Dynamic Voltage and Frequency Scaling for Energy-Efficient System Design

*Kihwan Choi and **Massoud Pedram***

University of Southern California
Dept. of EE

April 27th, 2005
NCTU, Taiwan

Outline

- Introduction and review of basic concepts
- Dynamic voltage and frequency scaling
- Workload decomposition
- DVFS targeting total system energy reduction
- Conclusion and research directions

Realities

- Power has emerged as the #1 limiter of design performance beyond the 65nm generation.
- Dynamic and static power dissipation limit achievable performance due to fixed caps on chip or system cooling capacity.
- Power related signal integrity issues (IR drop, L di/dt noise) have become major sources of design re-spins.

Transistors (and silicon) are free.
Power is the only real limiter.
Optimizing for frequency and/or area may achieve neither.

Pat Gelsinger, Senior Vice President & CTO, Intel

Industry View

BusinessWeek online

BW HOME | BW MAGAZINE | TOP NEWS | INVESTING | GLOBAL BIZ | TECHN

OCTOBER 4, 2004 · Editions: N. America | Europe | Asia | Edition Prefe

Customer Service
Register
Subscribe to BW

Get Four
Free Issues

Full Table of Contents
Cover Story
International Cover Story
Up Front
The Great Innovators
Readers Report
Corrections & Clarifications
Books
Technology & You
Economic Viewpoint
Business Outlook

MLB.com™ knows

TECHNOLOGY & YOU

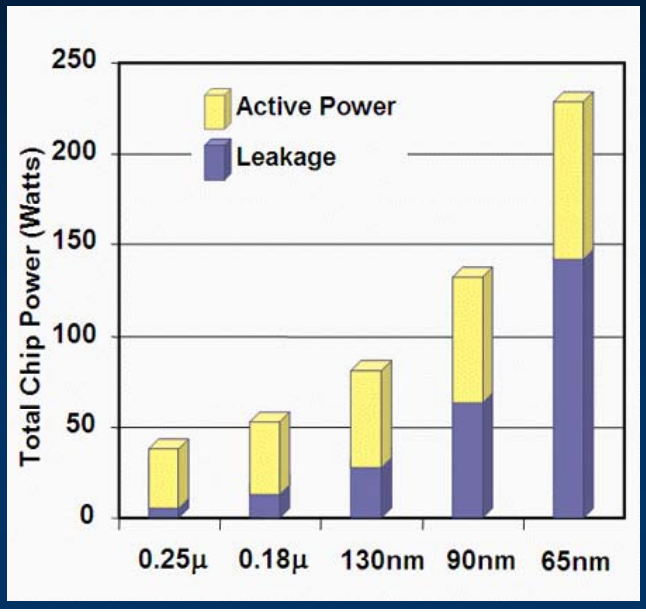
Those Superfast Chips: Too Darn Hot

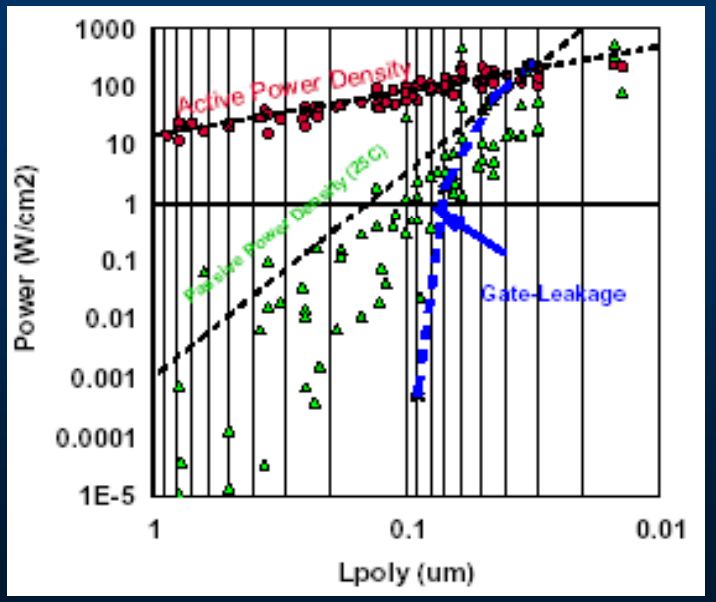
Without cooler new processors, PC makers could hit a speed bump

Intel's (INTC) recent announcement that it plans to produce new "dual-core" processors that amount to two Pentiums on a single chip drew attention mainly from hard-core techies. But it was an admission that the company's strategy for making PCs ever cheaper and faster has hit a wall: The chips are simply getting too hot. Further progress will require new technologies.

CMOS Scaling

- Scaling improves:
  - Transistor Density & Functionality on a chip.
  - Speed and frequency of operation $\Rightarrow$ Higher performance.
- Scaling and power dissipation
  - Active power $\uparrow$ - CV_{DD}^2f
    - Scale V_{DD}
    - Scale V_{th} to recover speed $\Rightarrow$ $I_{leak}\uparrow$
  - Standby (or leakage) power $\uparrow$ $V_{DD}I_{leak}$
- Leakage power is catching up with the active power in UDSM CMOS circuits.



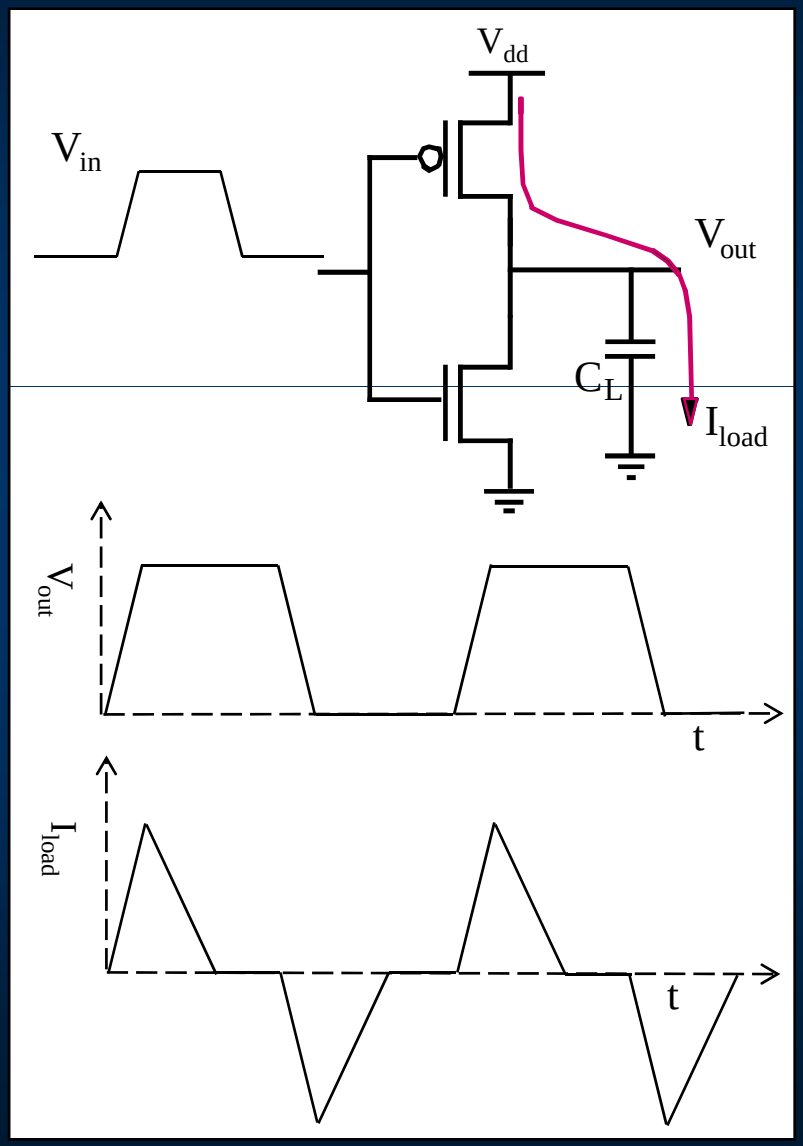


Capacitive Power Dissipation

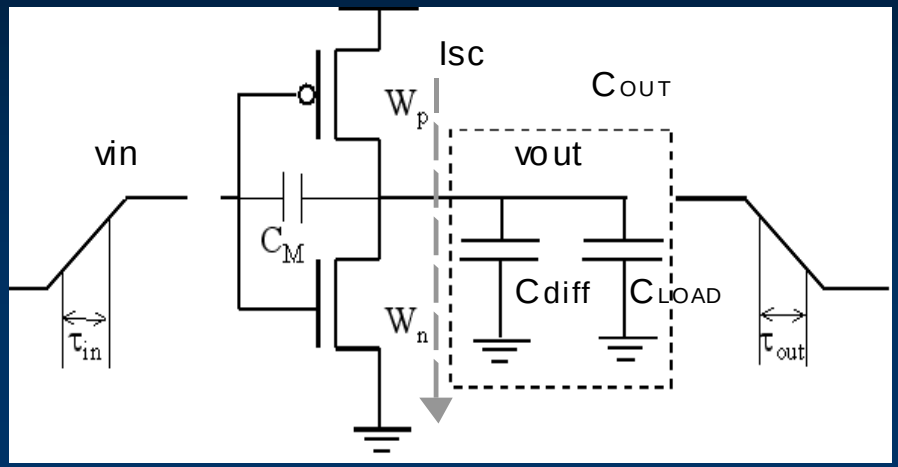
- The capacitive component of power dissipation is a function of:
  - Frequency
  - Capacitive loading
  - Voltage swing
  - Activity factor.

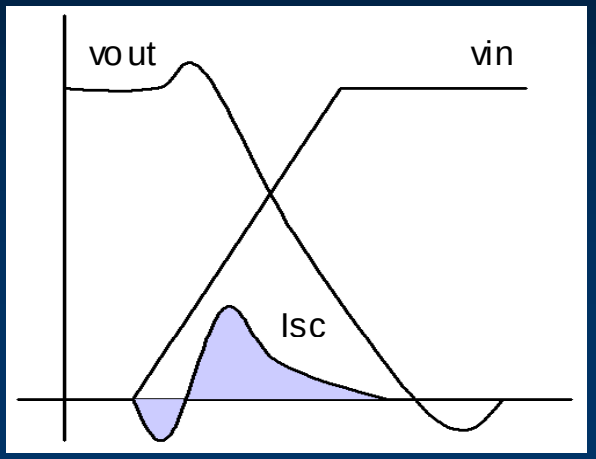
$$\text{E} = \text{Energy/transition} = \frac{1}{2} \cdot C_L \cdot V_{dd}^2$$

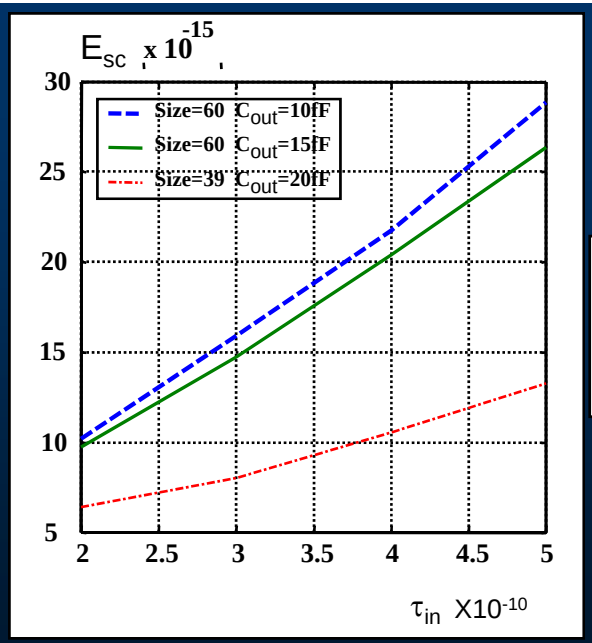
$$\text{P} = \text{Power} = E \cdot f \cdot a = \frac{1}{2} \cdot C_L \cdot V_{dd}^2 \cdot f \cdot a$$



Short Circuit Power Dissipation

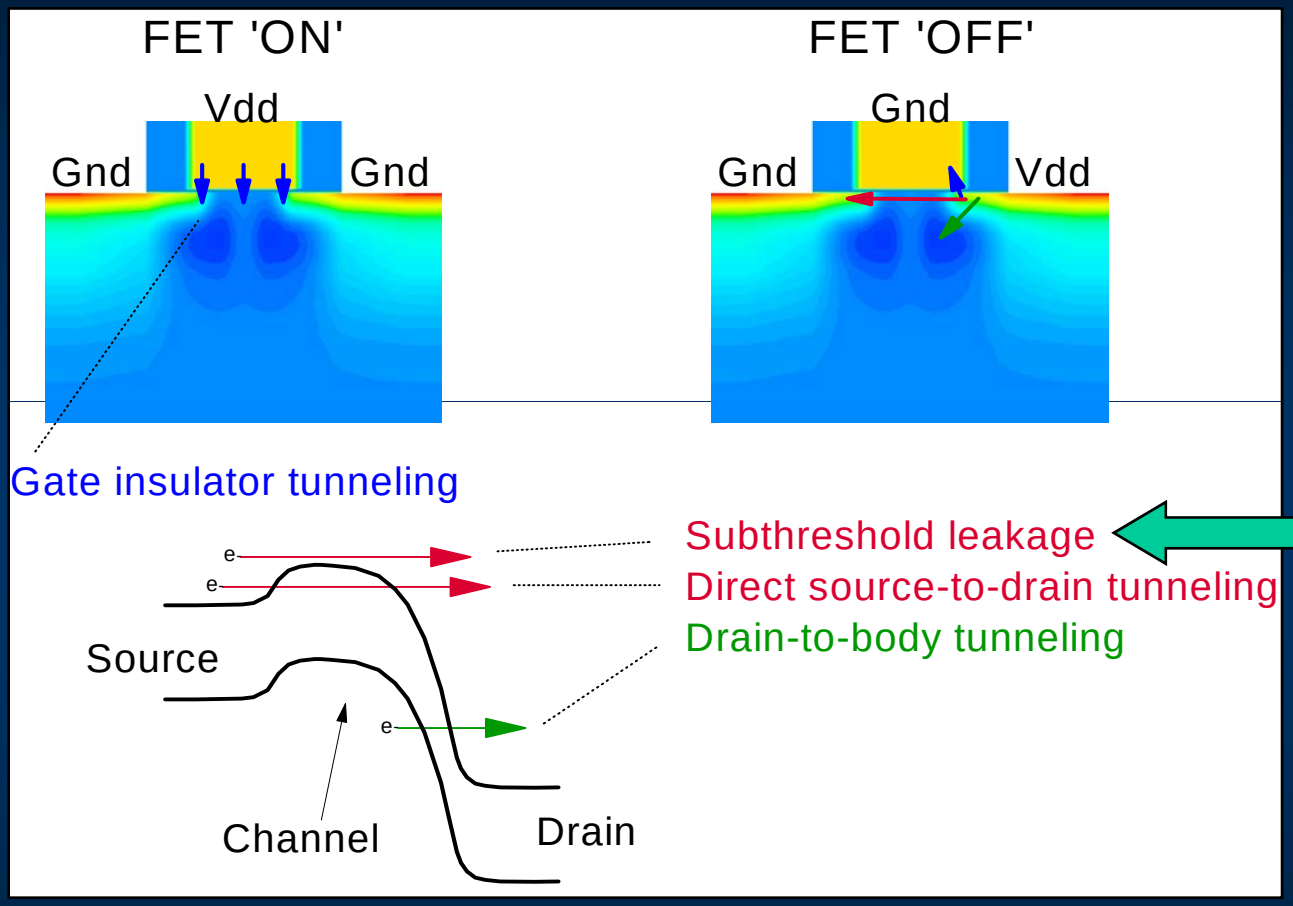






$$E_{sc}\left(\tau_{in}, W, C_{out}\right) = \sum_{i=0}^{1}\sum_{j=0}^{1}\sum_{k=0}^{1} m_{ijk}\,\frac{W^{i}\tau_{in}^{\;j}}{C_{out}^{\;k}}V_{DD}$$

Leakage Power Dissipation



$$I_{sub} = \frac{W}{L}\mu_e v_T^2 C_{sth}\, e^{\frac{V_{GS}-V_{th}'+\eta V_{DS}}{nv_T}}\left(1-e^{\frac{-V_{DS}}{v_T}}\right) \propto e^{\frac{V_{GS}-V_{th}'+\eta V_{DS}}{nv_T}} = 10^{\frac{V_{GS}-V_{th}}{S}}$$

Outline

- Introduction and review of basic concepts
- **Dynamic voltage and frequency scaling**
- Workload decomposition
- DVFS targeting total system energy reduction
- Conclusion and research directions

Dynamic Voltage & Frequency Scaling

- DVFS is a method to provide variable amount of energy for a task by scaling the operating voltage/frequency.
- Power consumption of a CMOS-based circuit is

$$P = \alpha \cdot C_{eff} \cdot V^2 \cdot f$$

α : *switching factor*
C_{eff} : *effective capacitance*
V : *operating voltage*
f : *operating frequency*

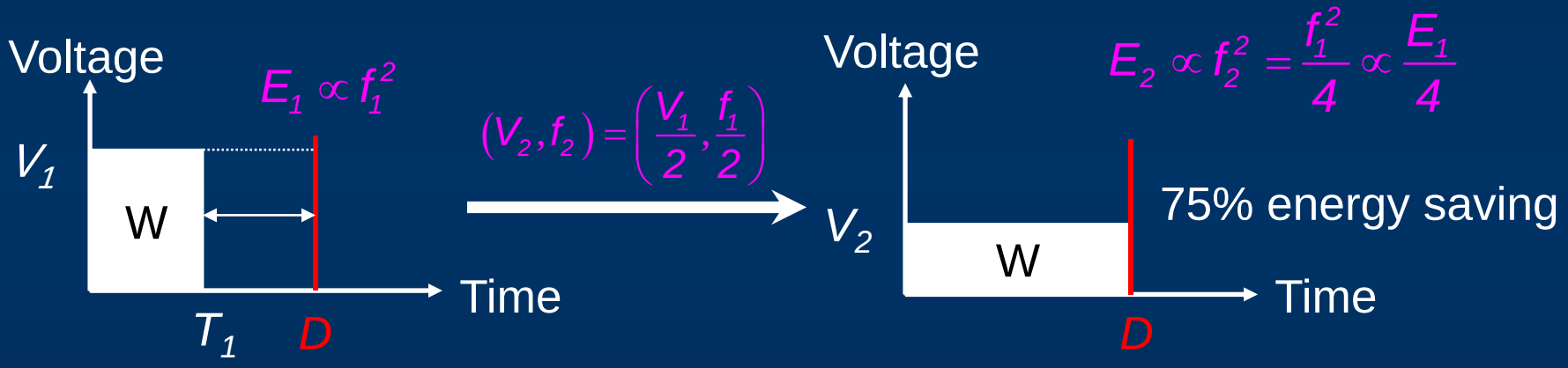
- Energy required to run a task during *T is*

$$E = P \cdot T \propto V^2 \quad \textit{(assuming } V \propto f,\ T \propto f^{-1}\textit{)}$$

- By lowering CPU frequency, CPU energy can be saved.

Energy saving with DVFS

- Example : a task with workload *W* should be completed by a deadline, *D*



- DVFS is an effective way of reducing the CPU energy consumption by providing “just-enough” computation power.

Choosing a frequency in DVFS

- Workload of a task, W_{task}, is defined as the total number of CPU clock cycles required to finish the task

$$W_{task} \equiv \sum_{i=1}^{N} CPI_i$$

N : total number of instructions in a task
CPI : clock cycles per instruction

- Task execution time, T_{task}, is a function of the CPU frequency, f^{cpu}

$$T_{task} = \frac{W_{task}}{f^{cpu}}$$

- Given a deadline of D, f_{target} denotes the CPU frequency that results in T_{task} closest to D

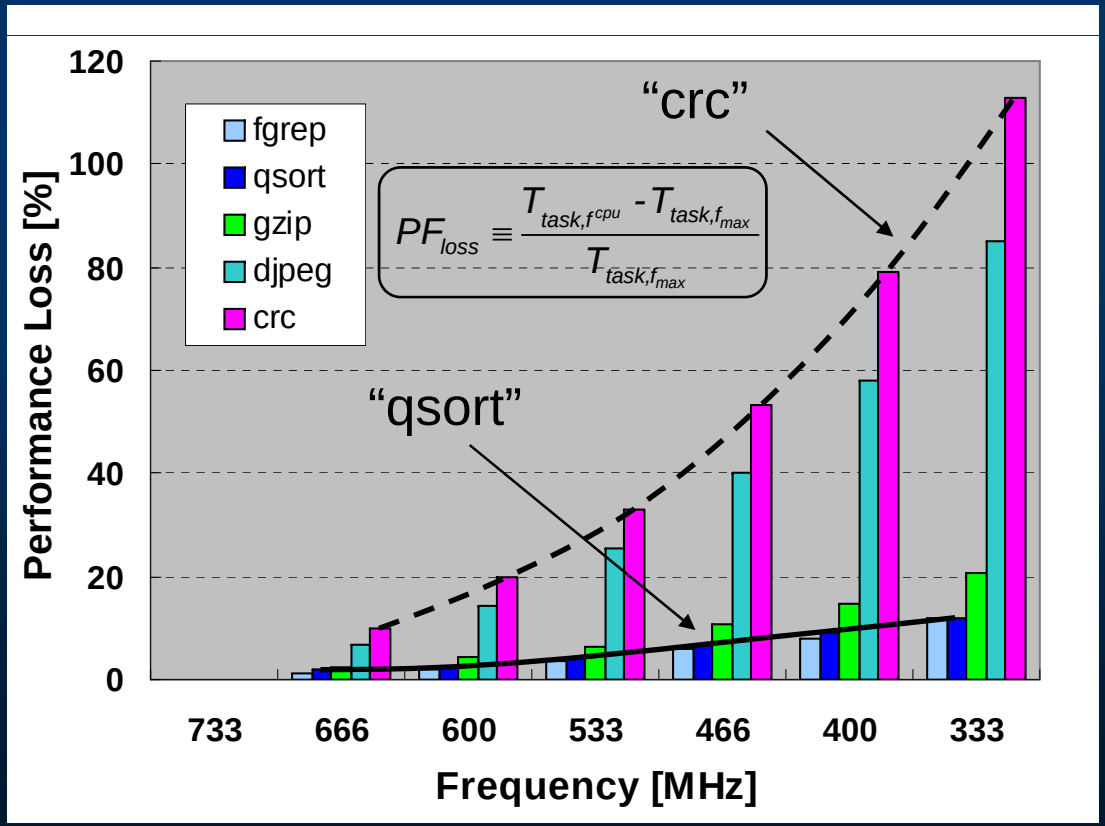
$$f_{target} = \frac{W_{task}}{D} \Rightarrow T_{task} = D$$

Outline

- Introduction and review of basic concepts
- Dynamic voltage and frequency scaling
- **Workload decomposition**
- DVFS targeting total system energy reduction
- Conclusion and research directions

Motivation for workload decomposition

- CPU-bound vs. memory-bound applications
  - The figure shows execution time variation according to the CPU frequency ranging from 733 MHz to 333 MHz.



PF_{loss} @ 333MHz

	PF_{loss}
expected*	120 %
"crc"	110 %
"qsort"	10 %

*assume $T_{task} = \frac{W_{task}}{f^{cpu}}$

Workload decomposition

- A program execution sequence consists of on-chip and off-chip work
  - On-chip work : performed inside the CPU (e.g. cache hit)
  - Off-chip work : performed outside the CPU (e.g. cache miss).
- An external memory access is asynchronous to the CPU
  - Execution time variation as a function of the CPU frequency depends on the workload composition of the task.

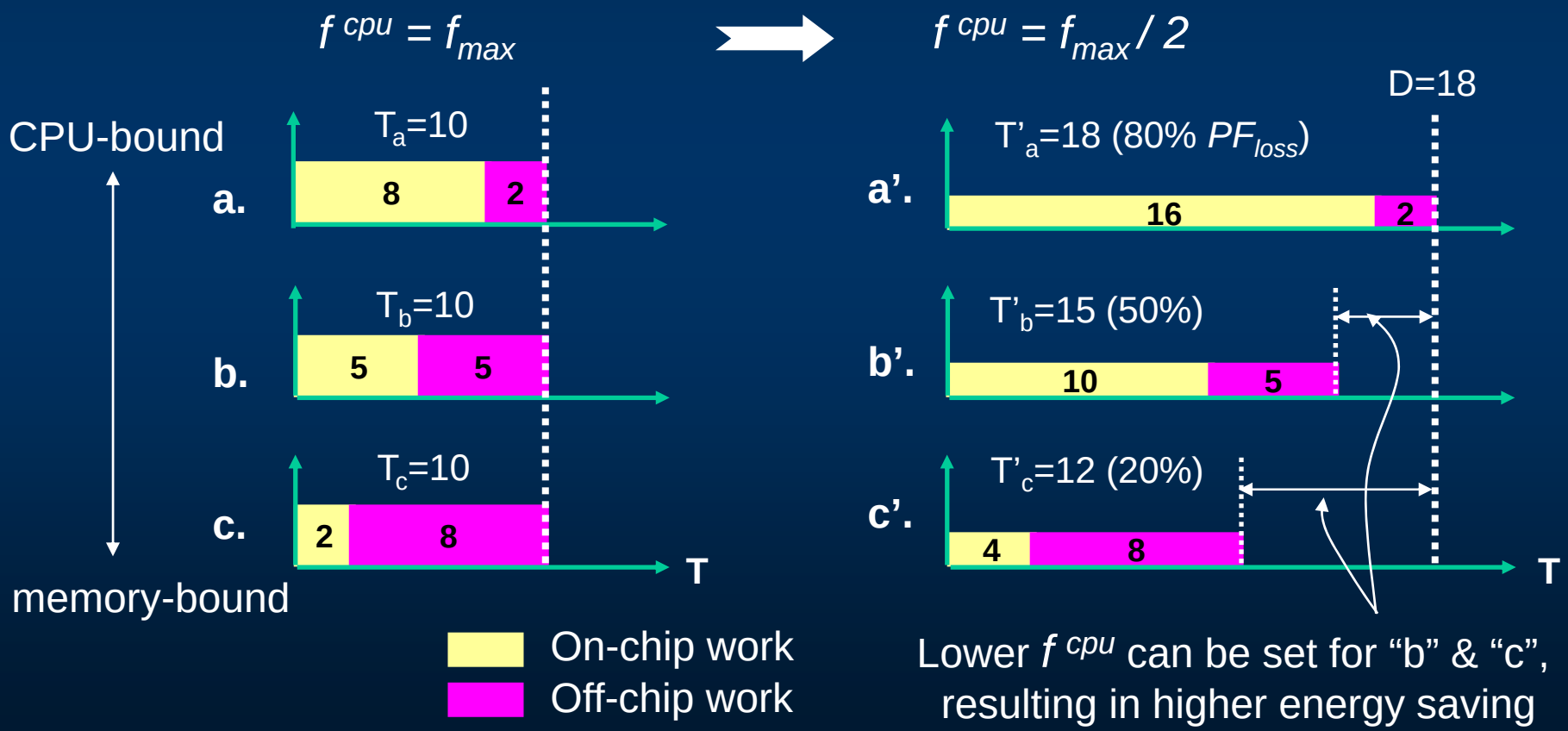
$$T_{task} = \frac{W^{on}}{f^{cpu}} + \frac{W^{off}}{f^{ext}} \quad \Rightarrow \quad \frac{\partial T_{task}}{\partial f^{cpu}} = -\frac{W^{on}}{f^{cpu^2}}$$

Varied *Fixed*

(333MHz to 733MHz) *(100MHz for SDRAM access)*

Energy saving via workload decomposition

- For memory-bound application programs, CPU energy can be saved with a rather low performance penalty.



DVFS with workload decomposition

- With workload decomposition, the target frequency, f_{target}, is calculated as

$$f_{target} = \frac{W^{on}}{D - T^{off}}$$

$$T_{task} = T^{on} + T^{off} = \frac{W^{on}}{f^{cpu}} + \frac{W^{off}}{f^{ext}}$$

- Note that

$$\frac{W^{on}}{D - T^{off}} \leq \frac{W_{task}}{D}$$

target frequency without workload decomposition

- Workload decomposition–based DVFS results in lower CPU energy consumption due to more aggressive voltage scaling.

Decomposing the workload at runtime

- Utilize the embedded hardware in modern microprocessors, i.e., the performance monitoring unit (PMU).
- PMU can report 15 ~ 20 different dynamic events during execution of a program
  - Cache hit/miss counts
  - TLB hit/miss counts
  - No. of stall cycles
  - Total no. of instructions being executed
  - Branch misprediction counts.
- Based on events from the PMU, workload may accurately be decomposed at runtime.

Outline

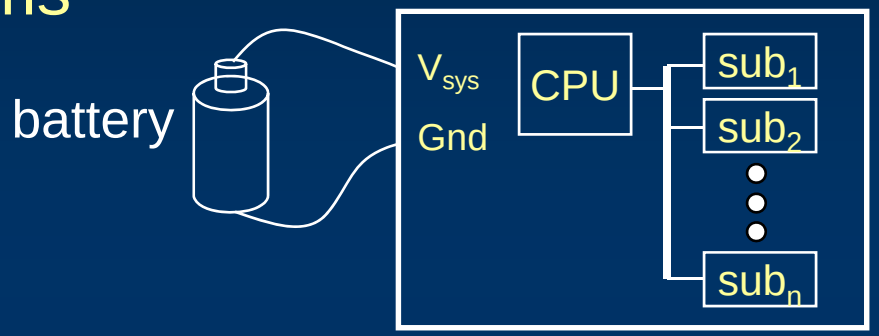
- Introduction and review of basic concepts
- Dynamic voltage and frequency scaling
- Workload decomposition
- **DVFS targeting total system energy reduction**
- Conclusion and research directions

Overview of prior DVFS works

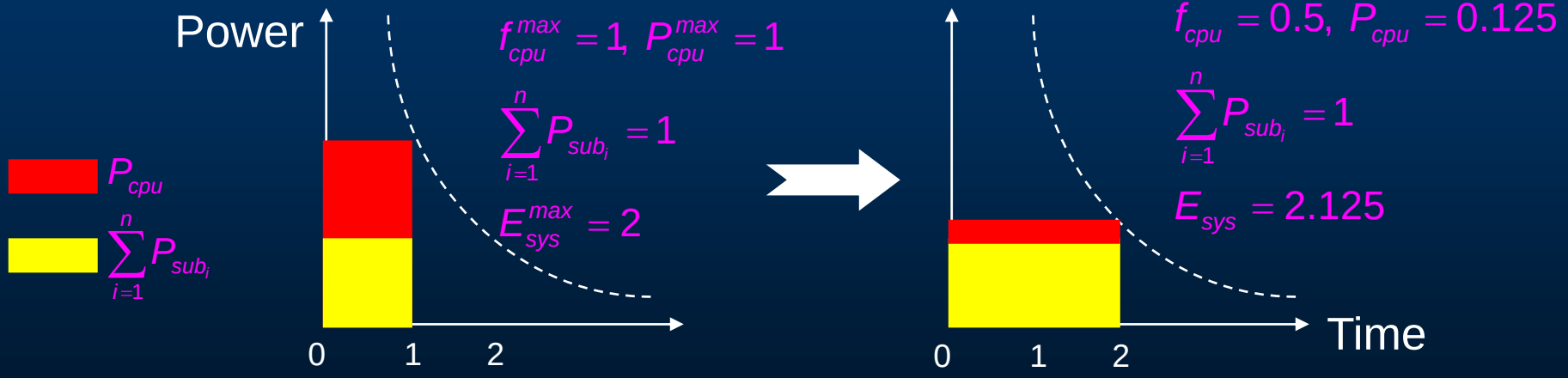
- Most DVFS methods are concerned about CPU energy reduction only
  - More precisely, dynamic portion of the CPU energy
  - Lower CPU frequency always causes less CPU energy.
- Most computing systems, however, comprise of many subsystems such as memory subsystems and peripheral devices.
  - Battery lifetime also depends on power consumption in subsystems, which is not affected by CPU frequency changes.
  - Lowering CPU frequency can cause shorter battery lifetime due to an increase in the standing and idle portions of the system energy consumption.

System energy with DVFS

- A computing system with a CPU and *n* memory and I/O subsystems



- Consider a CPU-bound application as an example



6.25% higher energy consumption

DVFS for the minimal system energy

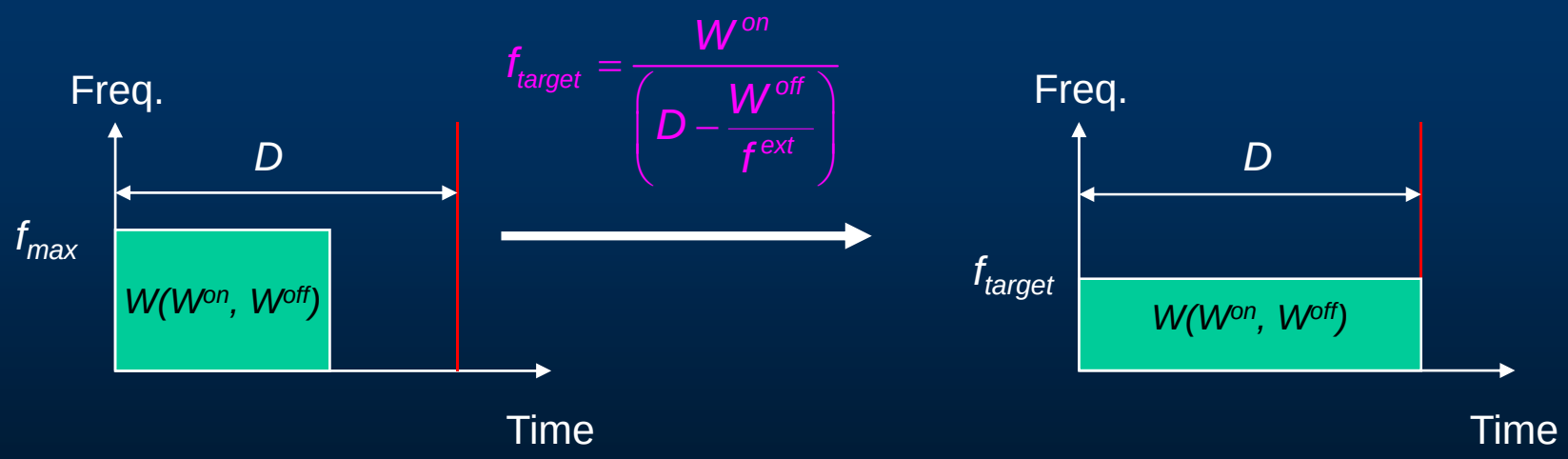
- Timing constraint
  - Different applications exhibit different execution time variation as a function of the CPU frequency.
  - Need an accurate task execution time model as a function of the CPU frequency.
- Minimal system energy
  - Each component's power consumption must be known a priori.
  - Information about the power state of each component (i.e., *active* or *idle* state info) is also required.
- These two requirements can be satisfied by using the workload decomposition approach.

Timing constraint

- Program execution time, T

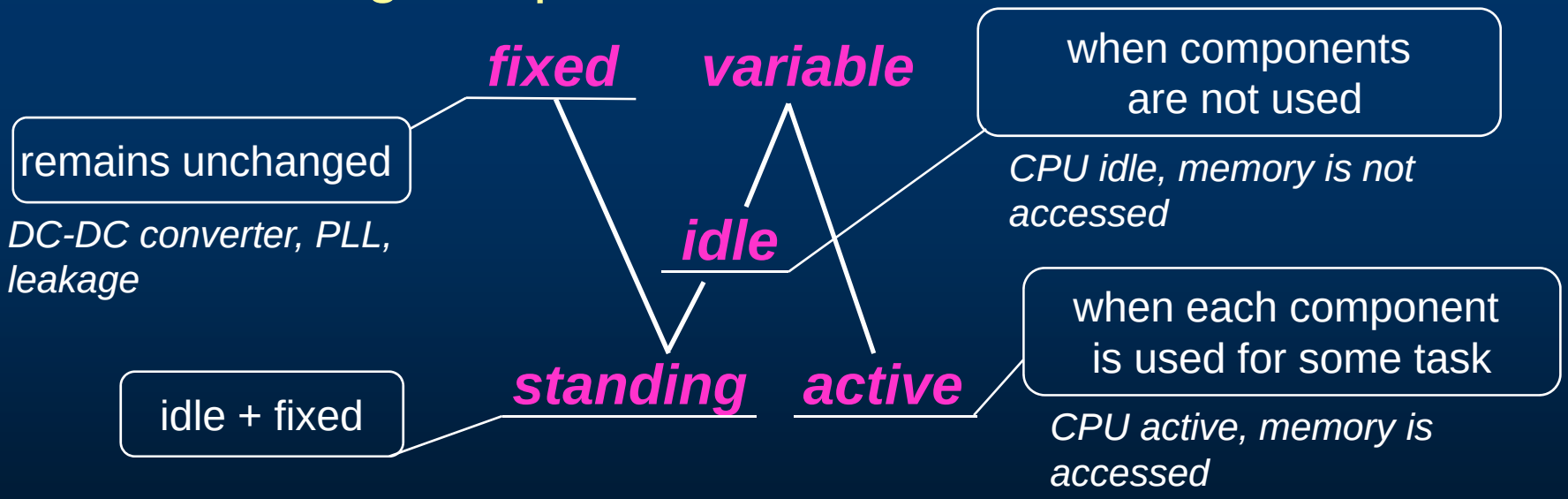
$$T = T^{on} + T^{off} = \frac{W^{on}}{f^{cpu}} + \frac{W^{off}}{f^{ext}}$$

- Given a task with workload, W^{on} and W^{off}, and latency constraint, D



System power breakdown

- Power consumption profile severely fluctuates due to alternate execution of W^{on} and W^{off}.
  - W^{on} (W^{off}) results in CPU (subsystem) power consumption.
- System power consumption may thus be divided into the following components:

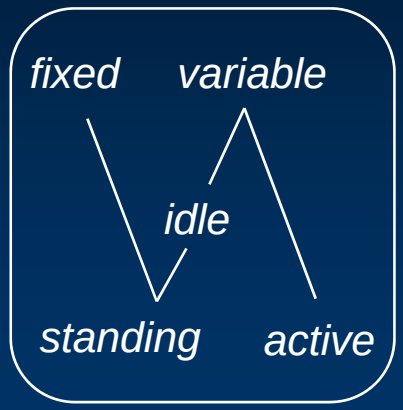


Obtained by *simple measurements* or using *values in the spec.*

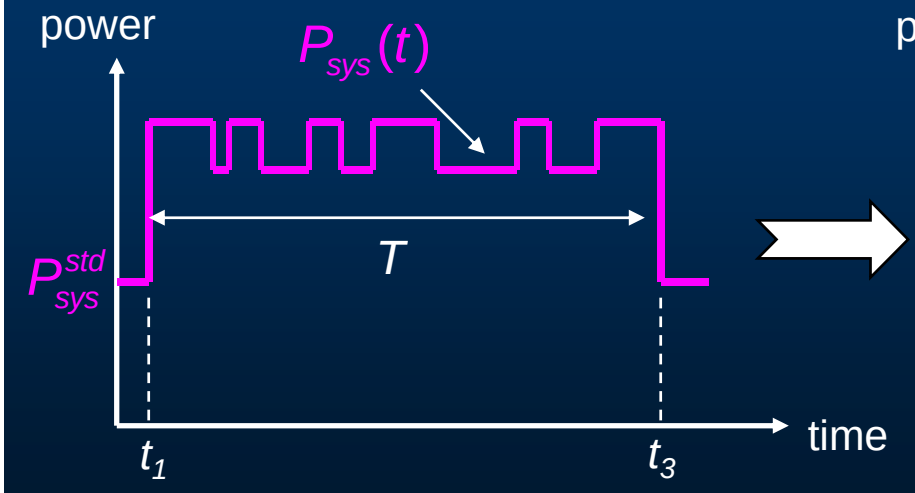
System energy model

- Using workload decomposition, the system energy is modeled as:

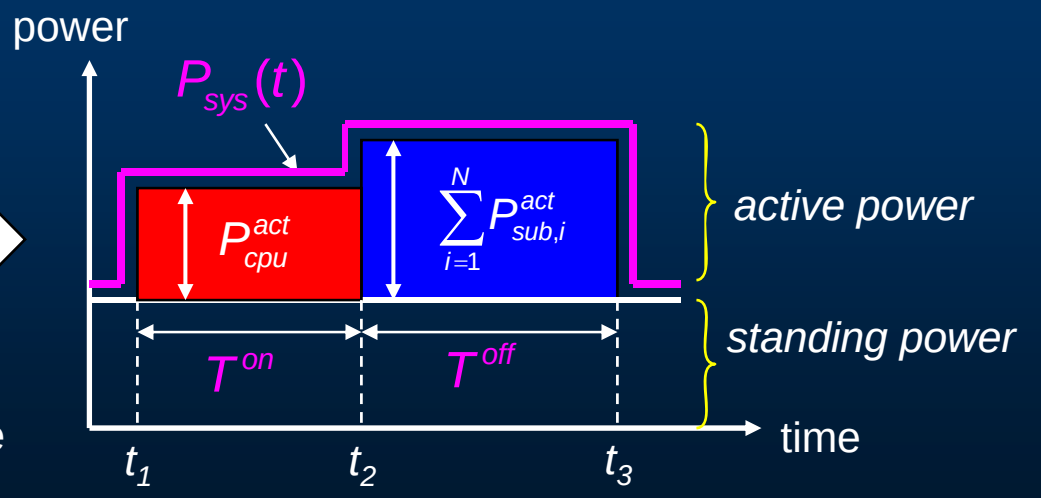
$$E_{sys} = \int_{t_1}^{t_3} P_{sys}(t) = P_{sys}^{std} \cdot T + P_{cpu}^{act} \cdot T^{on} + \sum_{i=1}^{N} P_{sub,i}^{act} \cdot T^{off}$$



without decomposition

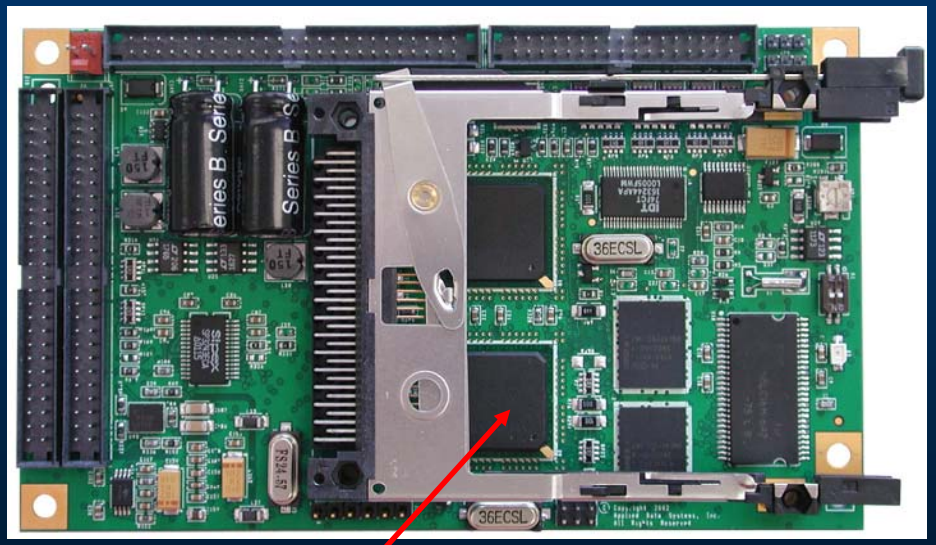


with decomposition



The BitsyX Platform

- The ADS's BitsyX board has a PXA255 microprocessor which is a 32-bit RISC processor core, with a 32KB instruction cache and a 32KB write-back data cache, a 2KB mini-cache, a write buffer, and a memory management unit (MMU) combined in a single chip.



Clock frequencies in BitsyX

- PXA255 can operate from 100MHz to 400MHz, with a core supply voltage of 0.85V to 1.3V → f^{cpu}.
- Internal bus (PXbus) connects the core and other functional blocks inside the CPU → f^{int}.
- External bus is connected to SDRAM (64MB) → f^{ext}
- When f^{cpu} is changed, f^{int} and f^{ext} are also changed.

Frequency settings in BitsyX

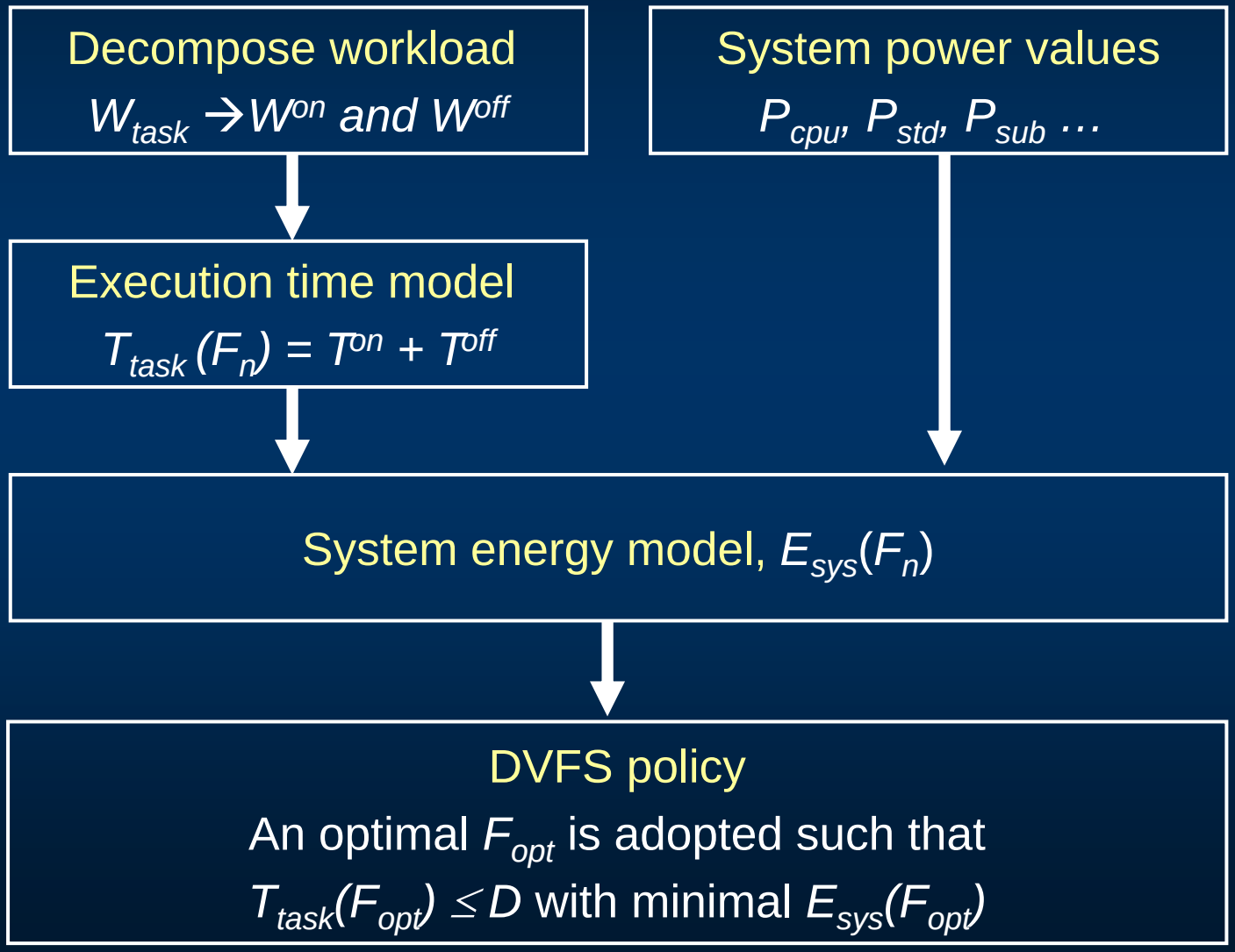
- Nine frequency combinations, $F_n(f^{cpu}, f^{int}, f^{ext})$

Freq. Set	f^{cpu} [MHz]	V^{cpu} [V]	f^{int} [MHz]	f^{ext} [MHz]
F_1	100	0.85	50	100
F_2	133	0.85	66	133
F_3	200	1.0	50	100
F_4	200	1.0	100	100
F_5	265	1.0	133	133
F_6	300	1.1	50	100
F_7	300	1.1	100	100
F_8	400	1.3	100	100
F_9	400	1.3	200	100

Performance monitoring unit (PMU)

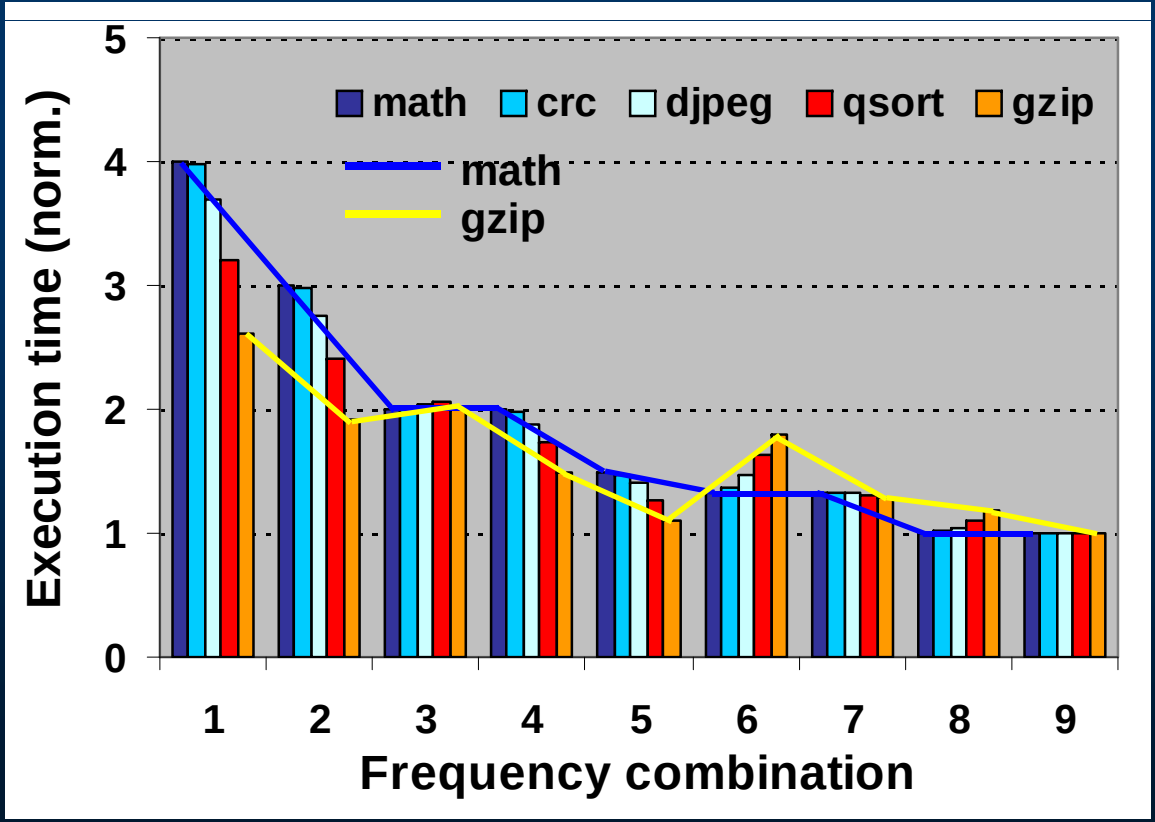
- PMU on the PXA255 processor can report up to 15 different dynamic events at run time.
  - Cache hit/miss counts, TLB hit/miss counts, No. of stall cycles, Total no. of instructions being executed, Branch misprediction counts.
- For DVFS, we use the PMU to generate statistics for
  - Total no. of instructions being executed (INSTR)
  - No. of stall cycles due to on/off-chip data dependencies (STALL)
  - No. of Data Cache misses (DMISS)
- We also record the no. of clock cycles from the beginning of the program execution (CCNT).

Algorithm flow of the SE-DVFS for BitsyX



Execution time and frequency settings

- Execution time variation for different frequency combinations – “math”, “crc”, “djpeg”, “qsort”, “gzip”
  - “math” is CPU-bound (strongly dependent on f^{cpu})
  - “gzip” is memory-bound (f^{int} & f^{ext} dependent)



Freq. Set	f^{cpu} (MHz)	f^{int} (MHz)	f^{ext} (MHz)
F_1	100	50	100
F_2	133	66	133
F_3	200	50	100
F_4	200	100	100
F_5	265	133	133
F_6	300	50	100
F_7	300	100	100
F_8	400	100	100
F_9	400	200	100

On-chip workload, W^{on}

- Total workload, W_{task}, is modeled as:

$$W_{task} = \sum_{i=1}^{N} CPI^{i} = \underline{N} \cdot \underline{CPI^{avg}}$$

number of instructions

CPU clocks per instruction

$$= N \cdot \left(\underline{CPI_0} + \underline{CPI^{avg}_{branch\_miss}} + \underline{CPI^{avg}_{stall\_on} + CPI^{avg}_{stall\_off}} \right)$$

CPU clocks due to on-chip (off-chip) stalls

CPU clocks due to branch misprediction overhead

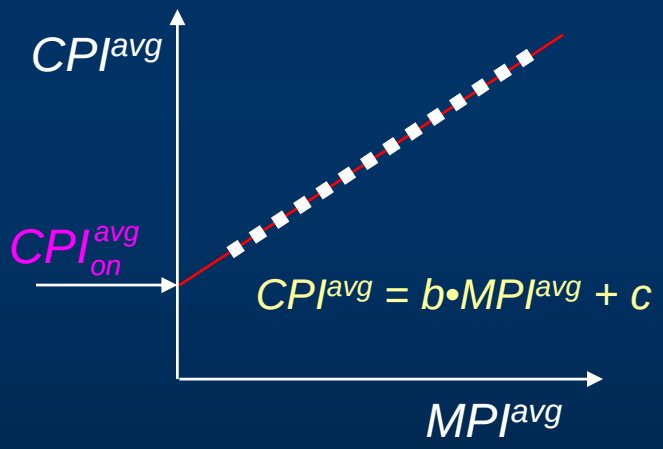
ideal *CPI* which is 1 for a single-issue processor

- On-chip workload, W^{on}, is given as:

$$W^{on} = \underline{N \cdot CPI^{avg}_{on}} = N \cdot \left(CPI_0 + CPI^{avg}_{branch\_miss} + CPI^{avg}_{stall\_on} \right)$$

On-chip CPI calculation

- In our previous work on XScale80200-based system (DATE'04), CPI_{on}^{avg} was obtained from the plot of CPI^{avg} vs. MPI^{avg}.



$$CPI^{avg} = \frac{CCNT}{INSTR}, \quad MPI^{avg} = \frac{MEM}{INSTR}$$

XScale-80200 processor

events	description
CCNT	number of clock counts
INSTR	number of executed instructions
MEM	number of off-chip accesses

- *MEM* is not provided by the PMU of PXA255 processor
  - Instead, we used *STALL* and *DMISS* events.

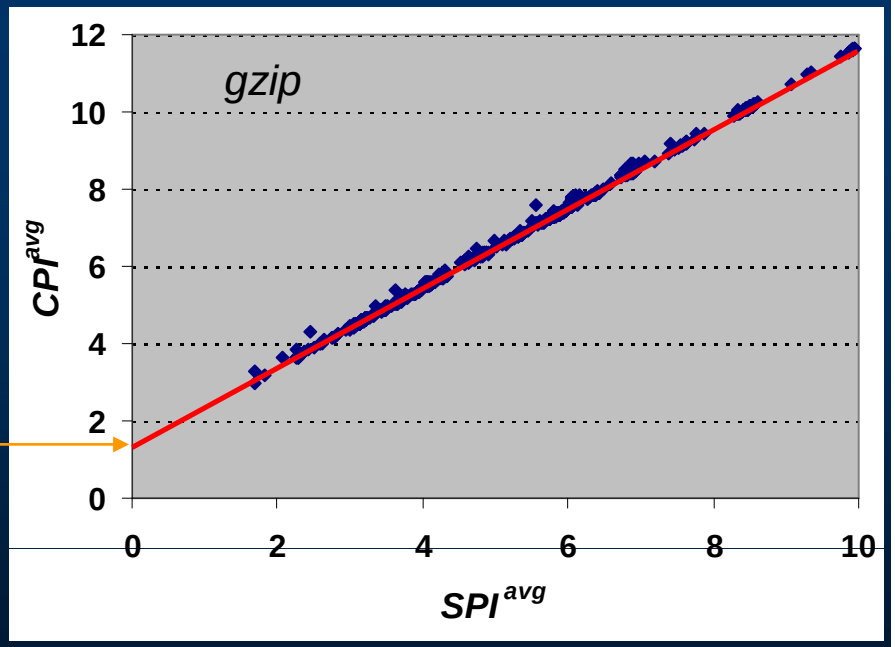
Plot of CPI^{avg} vs. SPI^{avg}

- We define *SPI* as ratio of the number of stall cycles to the total instruction count

$$SPI^{avg} \equiv \frac{STALL}{INSTR} = SPI_{on}^{avg} + SPI_{off}^{avg}$$

$$CPI_{on}^{avg} = CPI_0 + CPI_{branch\_miss}^{avg} + CPI_{stall\_on}^{avg}$$

CPI value without any stall cycles → CPI_{on}^{min}



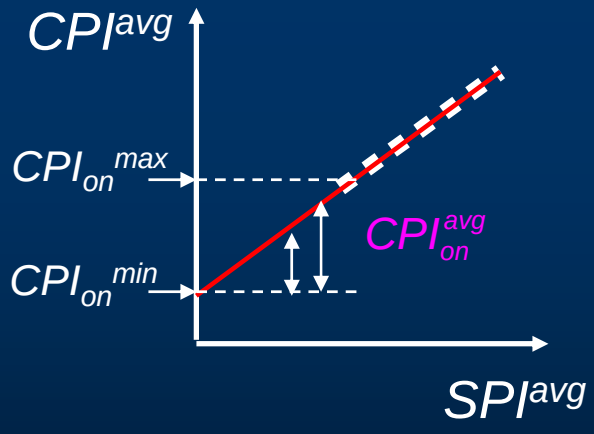
- Then, W^{on} is rewritten as:

$$W^{on} = N \cdot CPI_{on}^{avg}$$
$$= N \cdot \left(CPI_{on}^{min} + SPI_{on}^{avg} \right)$$

Calculating SPI_{on} using D-cache miss

- The more D-cache miss events, the higher the probability of off-chip accesses → most stalls are off-chip stalls.
- We define *DPI* as ratio of the number of D-cache miss events to the total instruction count.

$$DPI^{avg} \equiv \frac{DMISS}{INSTR}$$



$$DF = \frac{(CPI_{on}^{max} - CPI_{on}^{min})}{n}$$

$$CPI_{on}^{avg} = CPI_{on}^{min} + SPI_{on}^{avg} = CPI_{on}^{min} + dpi2spi(DPI)$$

DPI	dpi2spi(DPI)	CPI_{on}^{avg}	
$DPI \le k_1$	$CPI_{on}^{max} - CPI_{on}^{min}$	CPI_{on}^{max}	CPU-bound
$k_1 < DPI \le k_2$	$DF \bullet (n-1)$	$CPI_{on}^{min} + (n-1) \cdot DF$	
…	…	…	
$k_{n-2} < DPI \le k_{n-1}$	$DF \bullet 2$	$CPI_{on}^{min} + 2 \cdot DF$	
$k_{n-1} < DPI \le k_n$	$DF \bullet 1$	$CPI_{on}^{min} + 1 \cdot DF$	
$DPI > k_n$	0	CPI_{on}^{min}	mem-bound

k_n is constant: $k_1 < k_2 < \ldots < k_n$

Calculating W^{off} from T^{off}

- W^{off} is calculated from T^{off} which is given as

$$T_{F_n}^{off} = T_{task}(F_n) - T_{F_n}^{on} = \frac{CCNT}{f_{F_n}^{cpu}} - \frac{W^{on}}{f_{F_n}^{cpu}}$$

- T^{off} is dependent on the f^{ext} as well as f^{int}.
- Example: when a D-cache miss occurs, two operations are performed:
  - Data fetch from the external memory (f^{ext})
  - Data transfer to the CPU core where the cache-line and destination register are updated (f^{int})
- Due to lack of exact timing information, we have opted to model T^{off} as:

$$T_{F_n}^{off} = T_{int,F_n}^{off} + T_{ext,F_n}^{off} = \frac{\alpha \cdot W^{off}}{f_{F_n}^{int}} + \frac{(1-\alpha) \cdot W^{off}}{f_{F_n}^{ext}}$$

Execution time model summary

- Execution time, $T_{task}(F_n)$, in BitsyX system

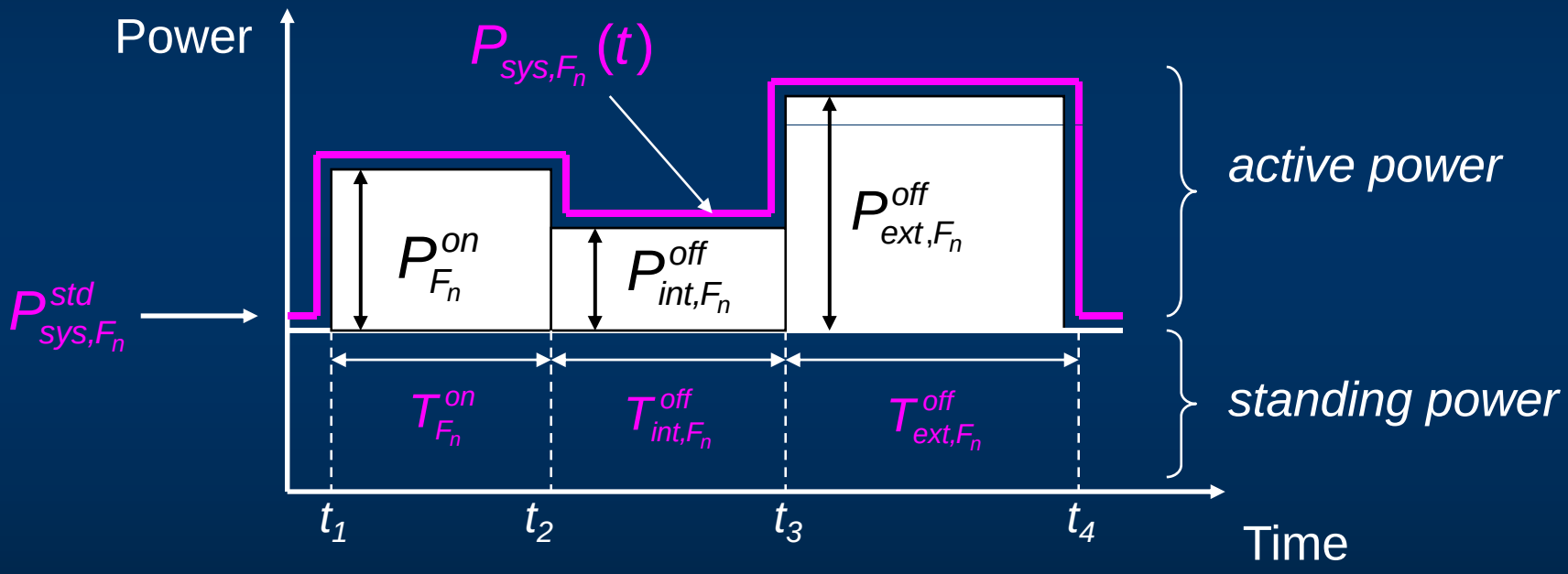
$$T_{task}(F_n) = T_{F_n}^{on} + T_{int,F_n}^{off} + T_{ext,F_n}^{off} = \frac{W^{on}}{f_{F_n}^{cpu}} + \frac{\alpha \cdot W^{off}}{f_{F_n}^{int}} + \frac{(1-\alpha) \cdot W^{off}}{f_{F_n}^{ext}}$$

- An α value of ~0.35 was obtained for tested applications
  - The average error in predicting the execution time was less than 2% for all nine frequency settings.

System energy model for BitsyX

- Using workload decomposition

$$T_{F_n} = T^{on}_{F_n} + T^{off}_{int,F_n} + T^{off}_{ext,F_n}$$



$$E_{sys,F_n} = P^{std}_{sys,F_n} \cdot \underline{T} + P^{on}_{F_n} \cdot \underline{T^{on}_{F_n}} + P^{off}_{int,F_n} \cdot \underline{T^{off}_{int,F_n}} + P^{off}_{ext,F_n} \cdot \underline{T^{off}_{ext,F_n}}$$

Accuracy of the system energy model

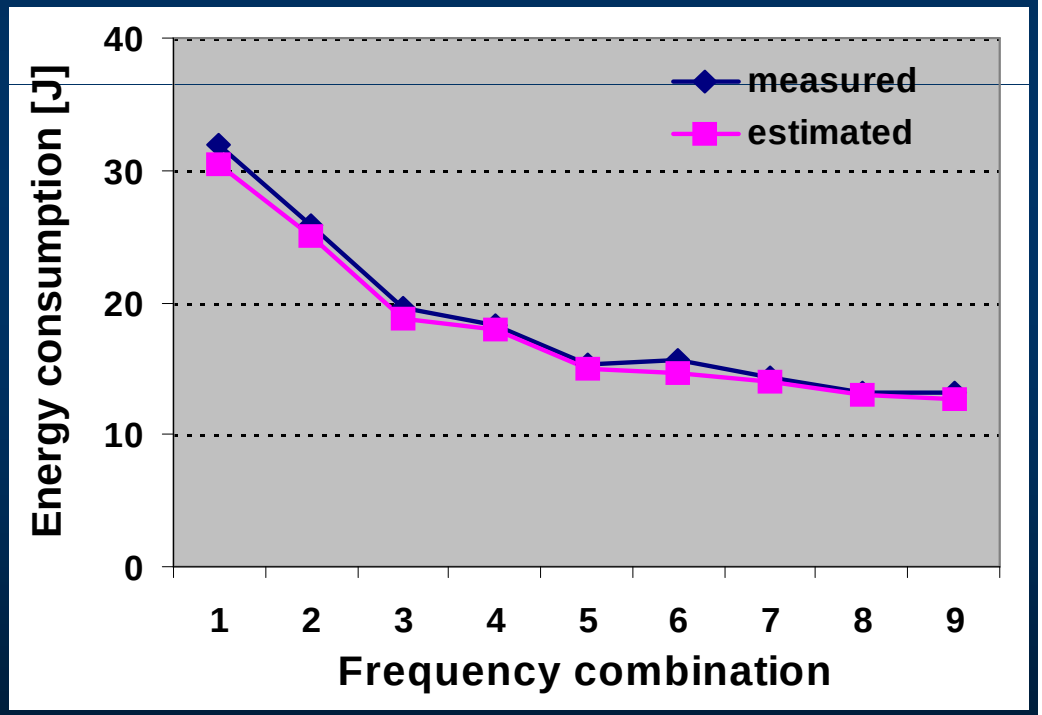
- The estimated energy consumption for “djpeg”
  - The average error rate is less than 4%.

measured parameters [mW]

Freq. set	P^{std}_{sys,F_n}	$P^{on}_{F_n}$	P^{off}_{int,F_n}	P^{off}_{ext,F_n}
F_1	1665	89	363	785
F_2	1757	148	479	785*1.33
F_3	1699	218	456	785
F_4	1728	217	766	785
F_5	1836	336	1018	785*1.33
F_6	1732	344	575	785
F_7	1778	378	885	785
F_8	1869	673	1113	785
F_9	1963	675	1733	785

$$P^{off}_{int,F_n} \sim k_1 \cdot V^2_{F_n} \cdot f^{cpu}_{F_n} + k_2 \cdot f^{int}_{F_n}$$

$k_1 = 0.73$ [nF], $k_2 = 6.2$ [V^2nF]



Scaling granularity

- Fine-grained SE-DVFS may be applied to non real-time applications.
- Considering the scaling overhead, frequent voltage scaling may outweigh any advantage from DVFS.
- OS quantum unit (~60 msec in Linux) is suitable as a scale unit, which is three orders of magnitude larger than the voltage scaling overhead (~500 μsec).
- Using the OS quantum, our DVFS can be applied to each process when it is scheduled.

Determining the optimal frequency setting

- First satisfy the timing constraint; Next find a setting that minimizes the total system energy (*SE-DVFS*).
- Pseudo code for optimal frequency selection:

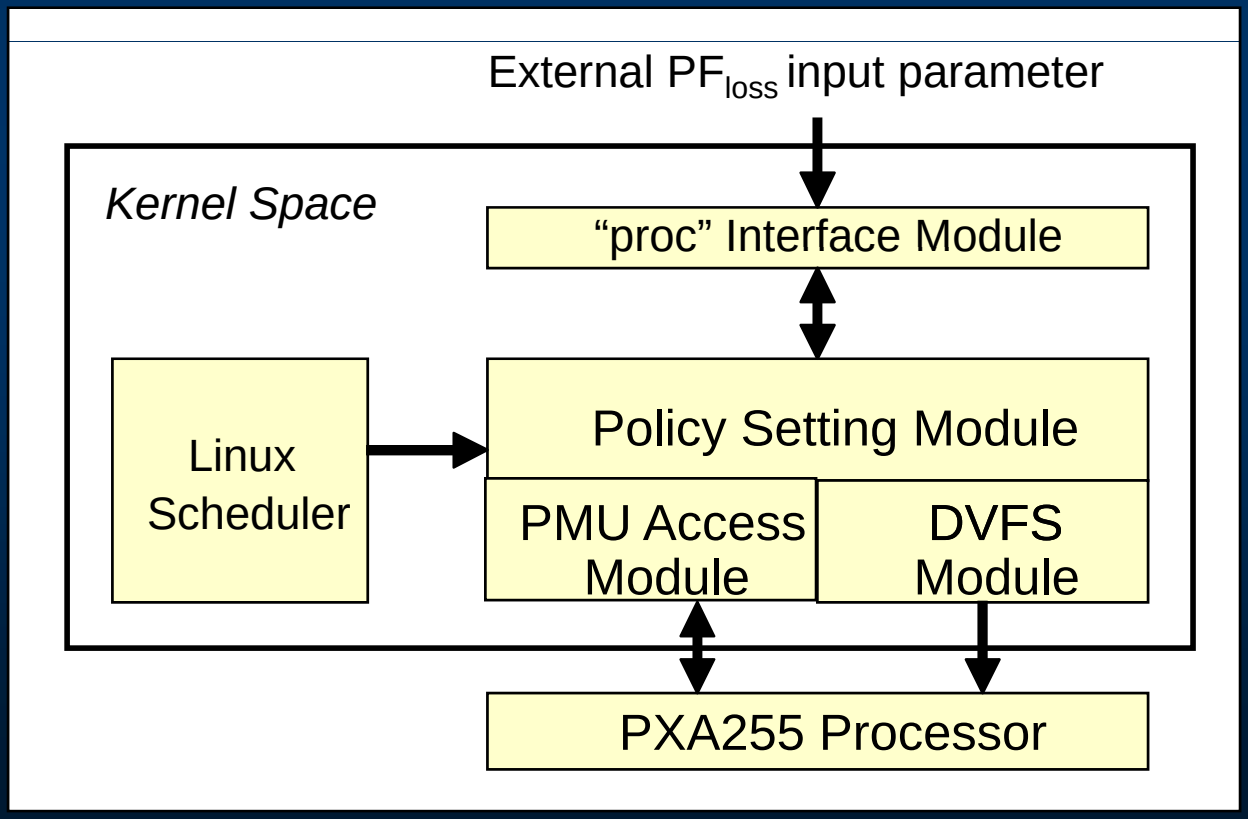
Timing constraint →

System energy minimization →

1. $\Psi = \{ F_{min}, \ldots, F_{max} \}$, $\Gamma = \{\phi\}$, and $E_{min} = \infty$
2. for every frequency setting F_n in Ψ
3. if ($T_{F_n}^{i+1} \leq (1 + PF_{loss}) \cdot T_{F_{max}}^{i}$) ← *CE-DVFS*
4. $\Gamma = \Gamma \cup F_n$;
5. for every frequency setting F_n in Γ
6. calculate system energy using proposed model
7. if ($E_{sys,F_n} \leq E_{min}$)
8. $E_{min} = E_{sys,F_n}$; $F^{opt}_{i+1} = F_n$;

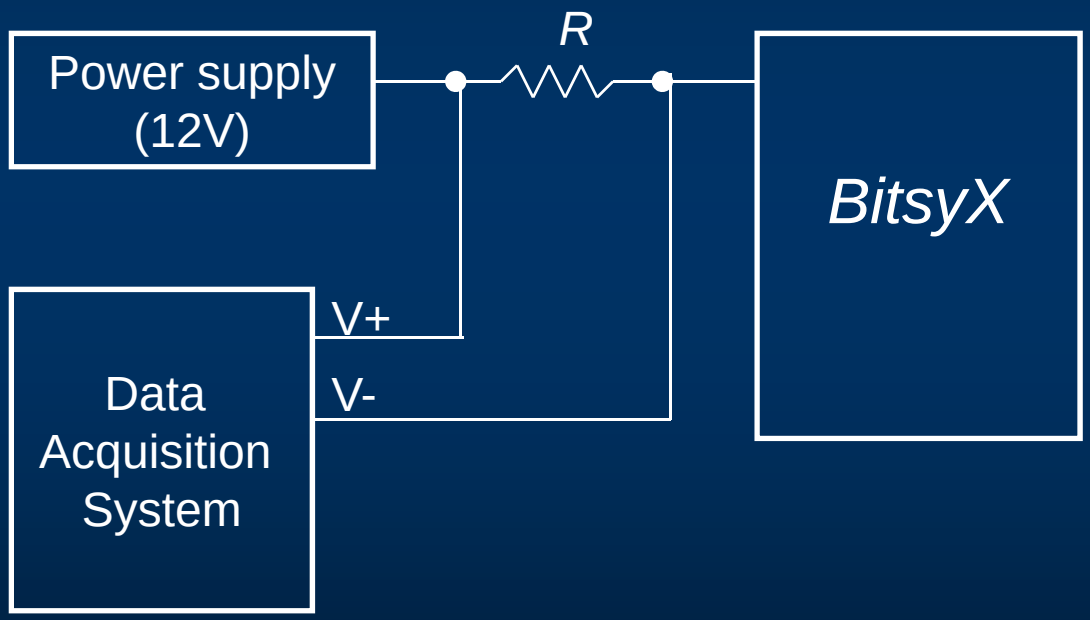
The software architecture

- The software architecture comprises of a proc interface module and a policy setting module tightly linked with the Linux scheduler, the PMU, and the freq. and voltage control circuitry on the BitsyX board.



Power measurement

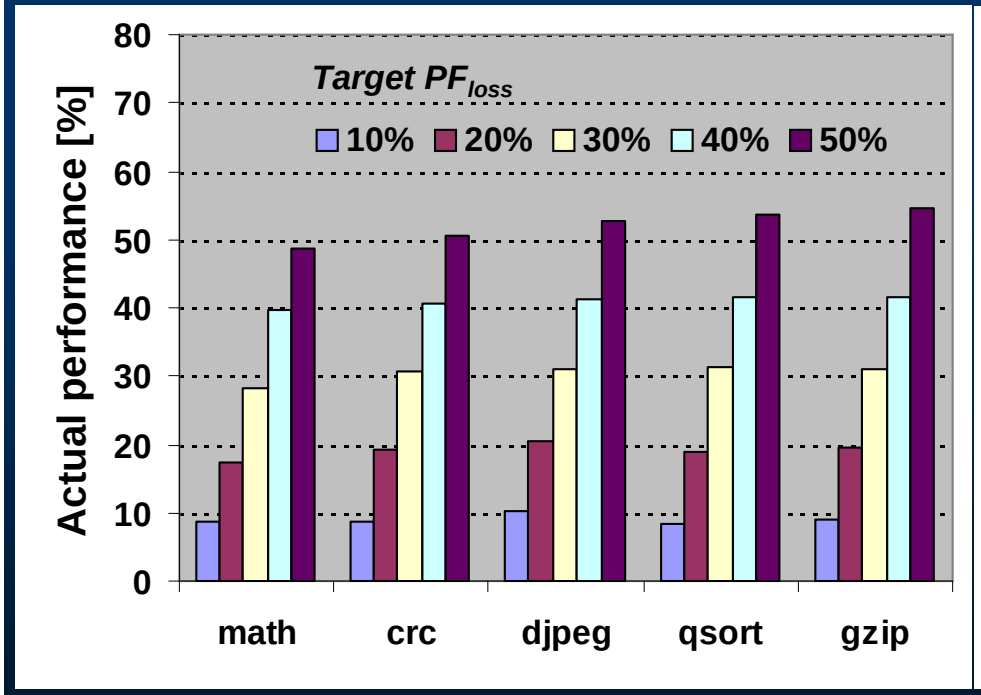
- Data acquisition system operates up to 100 kHz.



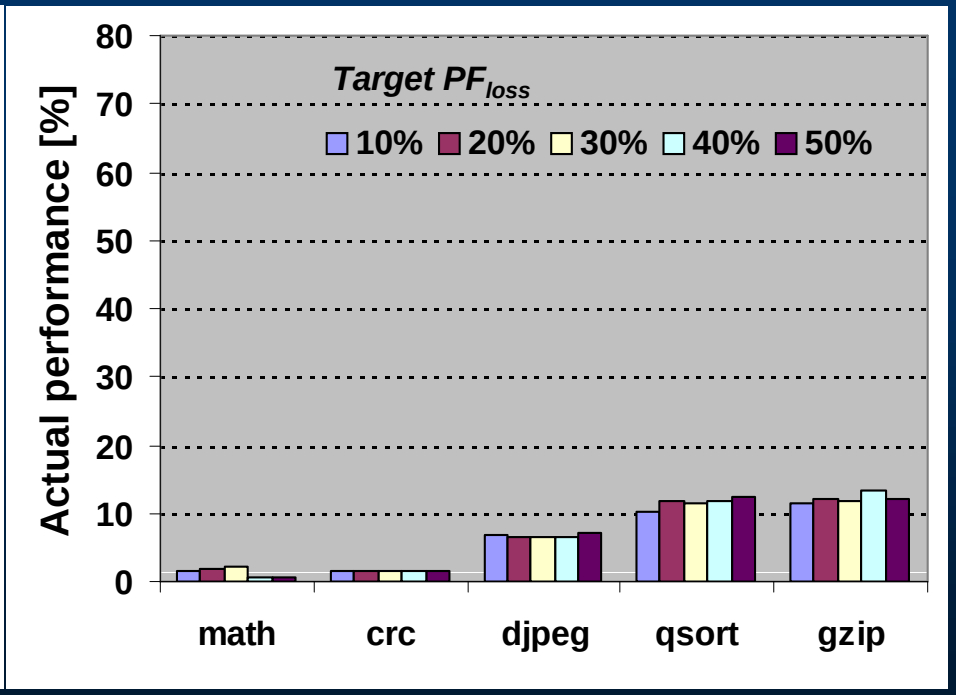
Experimental results (I)

- Comparing two DVFS techniques:
  - *SE-DVFS* vs. *CE-DVFS*
- Resulting performance loss

CE-DVFS



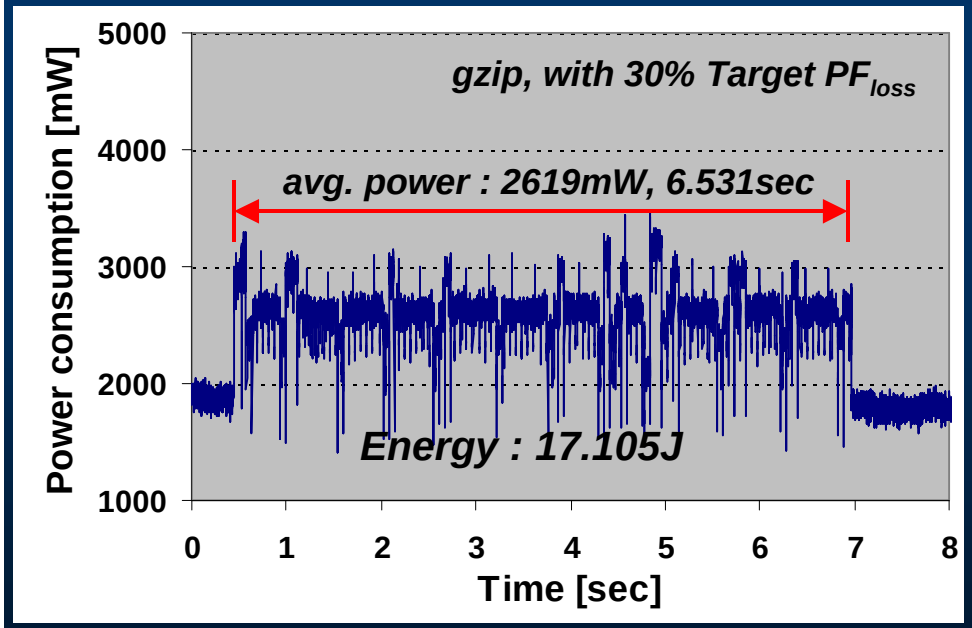
SE-DVFS



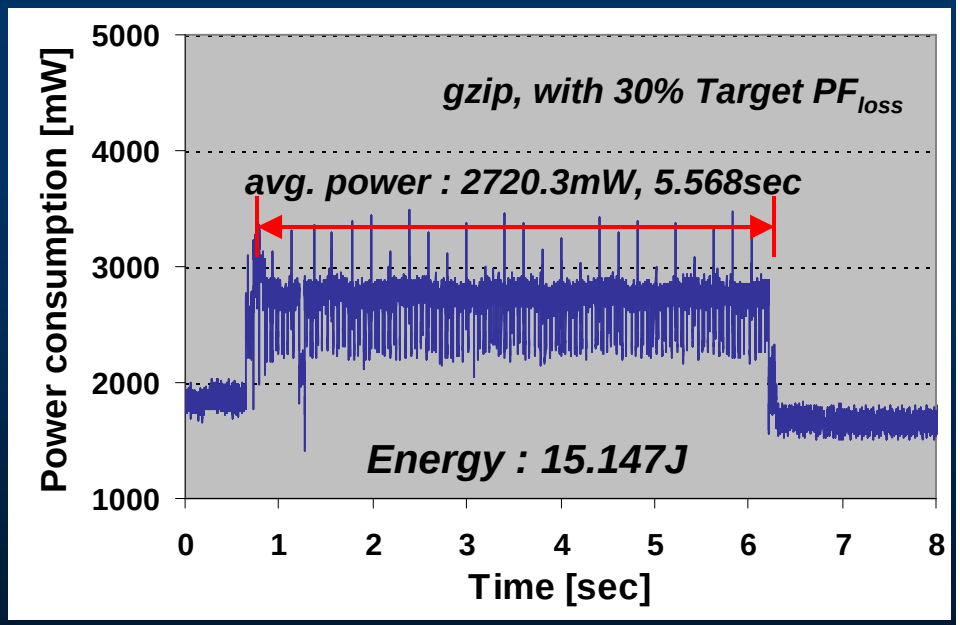
Experimental results (II)

- Actual power consumption of two DVFS methods
- For “gzip” with 30% target PF_{loss}, SE-DVFS results in 11.4% lower total system energy than CE-DVFS

CE-DVFS

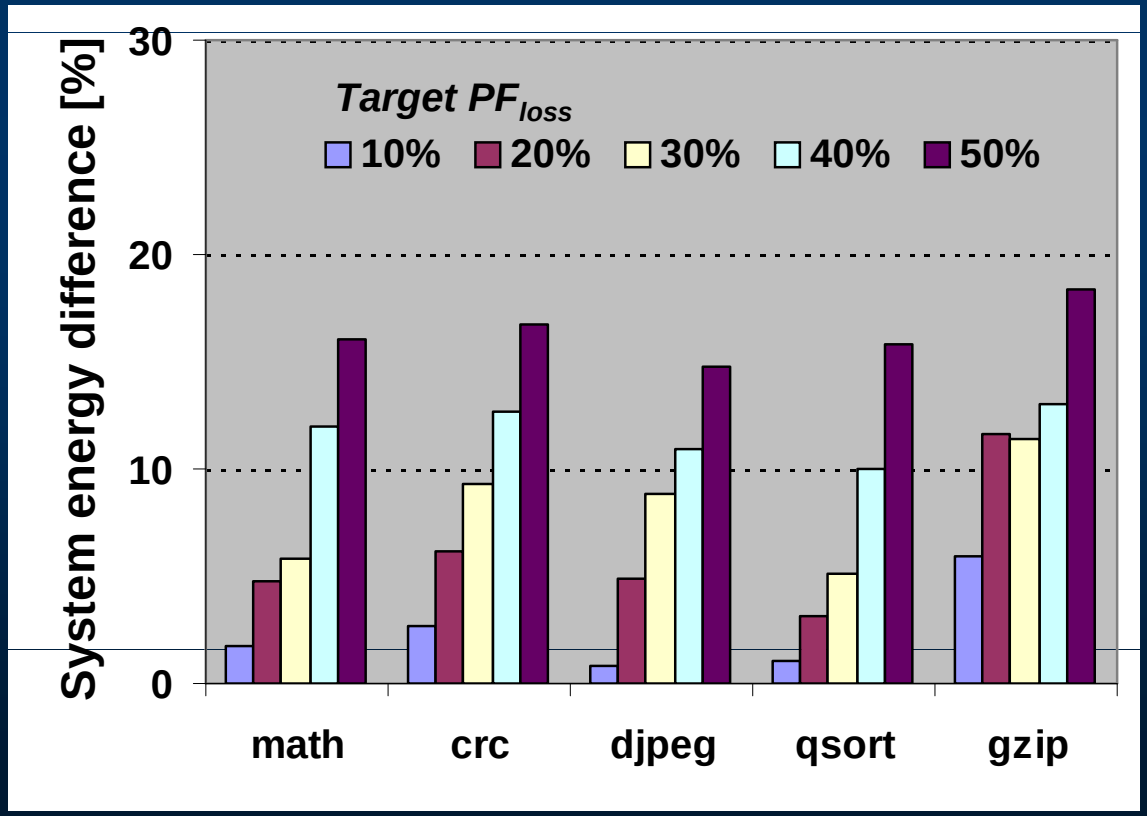


SE-DVFS



Experimental results (III)

- CE-DVFS vs. SE-DVFS
  - SE-DVFS results in 2% ~ 18% higher system energy savings compared to CE-DVFS



Outline

- Introduction and review of basic concepts
- Dynamic voltage and frequency scaling
- Workload decomposition
- DVFS targeting total system energy reduction
- **Conclusion and research directions**

Conclusion

- A DVFS policy for the actual system energy reduction was proposed and implemented, which uses online decomposition of the application workload into on-chip and off-chip components
- Based on actual current measurements in the BitsyX platform, up to 18% more system energy saving was achieved with the proposed DVFS compared with the results in the previous DVFS techniques
- For both CPU and memory-bound programs, the specified timing constraints were satisfied