Aware Dynamic Scheduling for Real-Time Adaptive Applications on Multiprocessor Systems

Heng Yu, Bharadwaj Veeravalli and Yajun Ha, National University of Singapore
DAC June 13-18 2012

Outline

1. Adaptable Applications
2. Types of Power dissipation
3. Leakage Power
4. What all can reduce Power?
5. Slack saving in 2 processor system
6. Minimum Energy at given Frequency
7. Frequency and Min Energy Settings
8. Slack Receiver Selection
9. Guided Search Heuristics
10. Results
11. Conclusion and Improvement Area

Adaptable Applications World



Advantages

+ scalable performance quality as per the environment

+ more program cycles and/or energy budget assigned

    o higher performance quality it achieves (till a threshold)

Examples:

1. Scalable Video Coding (SVC) scheme in H.264/ MPEG-4 standards

+ Customized service quality to accommodate n/w and device conditions

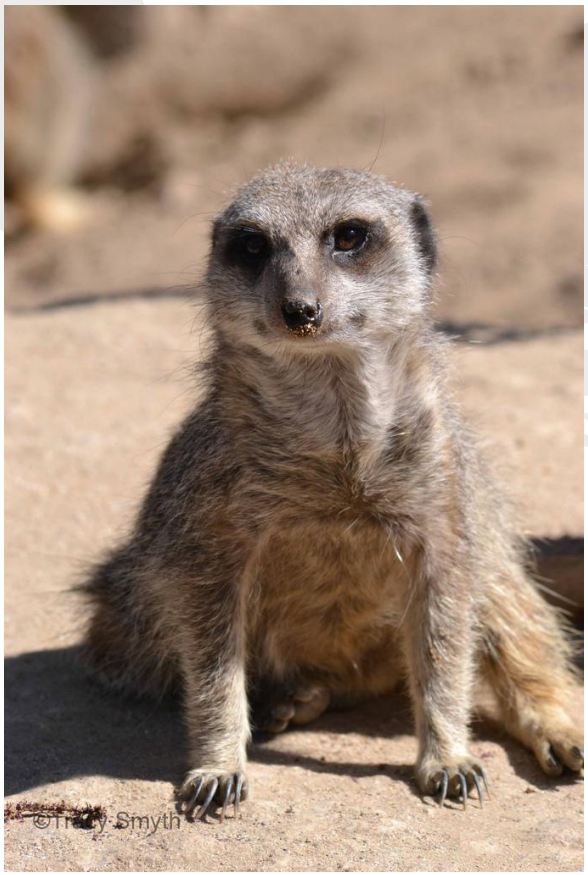
2, JPEG2000 codec : Multiple playback resolutions.

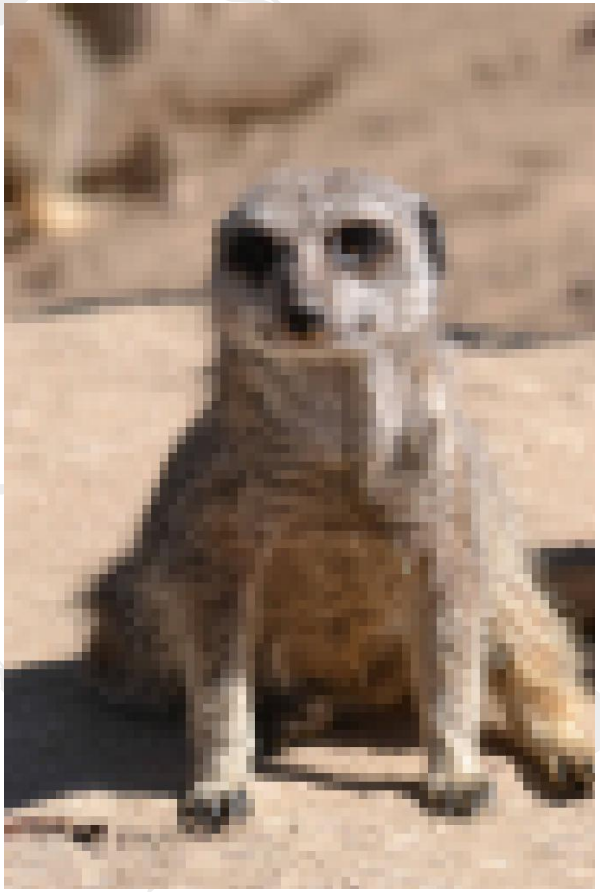
Instead of completing or failing the execution, adaptive applications usually define multiple execution granularities with finer grained with better results

+ Cost of increased program cycles and energy

Strong Motivation for Low Power Vs Performance Tradeoffs

Scalable App Example





Types of Power Dissipation

1. Dynamic Power
    + Power in Charging and discharging of Loads
    + Depends on
        - Toggle Rate and Frequency
        - Vdd
2. Leakage Power
    + Power lost when the device is off
    + Depends on Vdd, Vbs and process parameters
3. Short circuit Power
    + Depends on
        - O/P Load, I/p Slew, Vdd, Toggle, Freq

Leakage Power Trends

With technology scaling the leakage power is getting high and becoming comparable to the dynamic power

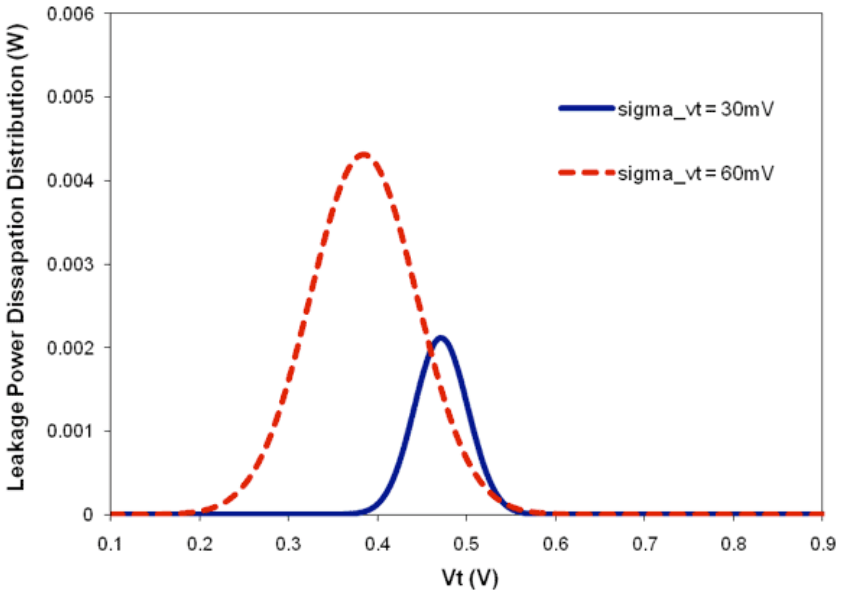


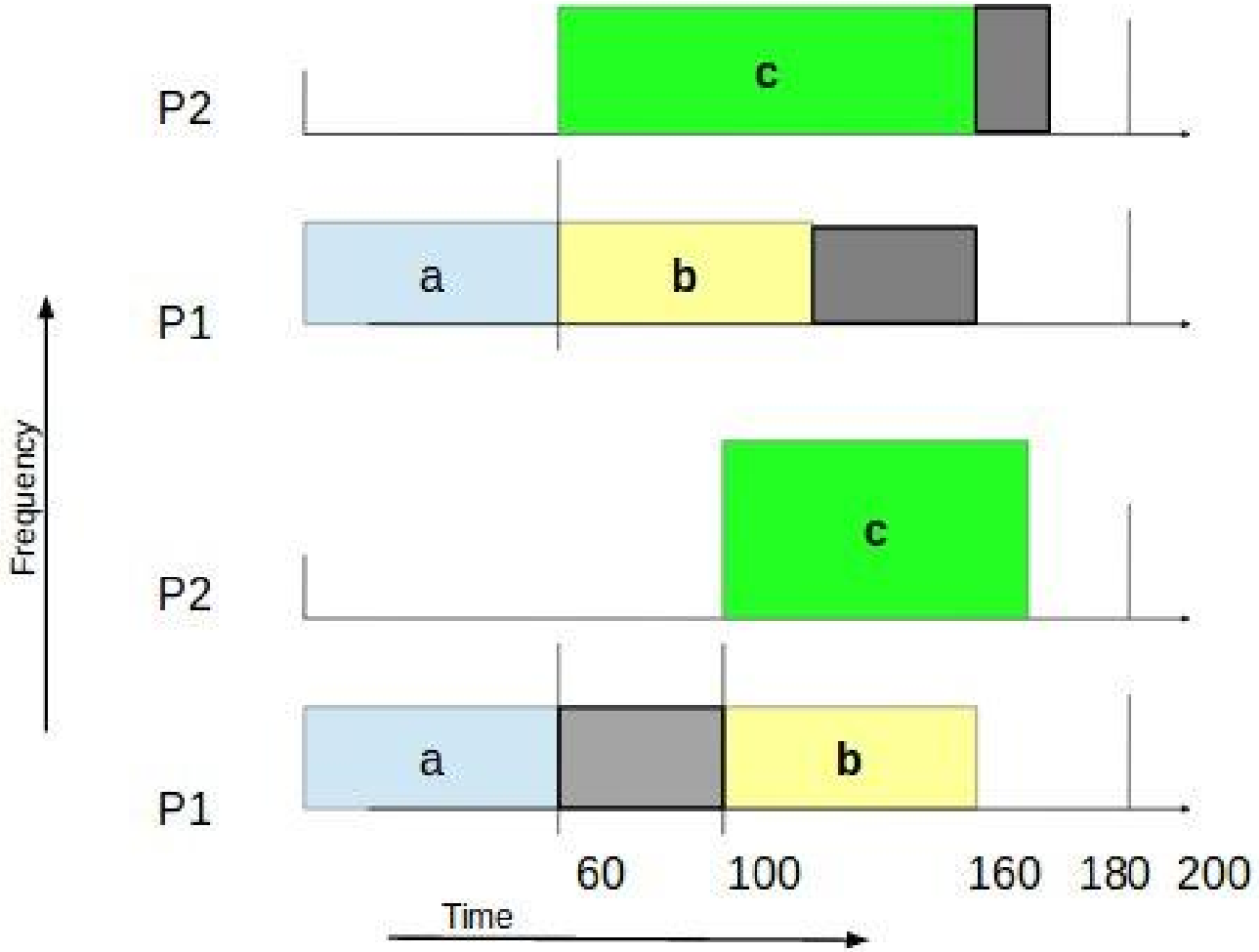
Figure 2. Since Leakage current is exponentially dependent on V_{TH}, leakage at low V_{TH} can be orders of magnitude larger than at high V_{TH}.

Increasing Need of Leakage Power Aware Scheduling Algorithms

What all can reduce power ?

1. Vdd : Supply voltage (Dynamic Voltage Scaling)
   - Pinst : directly proportional to Vdd*Vdd
   - *Decrease in Vdd by 0.7x reduces Dynamic power by Half*
2. Vbs: Bias Voltage
   - Impacts Vt and the Leakage power
3. Frequency
   - Impacts the short circuit and dynamic power
   - *Linear Relationship*
4. Turn off module completely : Heavy penalty on WCET

Slack saving in 2 processor system



Minimizing Energy at given Frequency

Total Power Specified by:

$$P = C_{eff} V_{dd}^2 f + V_{dd} K_3 e^{K_4 V_{DD}} e^{K_5 V_{BS}} + |V_{bs}| I_j$$

Energy :

$$E_{cyc} = C_{eff} V_{dd}^2 + L_g f^{-1} (V_{dd} K_3 e^{K_4 V_{dd}} e^{K_5 V_{bs}} + |V_{bs}| I_j)$$

Lg : Logic path length of the circuit

Frequency Selection:

$$f = (L_d K_6)^{-1} ((1 + K_1) V_{dd} + K_2 V_{bs} - V_{th1})^{\alpha}$$

By adjusting (Vdd,Vbs) values Ecyc can be minimized at a given frequency

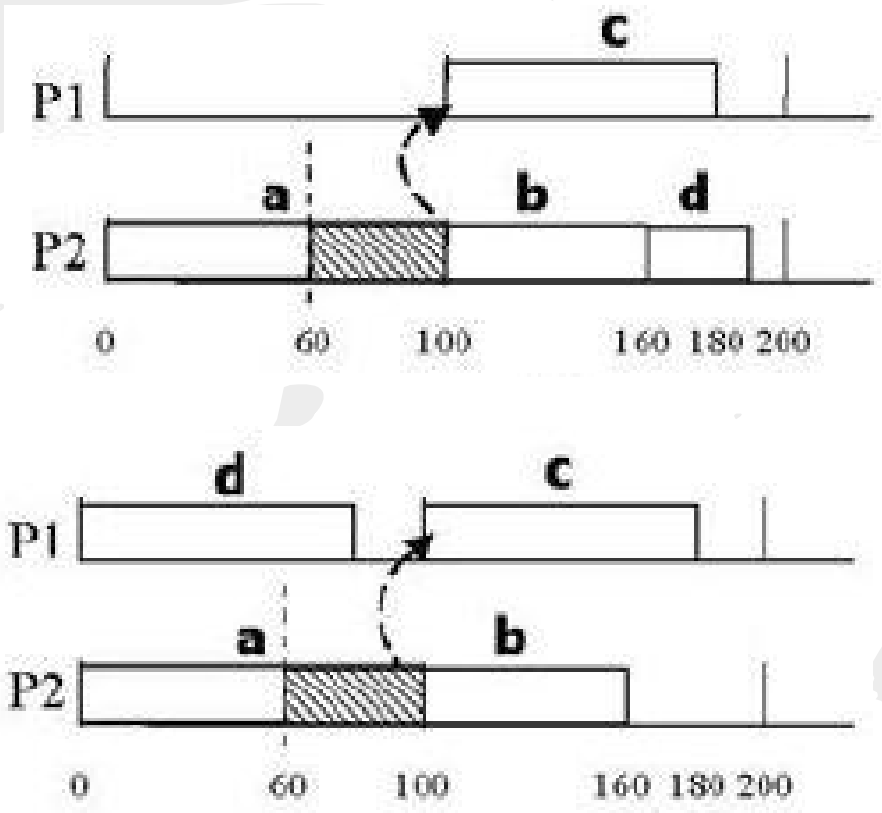
Frequency and Min Energy Settings

For each frequency in the set of available frequencies {f1, f2,…,fj} choose Vdd and Vbs in order to get minimum minimum Leakage Power

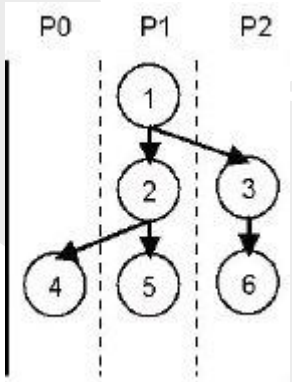
Slack Receiver Selection(1/2)

Issues with Greedy based receiver selection for choosing direct descendant task:

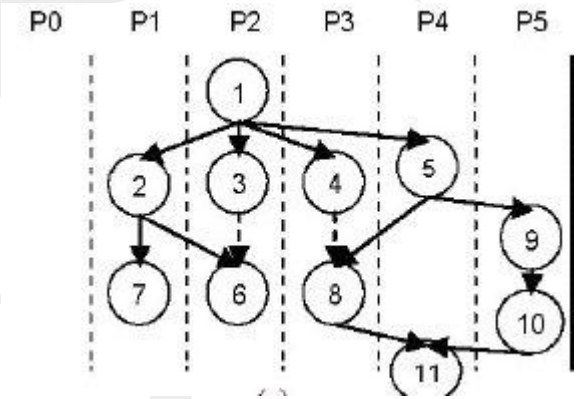
1. Direct Receivers may not fully utilize the slack time
2. Additional parallel candidates for slack distribution.



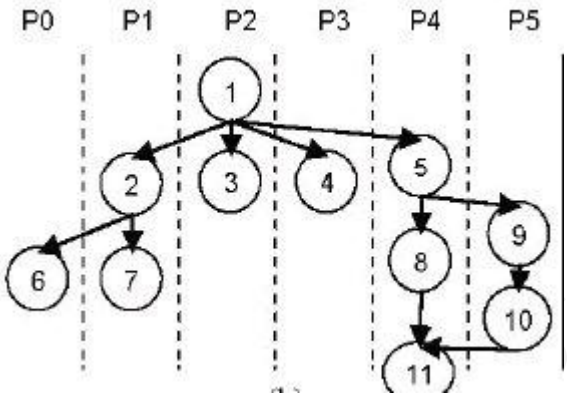
Slack Receiver Selection(2/2)



Candidates for receiving Slack from T1:
{T2, T3}, {T2, T6},
{T4, T5, T6}, and
{T4, T5, T3}.



Candidate Set (abbr. CS) of a slack generator is a set of slack receivers that fully adopts the slack time.



Guided Search Heuristics

A guided-search heuristics to select the "best-fit" frequency levels that maximize the additional program cycles of adaptive tasks.

Objective:

1. Maximize or Minimize Frequency so as to consume all the slack from previous node

2. Constrain the search in 1, with the Energy.

Maximize

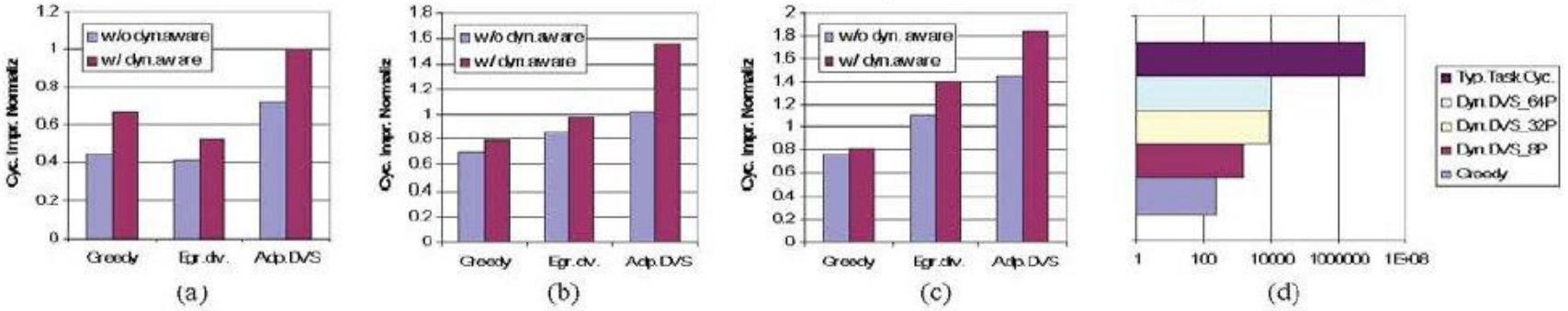
$$\sum_{T_i \in T} \Delta c_i$$

Subject to

$$\frac{c_i + \Delta c_i}{f_{i,new}} \le \frac{c_i}{f_{i,old}} + t_{s,i} + \Delta t_{oh}, \forall T_i \in T$$

$$\sum_{T_i \in T} ((c_i + \Delta c_i) E_{cyc}^{f_{j,new}}) \le E_s + \sum_{T_i \in T} (c_i E_{cyc}^{f_{j,old}}) + \Delta E_{oh}$$

Results



Performance over 2.5 times over even-energy approach, 31.6% better than the greed approach

Conclusion, Future Work and Improvement Areas

- Novel framework proposed for leakage aware multiprocessor dynamic scheduling on adaptive applications
- Efficient Slack distribution technique demonstrated for Leakage Aware Dynamic Scheduling
- *Approach does not take into account the toggle rate of the system.*
- *A processor may have multiple task running at a given time : Analysis and Algorithm needs to be based on multi threading options*
- *Overhead of different voltage levels needs to be studied*

Questions