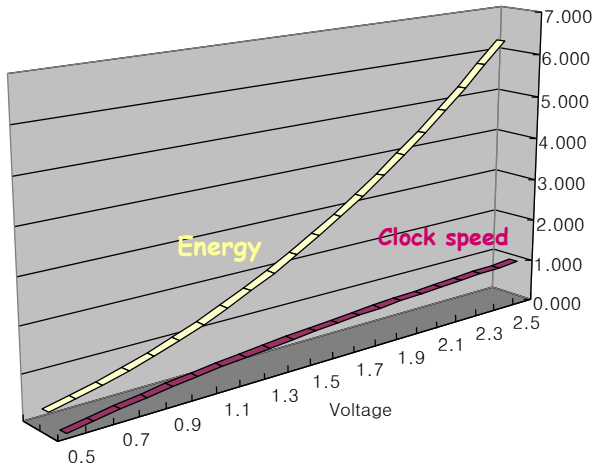
Software-level Power-Aware Computing

Lecture 2

Lecture Organizations

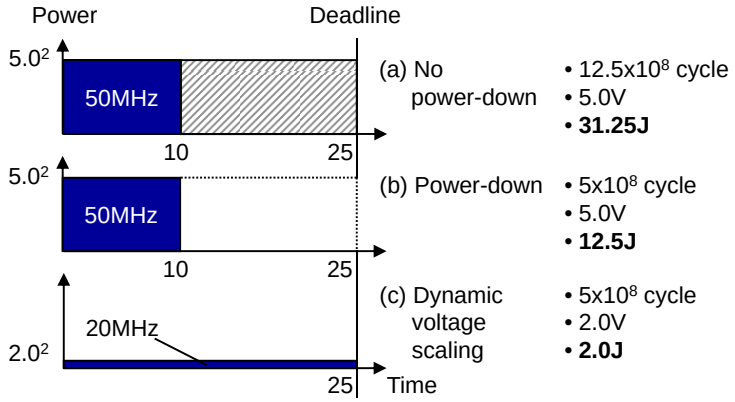
- Lecture 1:
  - Introduction to Low-power systems
  - Low-power binary encoding
  - Power-aware compiler techniques
- Lectures 2 & 3
  - Dynamic voltage scaling (DVS) techniques
    - OS-level DVS: Inter-Task DVS
    - Compiler-level DVS: Intra-Task DVS
    - Application-level DVS
  - Dynamic power management
- Lecture 4
  - Software power estimation & optimization
  - Low-power techniques for multiprocessor systems
  - Leakage reduction techniques

Voltage, Frequency & Energy



Basic Idea of DVS

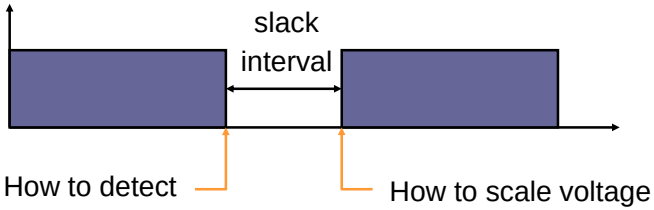
- $E \propto N_{cycle} \cdot V_{DD}^2$



→ **Slow and Steady wins the race!**

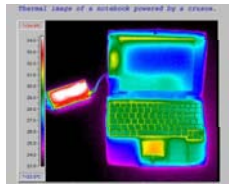
Key Issues for successful DVS

- Efficient Detection of Slack/Idle Intervals
- Efficient Voltage Scaling Policy for Slack Intervals



Commercial DVS Processors

- Transmeta Crusoe
- AMD K2+ (PowerNow Technology)
- Intel SpeedStep
- XScale

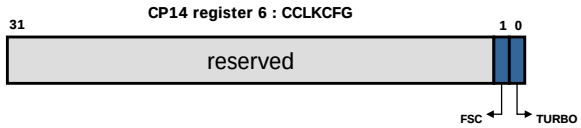


Voltage Scaling Processors

	Commercial			Academic	
Processors	Transmeta Crusoe (LongRun)	AMD Mobile K6 (PowerNow)	Intel PXA250	UC Berkely (ARM8)	Ubicom LART(StrongARM)
Scaling Level	200~700MHz 1.1~1.65V	192~588MHz 0.9~2.0V	100~400MHz 0.85~1.3V	5~80MHz 1.2~3.8V	59~251MHz 0.79~1.65V
Scaling Time	1.1 ↔ 1.65V < 300μs	0.9 ↔2.0V 200μs	Each step 500μs	1.2 ↔ 3.8V 520μs	59↔251MHz : 140μs 0.79→1.65V : 40μs 0.79←1.65V : 5.5ms
Scaling Power	??	??	??	130μJ	??

DVS Support in PXA250

- Use Two Registers in PXA250 Xscale Core
  - CCCR (Core Clock Configuration Register):
    - Specify memory clock & core clock
  - CCLKCFG (Core Clock Configuration) Register
    - Set FCS (Frequency Change Sequence) bit to change the clock speed



Change if FCS bit = 1

CCCR Setting

0x41300000 : CCCR

31	9	8	7	6	5	4	3	2	1	0
reserved			N			M		L		

L	M	N							
		2		3		4		6	
1	1	99.5	.85V			199.1	1.0V	298.6	1.1V
2	1	118.0	1.0V			235.9	1.1V	353.9	1.3V
3	1	132.7				265.4		398.9	
4	1	147.5				294.9			
5	1	165.9				331.8	1.3V		
1	2	199.1		298.6	1.1V	398.1	1.3V		
2	2	235.9	1.1V						
3	2	265.4							
4	2	294.9							
5	2	331.9	1.3V						

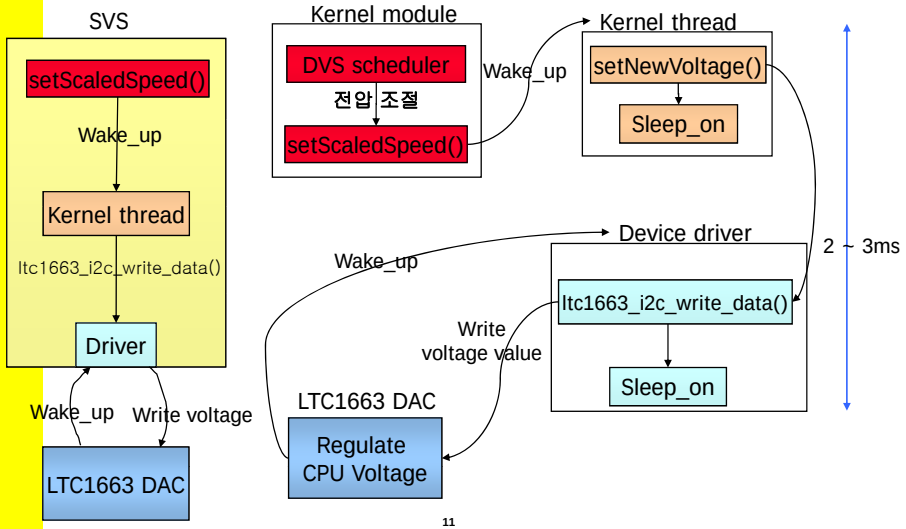
Example Voltage Scaling Code

```
1  #include <machine/pmu.h>
2  #include <machine/cp14.h>
3
4  int thread_args[3] = {0, 1, 2};
5
6  void
7  change_clock_speed(int speed)
8  {
9      int settings[20]={ 0, 0x121, 0x122, 0x123, 0x124, 0x125, 0x1a2,
10                0x141, 0x1a4, 0x142, 0x1a5, 0x143, 0x144, 0x145 };
11     int cccr_val = 0x121, clkcfg_val = 2;
12
13     cccr_val = settings[speed];
14     switch (speed) {
15               case 6  : clkcfg_val = 3; break;
16               case 8  : clkcfg_val = 3; break;
17               case 10 : clkcfg_val = 3; break;
18               default : clkcfg_val = 2; break;
19     }
20     memcpy(0x40000000+0x1300000, &cccr_val, 4);
21     CP14_WRTIE_CCLKCFG(clkcfg_val);
22 }
23
24 int
25 get_os_time()
26 {
27               int ostime;
28
29               memcpy(&ostime, 0x40000000+0xa00010, 4);
30               return(ostime);
31 }
```

```
void Main(void)
{
    int i, bb, cc, j;

    for ( k = 1 ; k < 13 ; k++ ) {
        change_clock_speed(k);
        bb = get_os_time();
        for ( i = 0 ; i < 10000 ; i++ ) j = 10;
        cc = get_os_time();
        printf("%d : %d \n", k, cc - bb);
    }
}
```


Voltage Scaling in Linux



ARM IEM DEMO

Successful Low Power S/W Techniques

1. Understand workload variations of your target
2. Devise efficient ways to detect them
3. Devise efficient ways to utilize the detected workload variations using available H/W supports

Roadmap

- DVS in Non Real-Time Systems
- DVS in Real-Time Systems
  - Compiler-level DVS: Intra-task DVS
  - OS-level DVS: Inter-task DVS
  - Application-level DVS
    - MPEG-decoder implementation
  - Algorithm-level DVS
    - Low-power convolution

Non Real-Time Jobs

- Non Real-Time Jobs
  - No timing constraints
  - No periodic executions
  - Unknown WCET

It is hard to predict the future workload!!

DVS for Non Real-Time Jobs

- Basic Approach:
  - Predict workload based on history information
  - Usually based on *some variations of interval scheduler*
    - PAST, FLAT
    - LONG_SHORT, AGED_AVERAGE
    - CYCLE, PATTERN, PEAK

Key Question

How can we predict the future workload?

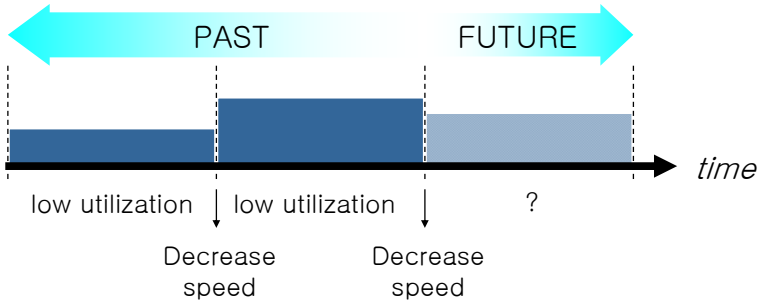
- Based on long term history:
  Hard to adapt quickly for the changed workload
- Based on short term history:
  Too many clock/voltage changes

PAST

- Looking a fixed window into the past
- Assume the next window will be like the previous one
- If the past window was
  - mostly busy ⇒ increase speed
  - mostly idle ⇒ decrease speed

Example: PAST

$$\text{Utilization} = \frac{\text{busy time}}{\text{window size}}$$

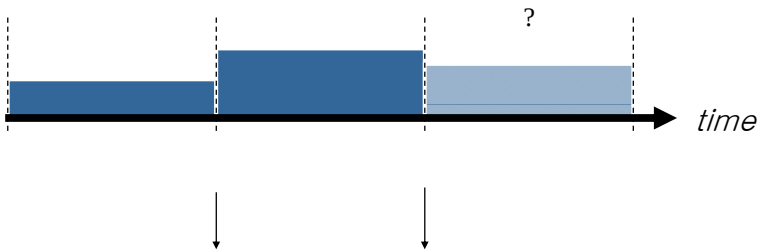


FLAT

- Try to smooth speed to a global average
- Make the utilization of next window to be <const>
  - Set speed fast enough to complete the predicted new work being pushed into the coming window

Example: FLAT

<Const>=0.7



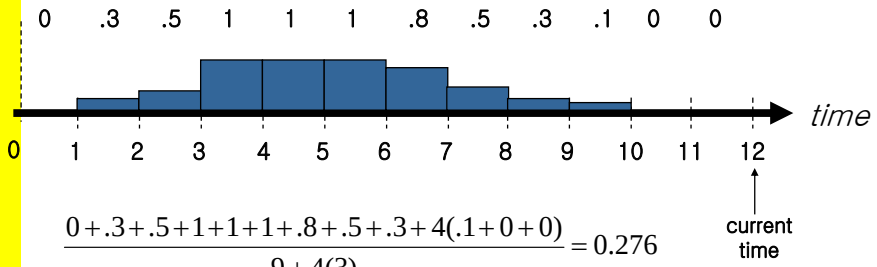
Increase/Decrease the speed the next utilization to be 0.7

LONG-SHORT

- Look up the last 12 windows
  - Short-term past : 3 most recent windows
  - Long-term past : the remaining windows
- Workload Prediction
  - the utilization of next window will be a weighted average of these 12 windows' utilizations

Example: LONG-SHORT

utilization = # cycles of busy interval / window size



$$\frac{0+.3+.5+1+1+1+.8+.5+.3+4(.1+0+0)}{9+4(3)} = 0.276$$

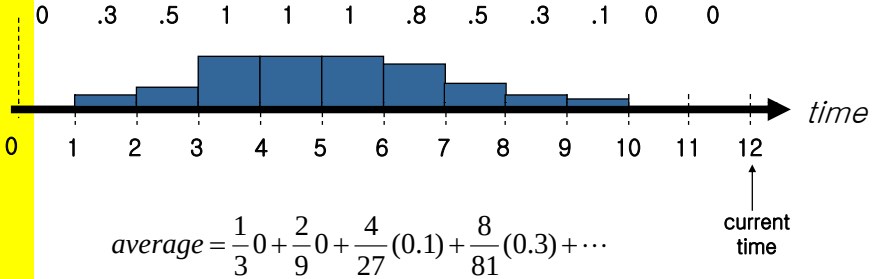
$$f_{clk} = 0.276 \times f_{\max}$$

AGED-AVERAGE

- Employs an exponential-smoothing method
- Workload Prediction
  - The utilization of next window will be a weighted average of all previous windows' utilizations
    - geometrically reduce the weight

Example: AGED_AVERAGE

utilization = # cycles of busy interval / window size

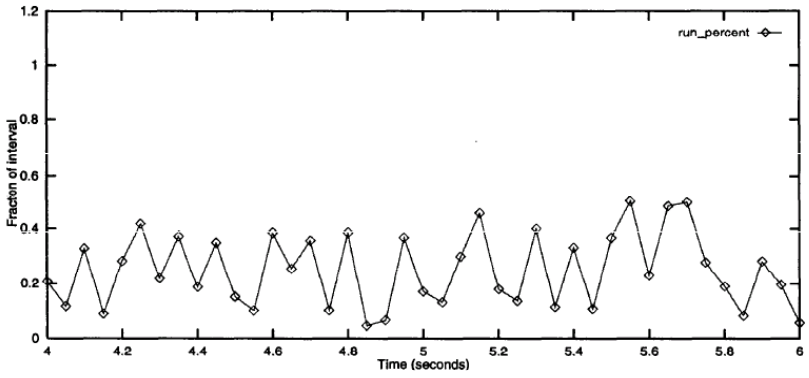


$$average = \frac{1}{3}0 + \frac{2}{9}0 + \frac{4}{27}(0.1) + \frac{8}{81}(0.3) + \cdots$$

$$f_{clk} = average \times f_{\max}$$

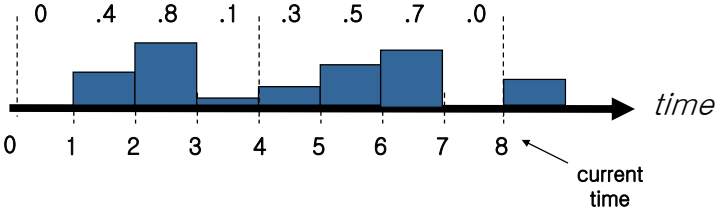
CYCLE

- Workload Prediction
  - Examine the last 16 windows
    - Does there exist a cyclic of length X?
    - If so, predict by extending this cycle
    - Otherwise, use the FLAT algorithm



Example: CYCLE

utilization = # cycles of busy interval / window size



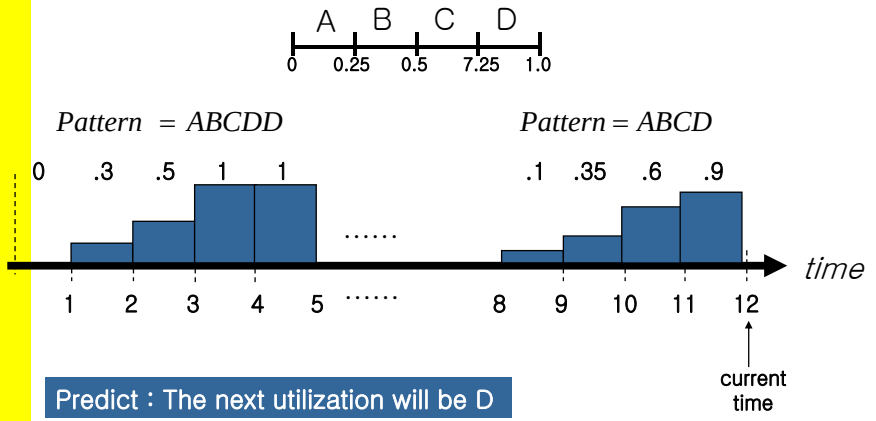
$$\text{error measure} = \frac{|0-.3|+|.4-.5|+|.8-.7|+|.1-0|}{4} = 0.15$$

Predict : The next utilization will be .3

PATTERN

- A generalized version of CYCLE
- Workload Prediction
  - Convert the n-most recent windows' utilizations into a pattern in alphabet {A, B, C, D}.
  - Find the same pattern in the past

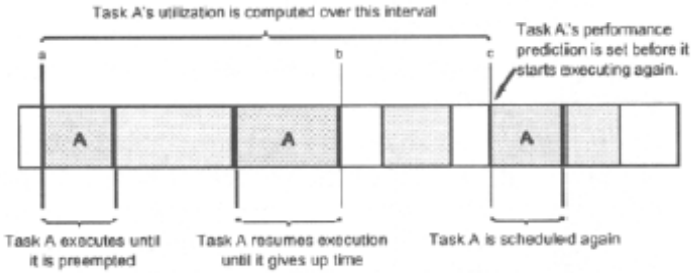
Example: PATTERN



Predict : The next utilization will be D

Perspectives-based Algorithm

- Per-Task Basis Performance Predictions



$$WorkEst_{new} = \frac{k \times WorkEst_{old} + Work_{fse}}{k+1}$$

$$Deadline_{new} = \frac{k \times Deadline_{old} + Work_{fse} + Idle}{k+1}$$

$$Perf = \frac{WorkEst}{Deadline}$$

[Flautner, OSDI2002]

Roadmap

- DVS in Non Real-Time Systems
- DVS in Real-Time Systems
  - Compiler-level DVS: Intra-task DVS
  - OS-level DVS: Inter-task DVS
  - Application-level DVS
    - MPEG-decoder implementation
  - Algorithm-level DVS
    - Low-power convolution

Two Types of DVS Algorithms

- Inter-task DVS algorithms
  - Determine the supply voltage and clock speed on *task-by-task* basis
- Intra-task DVS algorithms
  - Determine the supply voltage and clock speed within a single task boundary

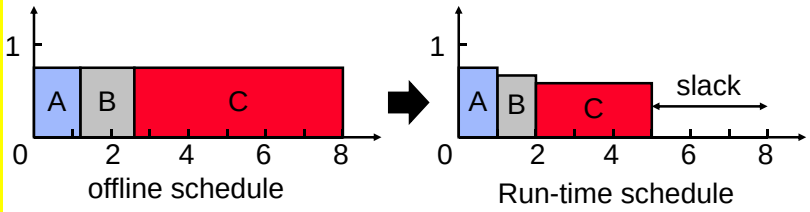
Inter-task DVS

- Inter-Task Voltage Scheduling for Hard Real-Time Systems [Yao95, Hong98, Okuma99, Shin99, Lee99].
  - Problem : Given a set of tasks, how to assign the proper speed to each task dynamically while guaranteeing all their deadlines.
  - Task-by-task Speed Assignment
    - The slack time due to a task used by following tasks, not by the current one.
  - Practical Limitations
    - Requires OS modifications
    - Cannot be applied to a single-task environment
    - Can be ineffective in a multi-task environment

An Example of Ineffective Inter DVS

task	T	D	C
A	8	4	1
B	8	4	1
C	8	8	4

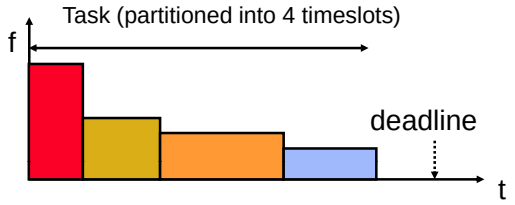
⇦ Dominant task



- A dominant task (C)
  - Exploits small slack times from other tasks.
  - Cannot use its own.

Earlier Version of Intra-task DVS

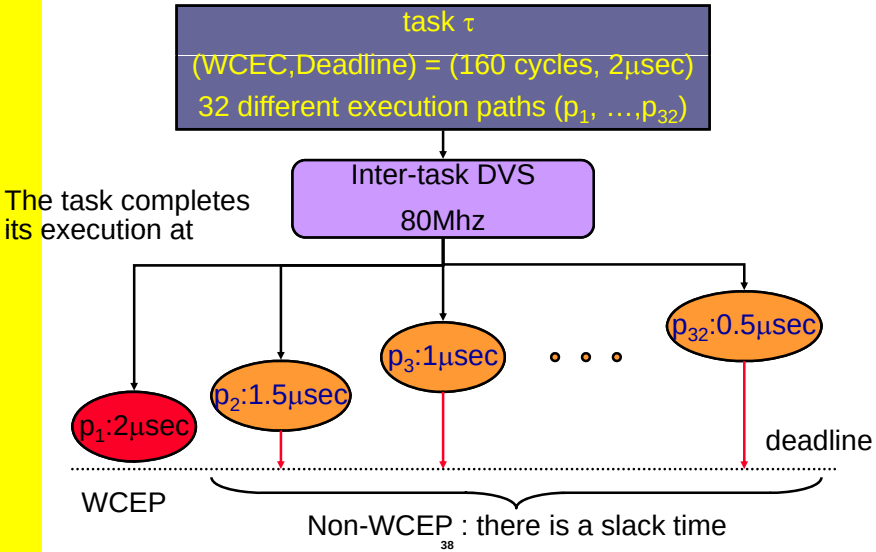
- Run-time voltage hopping [Lee00]
  - Each task is partitioned into N timeslots.
  - Frequency and voltage determined for each timeslot.
  - Voltage scheduling embedded in application programs.
  - Can be applied to conventional non-DVS OS.
    - No systematic guideline
    - Manual selection of scaling points
    - Too much burden for average programmers



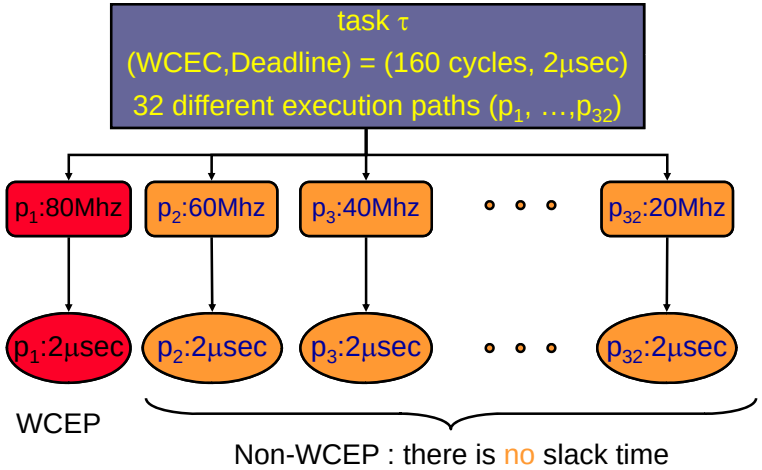
Overview of Intra-task DVS

- Intra-task voltage scheduling framework based on a static timing analysis of RT programs.
  - The clock speed is adjusted in a task by embedded codes.
  - Fully exploits all slack times coming from execution time variations within a single task
  - No OS modification
  - Applicable to a single-task environment.
  - Provides a systematic methodology for developing DVS-aware programs
    - ⇨Automatic Voltage Scaler Tool

Basic Idea : Inter-task DVS

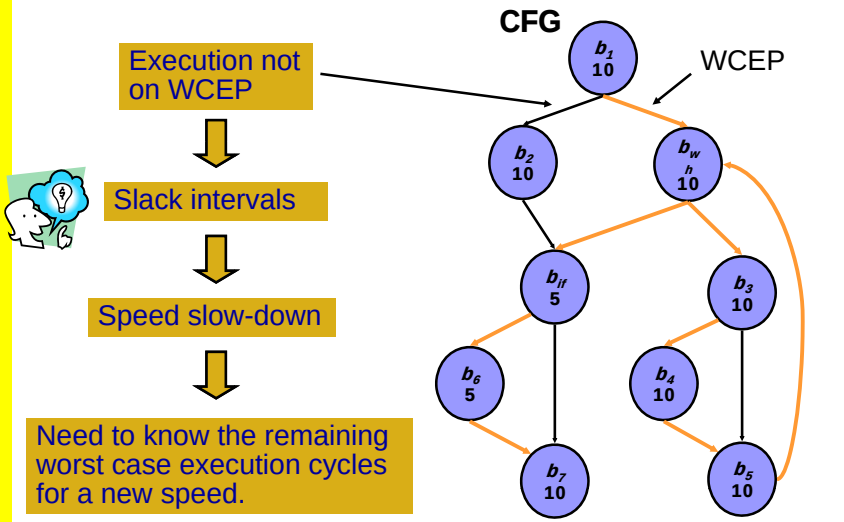


Basic Idea : Optimal DVS



But we cannot know the execution path in advance !!

Basic Idea : Intra-task DVS

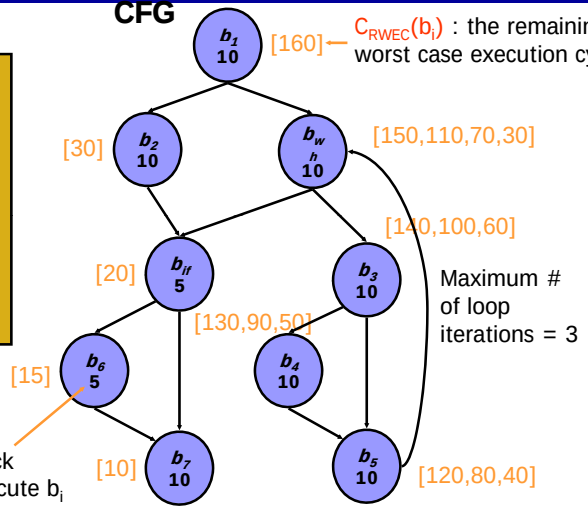


Remaining Execution Cycle Computation

program

```
S1;
if (cond1) S2;
else
    while (cond2) {
        S3;
        if (cond3) S4;
        S5;
    }
if (cond4) S6;
S7;
```

CFG



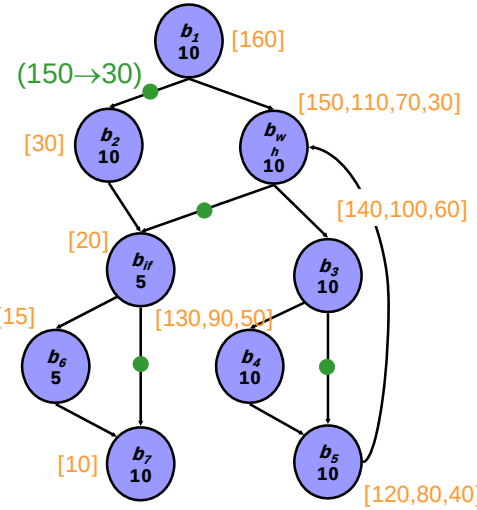
$C_{RWEC}(b_i)$: the remaining worst case execution cycles

Maximum # of loop iterations = 3

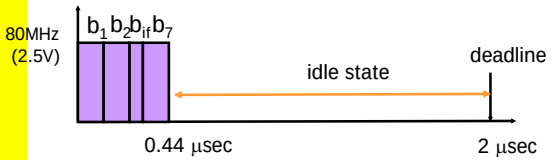
$C_{EC}(b_i)$: the # of clock cycles needed to execute b_i

Speed Assignment Algorithm

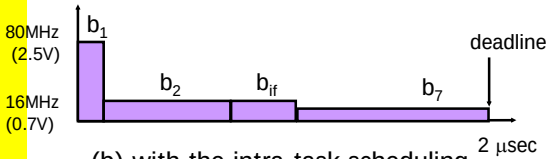
- Branching edges are candidate for speed changes :
  - Branches : The execution control follows the shorter path at if-then-else node. => B-type VSEs
  - Loops : The execution control exits a loop after it iterates by the smaller number of times than the maximum iteration number. => L-type VSEs



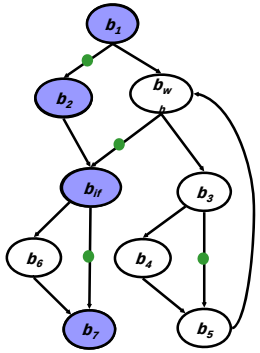
The Effect of Intra-Task Scheduling



(a) without the intra-task scheduling



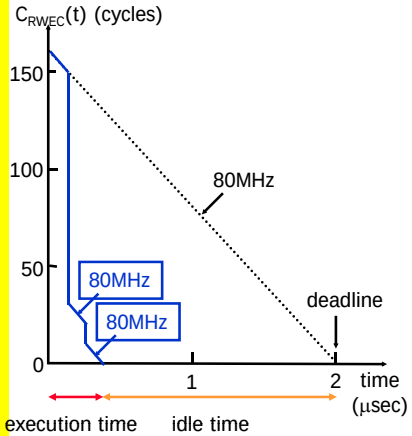
(b) with the intra-task scheduling



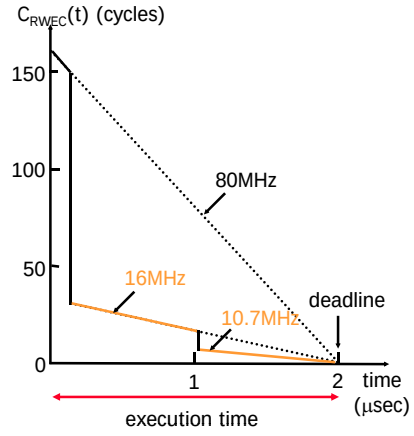
$$\frac{\text{Energy Consumption of (b)}}{\text{Energy Consumption of (a)}} = 0.34$$

The Change of C_{RWEC}

For execution path : (b_1, b_2, b_{if}, b_7)



(a) No intra-task scheduling

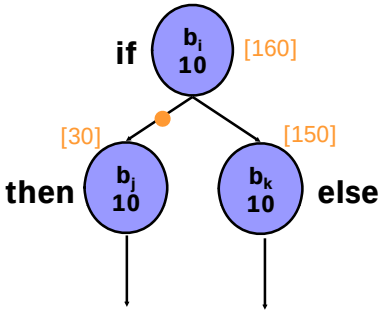


(b) Intra-task scheduling

B-type VSE

- C_{RWEC}
  - 150 → 30
- Speed
  - $S(b_j) \leftarrow S(b_i) \times 1/5$

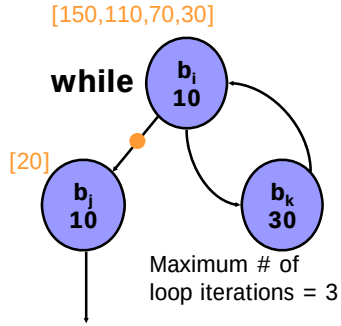
Speed update ratio



L-type VSE

- When the actual loop iterations measured at run time is 2
- C_{RWEC}
  - 60 → 20
- Speed
  - $S(b_j) \leftarrow S(b_i) \times 1/3$

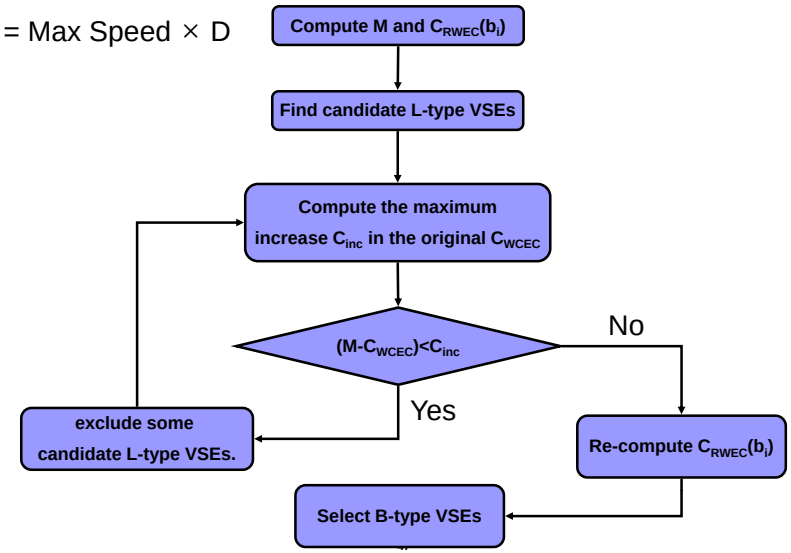
Speed update ratio



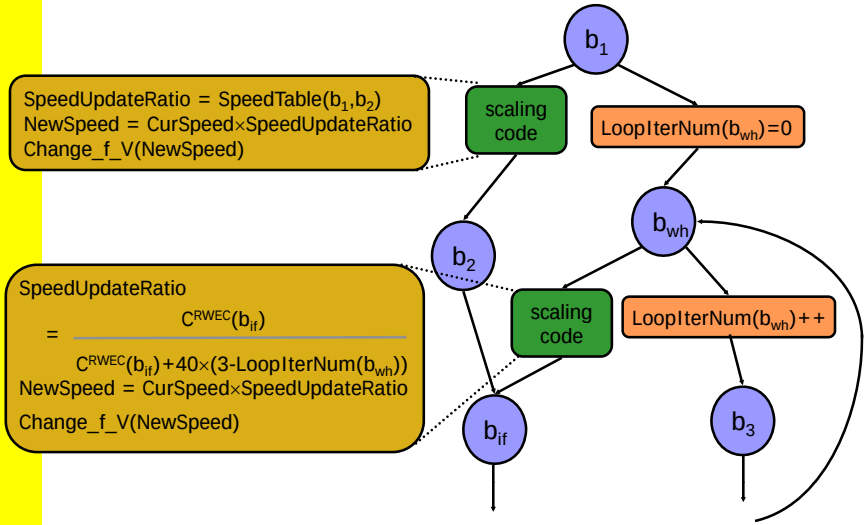
☞The transition overhead is considered to determine a new speed.

VSE Selection

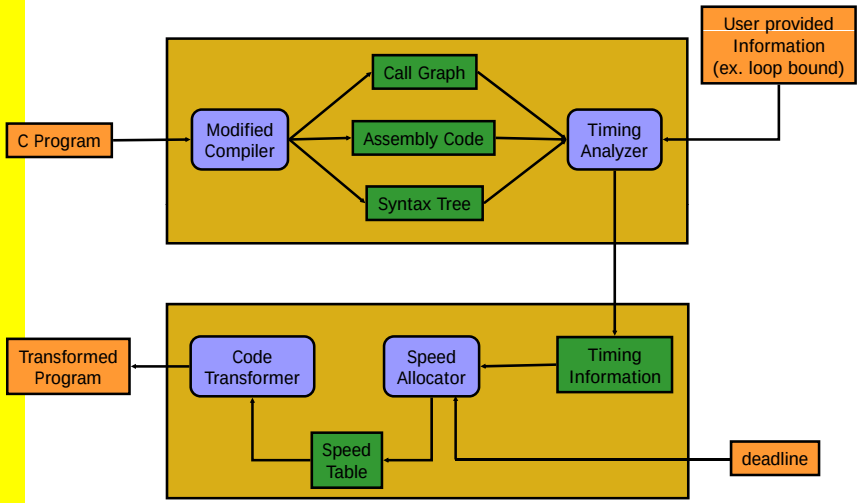
M = Max Speed × D



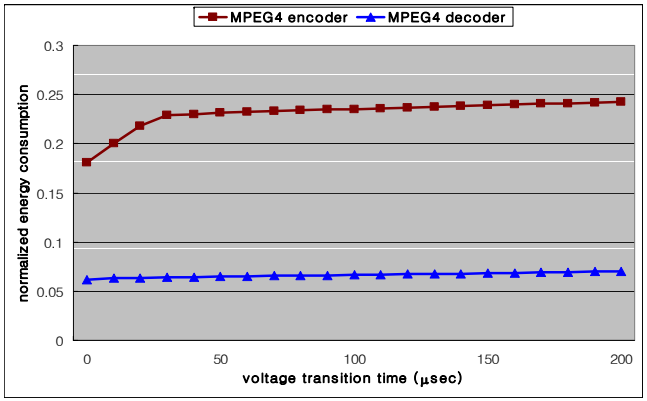
Code Generation for VSEs



Automatic Voltage Scaler

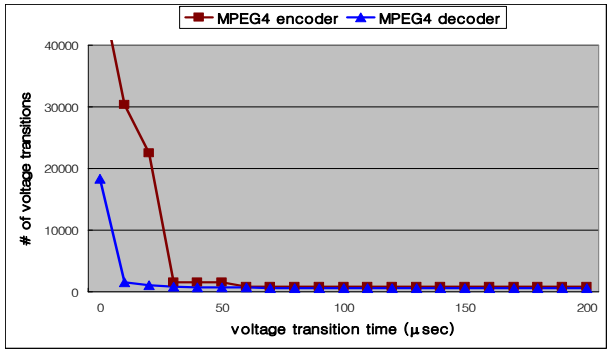


Simulation Results



- Less than 25% and 7% of the original program
- There is a large difference between WCET and ACET of the MPEG-4 decoder

Simulation Results

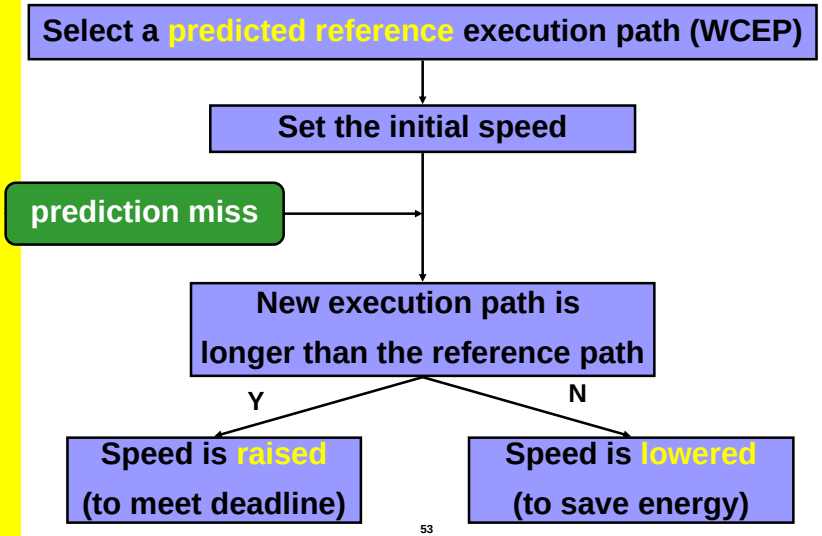


- How many times voltage scaling code were executed
- When C_{VTO} < 30μsec in MPEG-4 encoder, the number of voltage transitions decreases sharply, and energy consumption does not increase rapidly.
- How many copies of voltage scaling code ?
  - 20 VSEs are inserted when C_{VTO} > 50 μsec.

A Profile-Based Intra-Task Voltage Scheduling

- IntraVS algorithm based on average-case execution information
- Average-case execution paths (ACEPs) are the most frequently executed paths
- More effective than the original intraVS algorithm
- The timing constraints of a hard real-time program is still satisfied, even if the ACEPs are used for voltage scaling decisions.

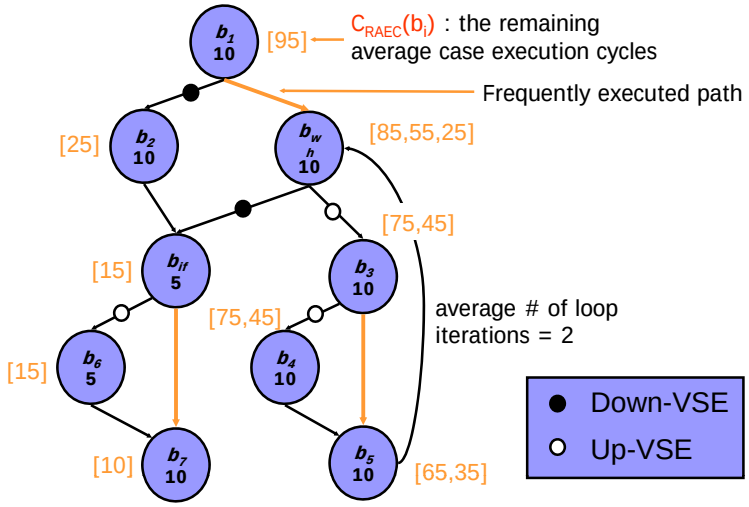
General IntraVS Algorithm



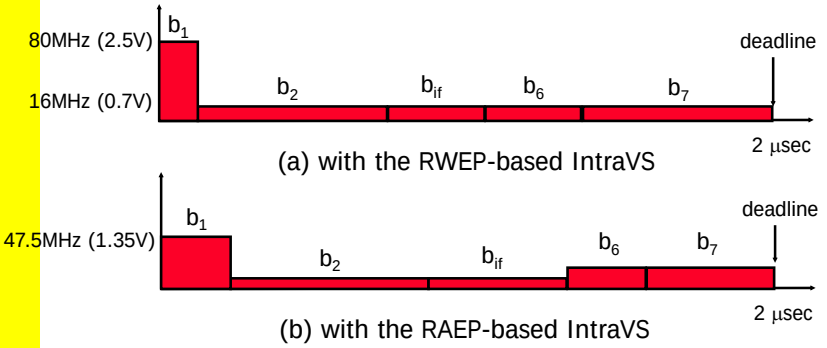
RAEP-based IntraVS

- Motivations
  - To make the common case more energy-efficient
  - If we use one of hot paths as a reference path for intraVS, the speed change graph for the hot paths will be a near flat curve with little changes in clock speed.
  - Even for the paths that are not the hot paths, they are more energy-efficient because they can start with a lower clock speed that RWEP-based IntraVS.
  - RAEP is the best representative of the hot paths.

RAEP-based IntraVS

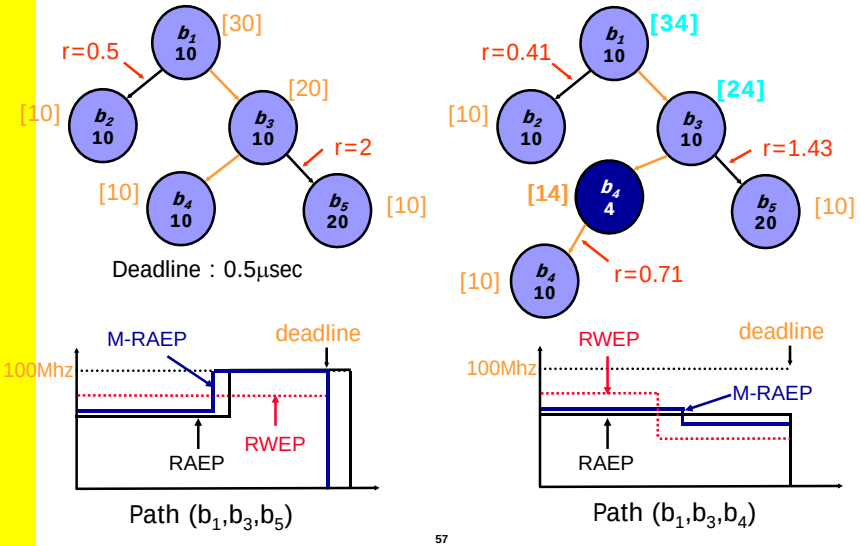


RAEP-based IntraVS

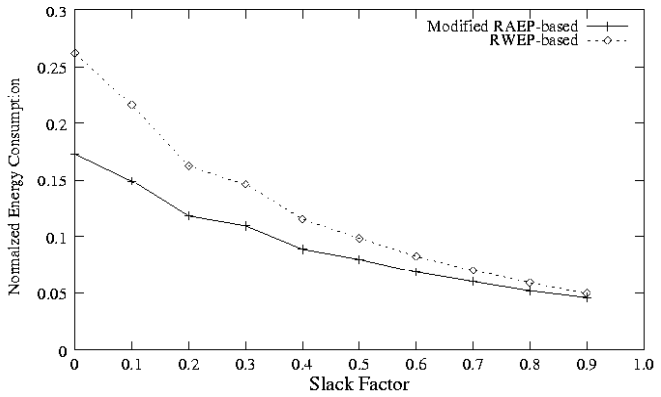


- There are Up-VSEs as well as Down-VSEs at the RAEP-based IntraVS.
- The RAEP-based IntraVS achieves 55% more energy reduction.
- But, the deadline can be missed.

Reference Path Modification



Experimental Results

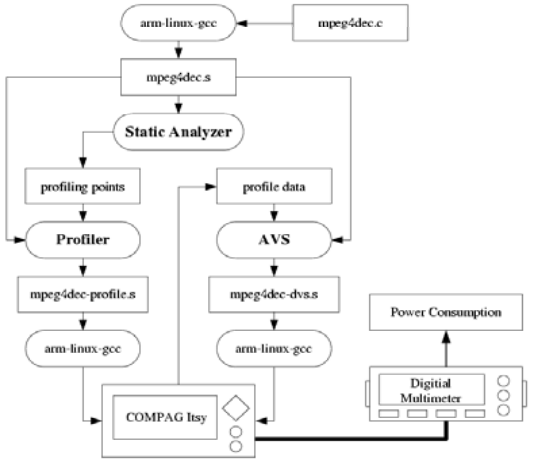


Slack Factor = (deadline-WCET)/deadline

Conclusion

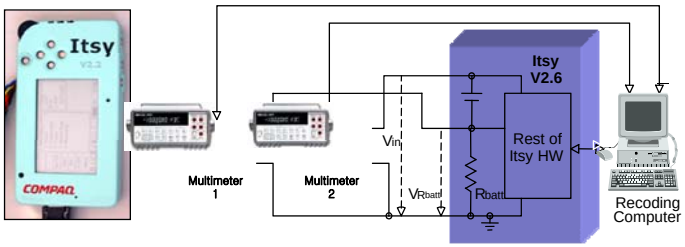
- Presented a novel intra-task DVS algorithm using static timing analysis on RWECs
- Provides a framework for automatic DVS-aware low-power program generation
- The RAEP-based IntraVS algorithm exploits the fact that the average-case execution paths are more likely to be followed at run time than the WCEP.
- Demonstrated the effectiveness of the approach using MPEG-4 encoder/decoder programs

Experiments on Itsy



Experimental Environment

- Itsy Pocket Computer V2.6
  - CPU : Intel StrongARM SA1110
  - Frequency scaling: 11 levels (59.0 MHz ~ 206.4 MHz)
  - Voltage scaling: 30 levels (1.00 V ~ 2.00 V)
  - Default setting: 1.55 V/206.4 MHz
- Linux operating system (ver. 2.0.30)



Experimental Results

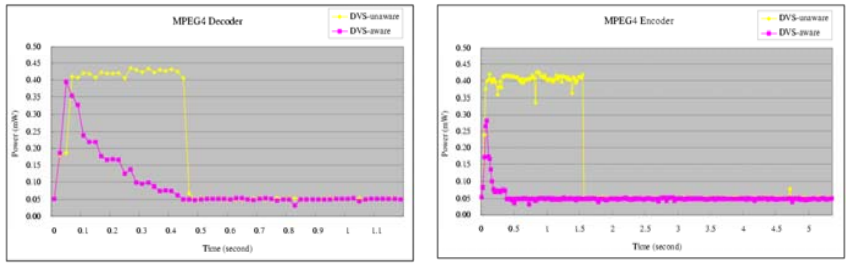
DVS EXPERIMENTS ON ITSY.

Factors		MPEG-4 Decoder		MPEG-4 encoder	
		DVS-aware	normal	DVS-aware	normal
Energy (mJ)		0.11	0.22(0.18)	0.28	0.81(0.62)
Normalized Energy		0.51(0.62)	1	0.35(0.46)	1
Execution Time (sec)		1.18	0.46	5.34	1.54
WCET (sec)		3.2		21.0	
Selected VSEs	B-VSE	2		1	
	L-VSE	1		2	
Management Code	Function	2		3	
	Loop	5		8	

MANAGEMENT CODE OVERHEAD.

Management Code	Code number in Fig. 7	Number of assembly instructions
Loop Enter	code 1 and 2	30
Loop Header	code 3	16
Loop Exit	code 4	16
Function Enter	code 5	14
Function Return	code 6	11
B-type VSE	code B	≈ 200
L-type VSE	code L	≈ 200

Experimental Results



Roadmap

- DVS in Non Real-Time Systems
- DVS in Real-Time Systems
  - Compiler-level DVS: Intra-task DVS
  - OS-level DVS: Inter-task DVS
  - Application-level DVS
    - MPEG-decoder implementation
  - Algorithm-level DVS
    - Low-power convolution

Inter-task DVS Overview

- Inter-task DVS algorithms
  - Determine the supply voltage and clock speed on *task-by-task* basis
- Inter-task DVS
  - Is similar to that of imprecise computation in conventional real-time systems
    - Imprecise computation
      - Use the slack time to increase the values of results
      - While guaranteeing the feasible schedule of tasks
    - Dynamic voltage scaling
      - Use the slack time to lower the voltage/clock speed
      - While guaranteeing the feasible schedule of tasks

Preliminaries

- Computing model
  - Non-real-time
    - tasks have no timing constraints
  - Real-Time
    - Timing constraints
    - Periodic and(or) aperiodic tasks
    - Scheduling policy : EDF, RM, and etc.
- Different DVS algorithms are necessary depending on different computing models.

Inter-Task DVS

- "Run-Calculate-Assign-Run" strategy for the supply voltage determination
  - Running the current task
  - Calculating the maximum allowable execution time for the next task
    - WCET plus slack time
  - Assigning the supply voltage for the next task
  - Running the next task

Generic Inter-DVS Algorithms

- Consist of two parts
  - Slack estimation
    - Identify as much slack times as possible
    - Slack times
      - Static slack times
        - Extra times available for the next task that can be identified statically
      - Dynamic slack times
        - Ones caused from run-time variations of the task executions
  - Slack distribution
    - Adjust the speed so that the resultant speed schedule is as flat as possible

Static and Dynamic Approaches

- Off-line (Static) voltage scheduling approaches
  - The execution times are assumed to be known a priori
  - There are several optimal solutions for EDF, RM, and etc.
- On-line (Dynamic) voltage scheduling approaches
  - The execution times are assumed to be not known
  - There cannot be an optimal solution

Slack Estimation Methods

	Voltage Scaling Methods	Scaling Decision
IntraDVS	(1) Path-based method	Off-line
	(2) Stochastic method	
InterDVS	(3) Maximum constant speed	
	(4) Stretching to NTA	On-line
	(5) Priority-based slack-stealing	
	(6) Utilization updating	

Maximum Constant Speed

- The lowest possible clock speed that guarantees the feasible schedule of a task set
  - EDF scheduling
    - If the worst case processor utilization U of a given task set is lower than 1.0 under the maximum speed $f_{\max}$, the task set can be scheduled with a new maximum speed

$$U = \sum_{i=1}^{n} \frac{c_i}{p_i} \qquad f_{MSC} = U \cdot f_{\max}$$

Maximum constant speed

  - Rate Monotonic scheduling

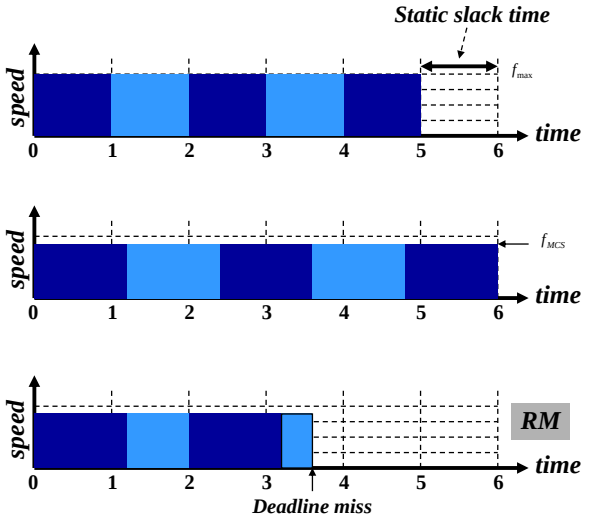
$$L_i(t) = \frac{\sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j}{t} \qquad f_{MCS} = f_{\max} \times \max_i \{L_i(t) \mid 1 < i \le n, 0 < t < d_i\}$$

Example

	Period Length	WCET
Task 1	2	1
Task 2	3	1

$$U = \frac{1}{2} + \frac{1}{3} = 0.833$$

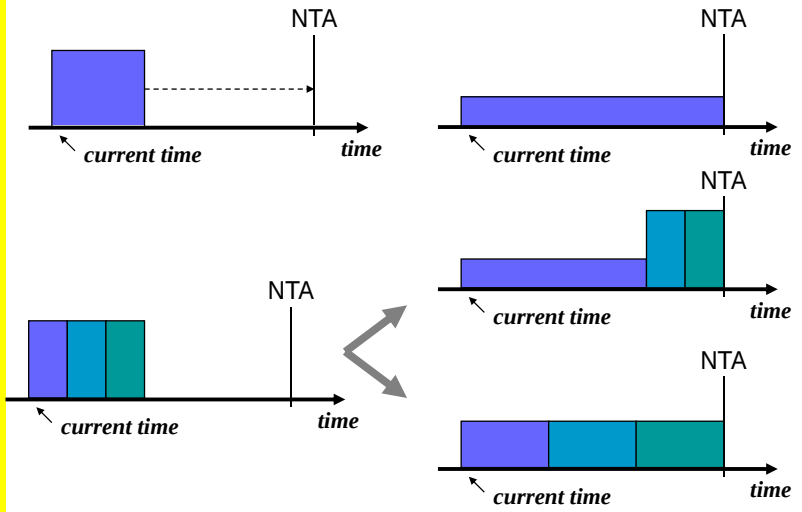
$$f_{MCS} = 0.833 \cdot f_{\max}$$



Stretching to NTA

- Even though a given task set is scheduled with the maximum constant speed, since the actual execution times of tasks are usually much less than their WCETs, the tasks usually have dynamic slack times
- For the task τ which is scheduled at time t
  - If the next task is later than $t + WCET(\tau)$
  - We can slow down the execution of τ so that its execution completes exactly at this next task arrival time (NTA)

Example

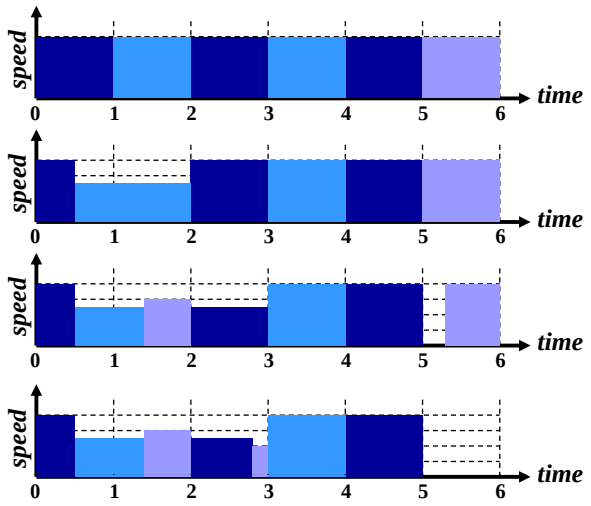


Priority-Based Slack Stealing

- Exploits basic properties of priority-driven scheduling such as EDF and RM
  - When a higher-priority task completes its execution earlier than its WCET, the following lower-priority tasks can use the slack time from the completed higher-priority task
- Advantage
  - Most task instances in a hyper-period may have chances to utilize dynamic slack times
    - Because most task executions complete earlier than WCETs
    - Therefore, many task instances can be scheduled with lowered voltages and speeds

Example

	Period Length	WCET
Task 1	2	1
Task 2	3	1
Task 3	6	1



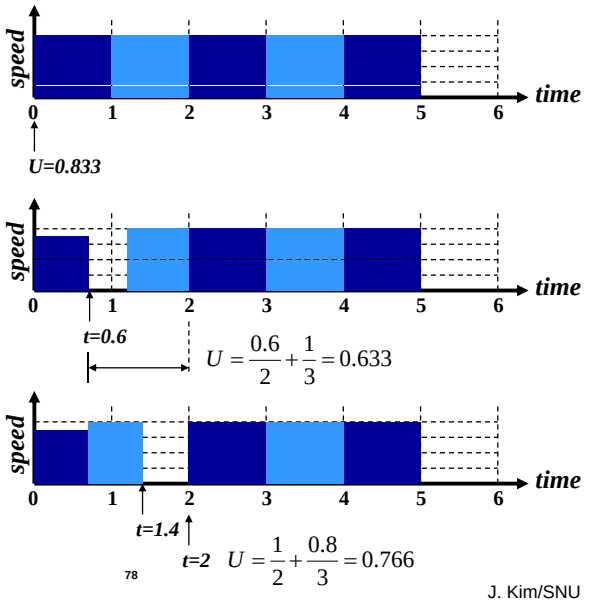
Utilization Updating

- The actual processor utilization during run time is usually lower than the worst case processor utilization
- This method is to estimate the required processor performance at the current scheduling point
  - By recalculating the expected worst case processor utilization
    - Using the actual execution times of completed task instances

Example

	Period Length	WCET
Task 1	2	1
Task 2	3	1

$$U = \frac{1}{2} + \frac{1}{3} = 0.833$$



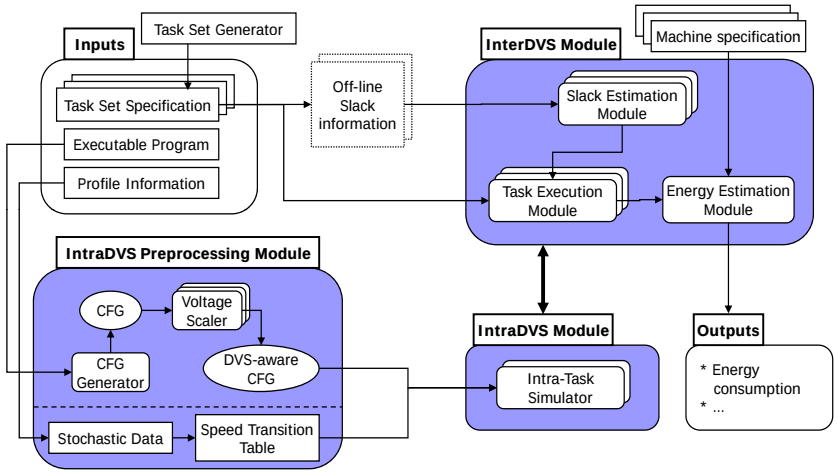
Slack Distribution Method

- Greedy approach
  - All the slack times are given to the next activated task
  - Most inter-task DVS algorithms have adopted it
  - Clearly, this approach is not an optimal solution, but is widely used because of its simplicity

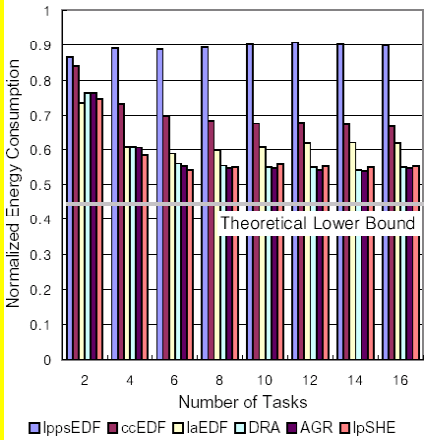
Existing Inter-Task DVS Algorithms

Category	Scheduling Policy	DVS Policy	Used Method
Inter-task DVS	EDF	lppsEDF	(3)+(4)
		ccEDF	(6)
		laEDF	(6)*
		DRA	(3)+(4)+(5)
		AGR	(4)*+(5)
		lpSEH	(3)+(4)+(5)*
	RM	lppsRM	(3)+(4)
		ccRM	(4)*
Intra-task DVS	Path-based method	intraShin	(1)
	Stochastic method	intraGruian	(2)

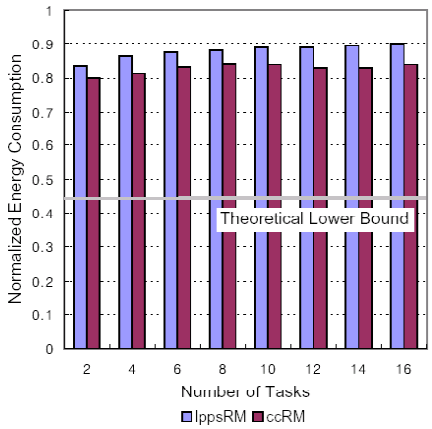
SimDVS: A Unified DVS Evaluation Environment



Experimental Results

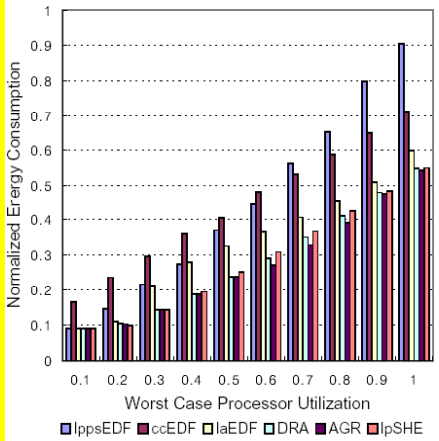


(a) DVS for EDF

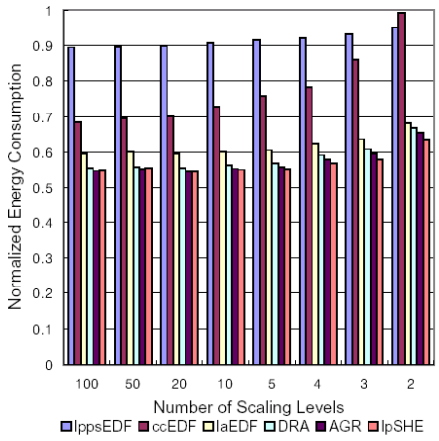


(b) DVS for RM

Experimental Results

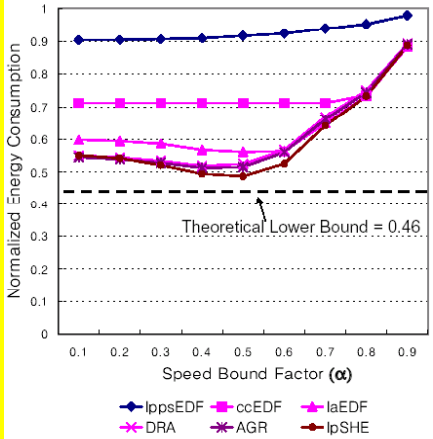


(a) WCPU

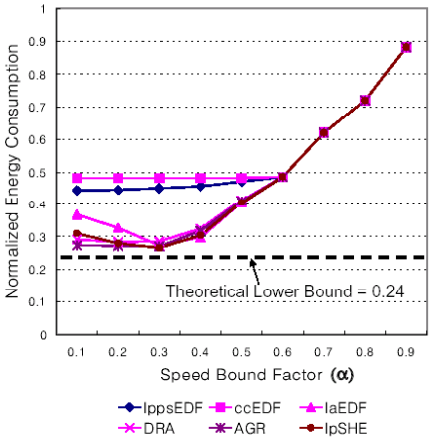


(b) Machine Specification

Experimental Results

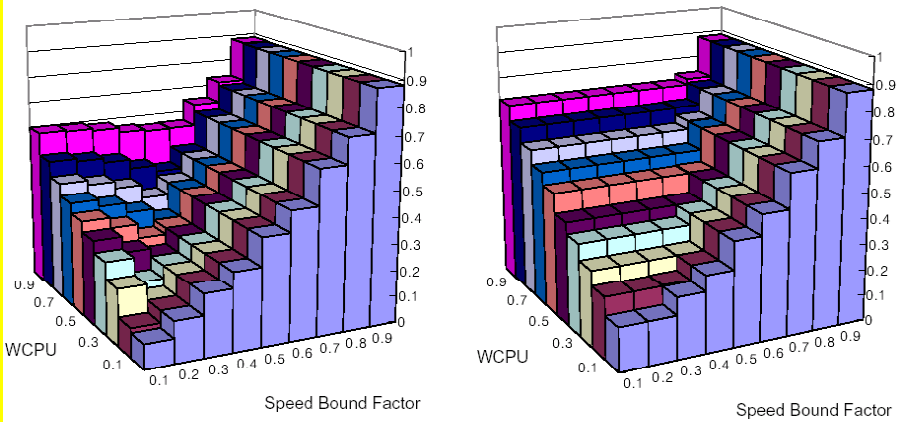


(a) WCPU=1.0 & ACPU=0.55



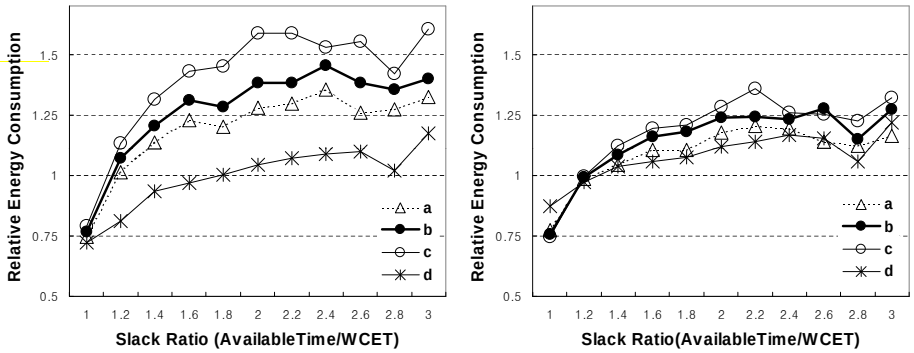
(b) WCPU=0.6 & ACPU=0.33

Experimental Results



(a) laEDF (b) ccEDF

Experimental Results (IntraDVS)

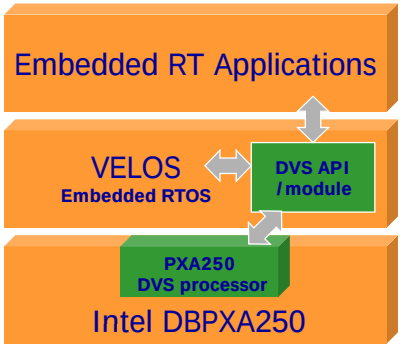


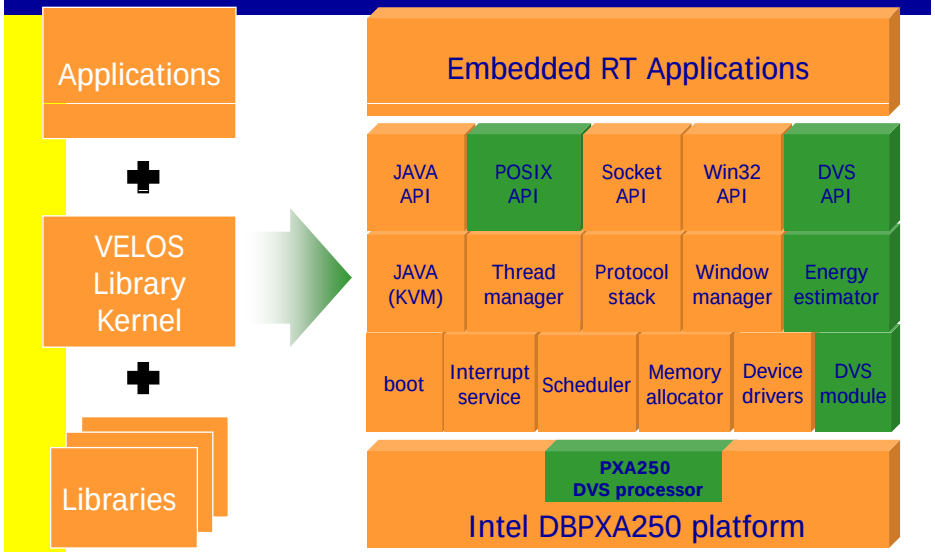
MPEG4 Decoder ***MPEG4 Encoder***

Performance Evaluation DVS Algorithms for Hard Real-Time Systems Using DEW

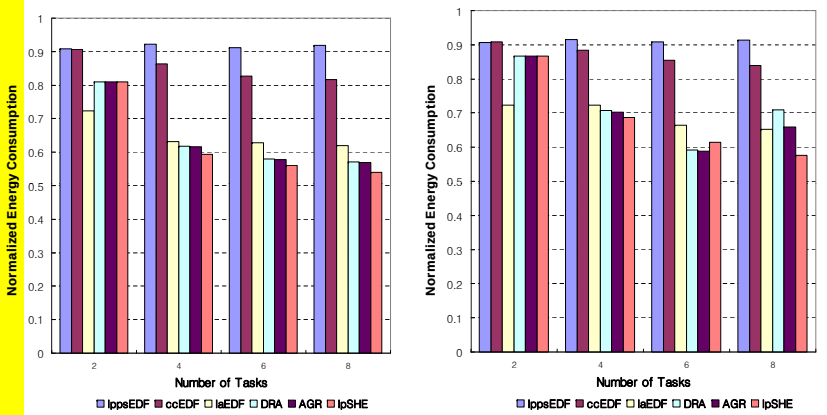
DEW – DVS Evaluation Workbench

- XScale-based DVS evaluation environment
  - Pros
    - Allows to monitor real system behaviors under DVS
  - Cons
    - Slower than software simulation
      - Because DEW runs actual applications
    - Less flexible for experimental studies
      - Because DEW represents a single machine specification





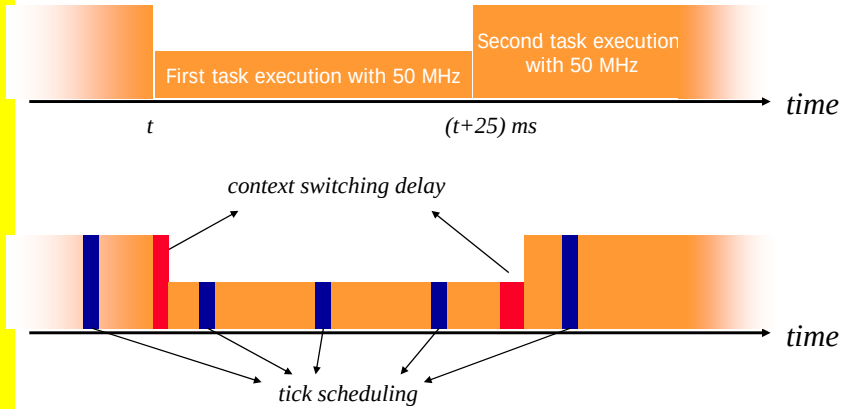
Evaluation Results Using SimDVS and DEW



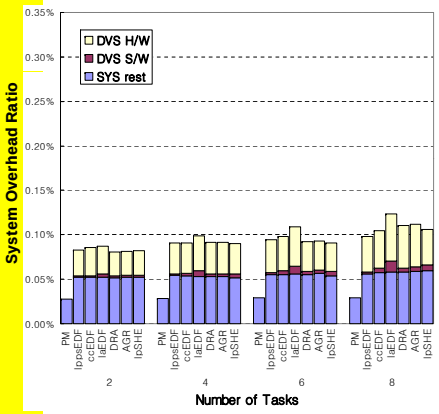
Sources of Differences

- Impacts of
  - System overhead
    - Basic : context switching overhead and tick scheduler overhead
    - DVS : slack computation and clock/voltage scaling
  - System timing resolution
    - Simulator : continuous time model
    - Real system : discrete time model
  - Memory behavior
    - Changes in cache and memory access behavior
    - Data/Instruction fetch latency

Example of System Overheads on a Real Platform



System Overhead



- DVS H/W

  The ratio of time delay caused by the clock/voltage scaling hardware

- DVS S/W

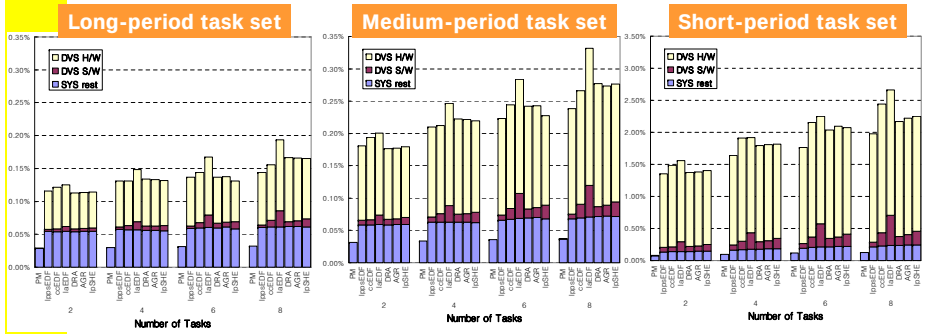
  The ratio of time delay caused by the slack computation in a DVS algorithm

- SYS rest

  The ratio of the rest of the system overhead such as context switching and timer service

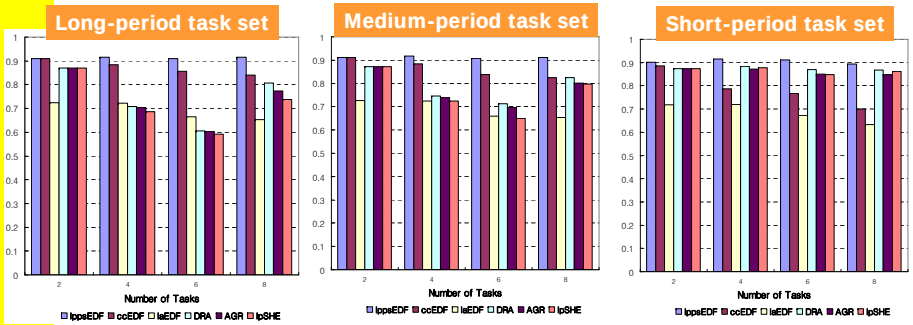
System Overhead Variations

- The system overhead increases very quickly as the task execution frequency increases
  - In particular, DVS parts increase quickly

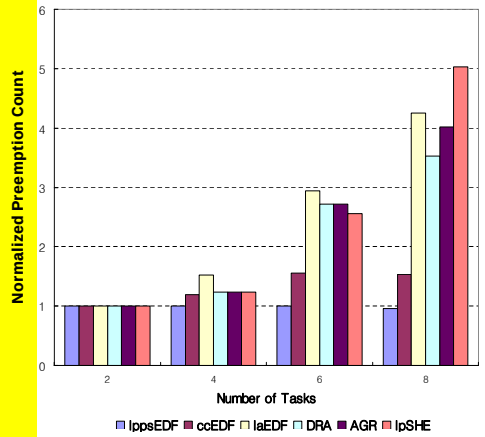


Energy Efficiency Variations

- In DRA, AGR, and lpSHE, the increased system overhead (due to the increased execution frequency) significantly affect the energy efficiency

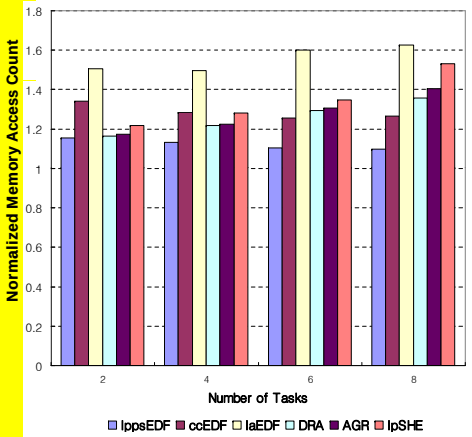


Changes in Memory System Behaviors (1)



- Under a DVS-enabled RTOS, Task's execution time increases due to the lowered clock speed
  - Desirable for reducing energy consumption
  - But, it can introduce negative side effects as well
    - An increase in the number of task preemptions which increases the number of memory accesses
- In aggressive algorithms, the number of preemptions increases more rapidly than the others

Changes in Memory System Behaviors (2)



- PXA250
  - Performance Monitoring Unit
  - 32-way set-associative cache of Inst/Data cache
- Each application
  - 16-KB program code
- The increases in memory accesses can be attributed to two sources
  - The increase in the number of preemptions
  - The increase in memory accesses from the algorithm itself

References

- Transmeta Corporation. Crusoe Processor. http://www.transmeta.com, June 2000.
- AMD Corporation. PowerNow! Technology. http://www.amd.com, December 2000.
- Intel Corporation. Intel XScale Technology. http://developer.intel.com/ design/ intelxscale/, November 2001.
- I. Hong, G. Qu, M. Potkonjak, and M. B. Srivastava. Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processor. In Proceedings of the IEEE Real-Time Systems Symposium, pages 178-187, December 1998.
- Y. Shin, K. Choi, and T. Sakurai. Power Optimization of Real-Time Embedded Systems on Variable Speed Processors. In Proceedings of the International Conference on Computer-Aided Design, pages 365-368, November 2000.
- H. Aydin, R. Melhem, D. Mosse, and P. M. Alvarez. Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems. In Proceedings of IEEE Real-Time Systems Symposium, December 2001.

References

- P. Pillai and K. G. Shin. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In Proceedings of 18th ACM Symposium on Operating Systems Principles (SOSP'01), October 2001.
- D. Shin, J. Kim, and S. Lee. Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications. IEEE Design and Test of Computers, 18(2):20-30, March 2001.
- F. Gruian. Hard Real-Time Scheduling Using Stochastic Data and DVS Processors. In Proceedings of the International Symposium on Low Power Electronics and Design, pages 46-51, August 2001.
- W. Kim, J. Kim, and S. L. Min. A Dynamic Voltage Scaling Algorithm for Dynamic-Priority Hard Real-Time Systems Using Slack Time Analysis. To appear in Proceedings of Design, Automation and Test in Europe (DATE'02), March 2002.

References

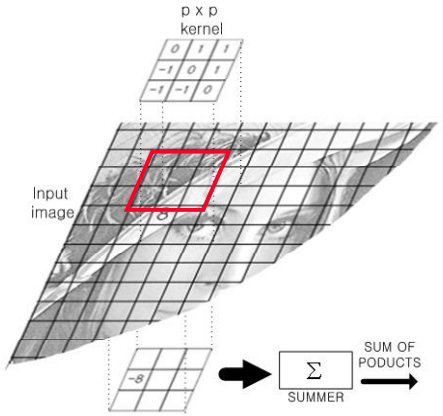
- D. Grunwald, P. Levis, and K. I. Farkas. Policies for Dynamic Clock Scheduling. In Proceedings of the 4th Symposium on Operating Systems Design and Implementation, pages 73-86, October 2000.
- S. Lee and T. Sakurai. Run-time Voltage Hopping for Low-power Real-Time Systems. In Proceedings of the 37th Design Automation Conference, pages 806-809, June 2000.
- T. Burd and R. Brodersen. Design Issues for Dynamic voltage scaling . In Proceedings of the International Symposium on Low Power Electronics and Design, pages 9-14, July 2000.
- D. Burger and T. M. Austin. The SimpleScalar Tool Set, version 2.0. Technical Report 1342, University of Wisconsin-Madison, CS Department, June 1997.
- F. Yao, A. Demers, and A. Shenker. A Scheduling Model for Reduced CPU Energy. In Proceedings of the IEEE Foundations of Computer Science, pages 374-382, 1995.

DVS-Aware Algorithm Development

Image Convolution

- One of the fundamental operations of image processing.
- DVS Unfriendly!!

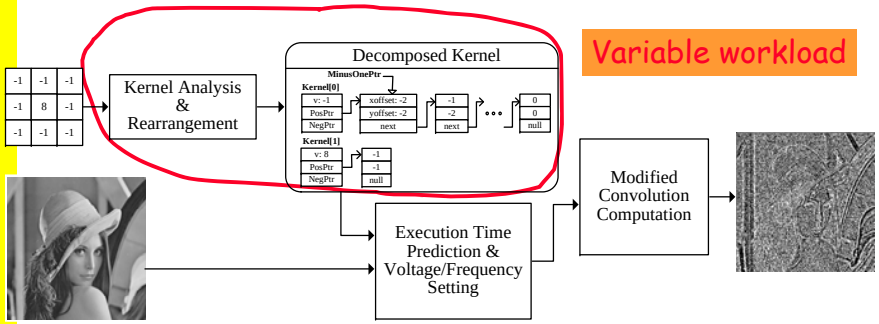
Direct Implementation: Constant-Workload Algorithm



- p^2 multiplications p^2 additions for each convolved element.

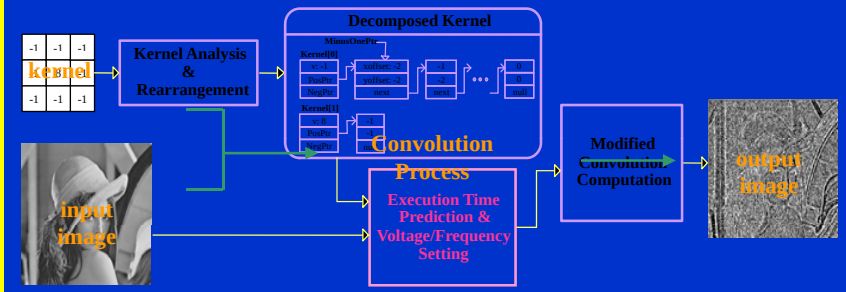
No Workload Variations!

Low-Power Implementation



Variable Workload Algorithm
Based on Kernel Characteristics

Low-Power Implementation



Overall processing step

Kernel Analysis and Rearrangement

- *Property 1*. For most kernels, the number of distinct kernel elements is small.
- *Property 2*. 0, 1 and –1 are used frequently.
- *Property 3*. Many kernel elements have the same absolute values.

0	1	1
-1	0	1
-1	-1	0

-1	-2	-1
0	0	0
1	2	1

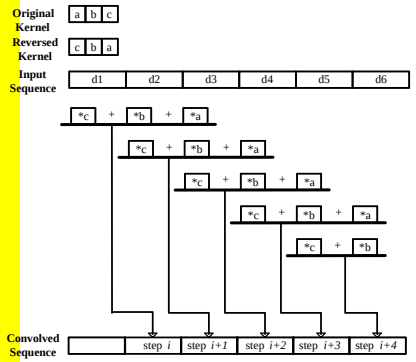
Modified Convolution Algorithm: SDMK

- For 1 or –1, no multiplication.
- For 0, no addition & no multiplication.
- For the same absolute values, a single multiplication.

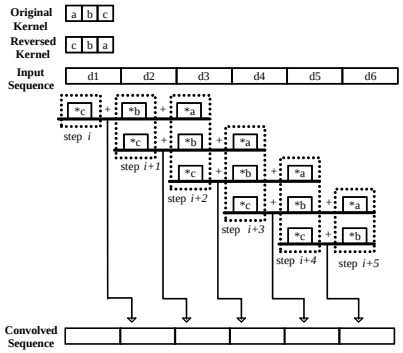
0	1	1
–1	0	1
–1	–1	0

	Direct	SDMK
Number of Operations/pixel	9 additions & 9 multiplications	6 additions & 0 multiplications

Direct vs. SDMK



Direct implementation



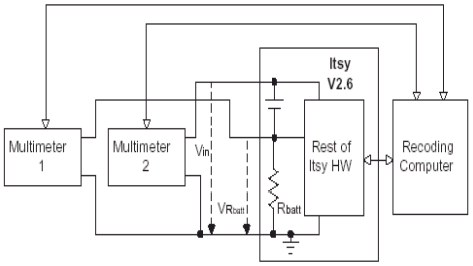
SDMK implementation

Exec Time Prediction & Speed Setting

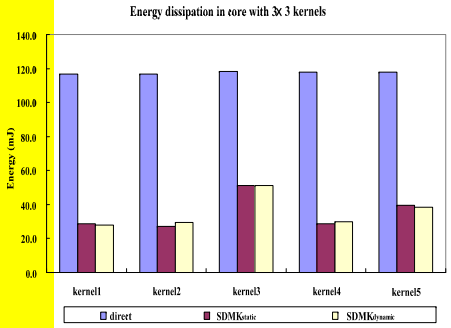
- By a static method
  - Based on the number of required arithmetic operations
- By a dynamic-method
  - Based on actual measurements of execution times
  - In the direct algorithm, by pre-constructed speed table

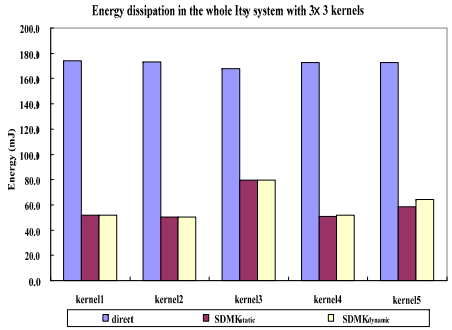
Experimental Environments

- Itsy Pocket Computer V2.6
  - CPU : Intel StrongARM 1110
  - Frequency scaling: 11 levels (59.0 MHz ~ 226.4 MHz)
  - Voltage scaling: 30 levels (1.00 V ~ 2.00 V)
- Linux operating system (ver. 2.0.30)



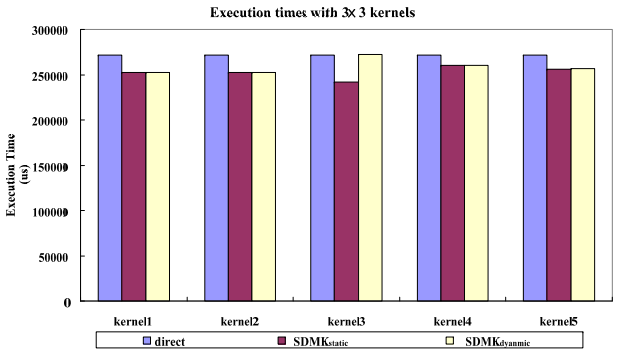
Results (Energy Dissipation)





- Average 67.6% energy saving in the core processor, and 62.8% in the whole Itsy system.

Results (Execution Time)



- There is no performance degradation over the direct approach.

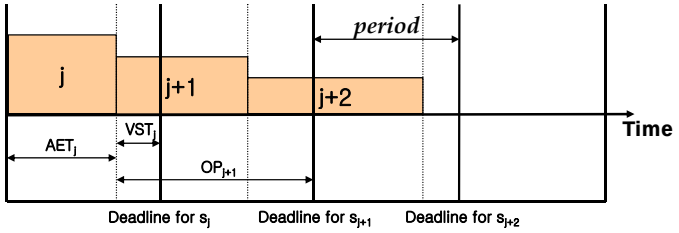
Conclusions

- Presented a low-power implementation of image convolution algorithm for variable voltage processors.
- The energy efficiency of the proposed implementation comes from:
  - Smaller N_{cycle}
  - Lower V_{dd}
  - Fewer memory references
    - i.e., less energy consumed in non-CPU components

$$E \propto N_{cycle} \cdot V_{DD}^2$$

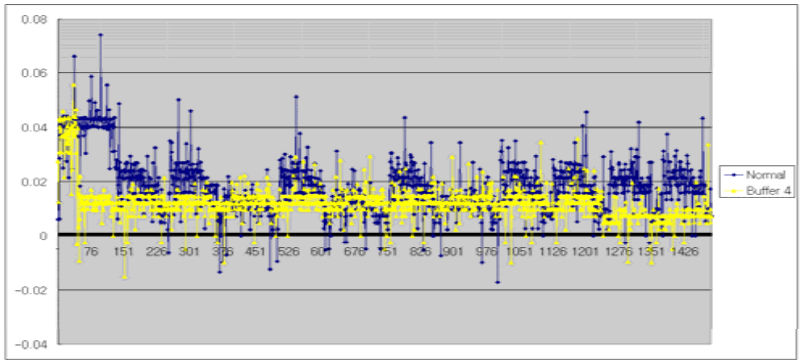
MPEG Decoder Implementation

- 버퍼를 활용한 DVS
  - Workload-variation 슬랙 시간을 활용하는 것이 가능
  - 추가적인 메모리 자원을 소모
    - **one-buffer-size** = *image_width* * *image_height* * *byte_per_pixel*



Measurement Results

- 버퍼 방식 DVS 기법 사용 (WCET사용)
  - Bitrate of sample video : 163Kbps



  - Energy saving : up to 53%

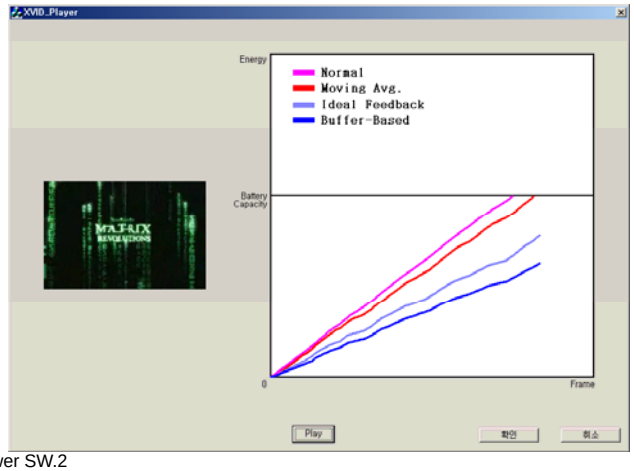
Demo (1)

- DAQ 보드로 프로파일링된 데이터를 실제 동영상과 같이 표현
  - Ideal DVS
  - DVS Using Moving Avg.
  - Buffer-Based DVS

현재 화면을 디코딩하는 전압

Demo (2)

- 각 DVS policy 들 사이의 총 전력 소모를 비교



Energy-Optimal Off-Line Voltage Scheduling

Off-Line Volt. Sched. Problem

- Voltage schedule (speed schedule) : S(t)
  - the processor speed as a function of time
  - The energy consumption under S(t) is given by
    $E(S) = \int_{interval} P(S(t))\, dt$
    - P is a convex function from speed to power
- Given N jobs $J_1, J_2, \ldots, J_N$ where
  - r_i : the release time of J_i
  - d_i : the deadline of J_i
  - c_i : the workload (# of execution cycles) of J_i
    - assumed to be known a priori
  - p_i : the priority of J_i

  compute a feasible voltage schedule S(t) that minimizes E(S)
- S(t) is feasible iff S(t) gives J_i its workload c_i between r_i and d_i for all $J_1, J_2, \ldots, J_N$

Existing Works for the Problem

- Note that the system model covers
  - Fixed-priority (RM, DM) periodic/aperiodic task set
  - EDF periodic/aperiodic task set
    - $p_i < p_j$ iff $d_i < d_j$
- For EDF job sets (a special case), the problem can be solved in poly. time by Yao's algo.[FOCS'95]
  - solution space = convex , obj. func. = convex
- For general job sets, the problem becomes much difficult
  - main source of difficulty : feasibility condition
  - Quan & Hu [TCAD'03]: exhaustive optimal algo.
  - Yun & Kim [TECS'03]: NP-hardness & FPTAS

References

- Optimal algorithm for EDF job sets
  - F. Yao, A. Demers, S. Shenker, “A Scheduling Model for Reduced CPU Energy”, In Proc. Foundations of Computer Sciences (FOCS’95), 1995
- Heuristic for FP job sets
  - G. Quan and X. Hu, “Energy Efficient Scheduling for Real-Time Systems On Variable Voltage Processor”, In Proc. Design Automation Conference (DAC’01), 2001.
- Exhaustive optimal algorithm for FP job sets
  - G. Quan and X. Hu, “Minimum Energy Fixed Priority Scheduling for Variable Voltage Processors”, IEEE Transactions on Computer Aided Design and Systems, 2003.
- NP-hardness proof & FPTAS for FP job sets
  - H.-S. Yun and J. Kim, “On Energy-Optimal Voltage Scheduling for Fixed-Priority Hard Real-Time Systems”, ACM Transactions on Embedded Computing Systems, 2003.