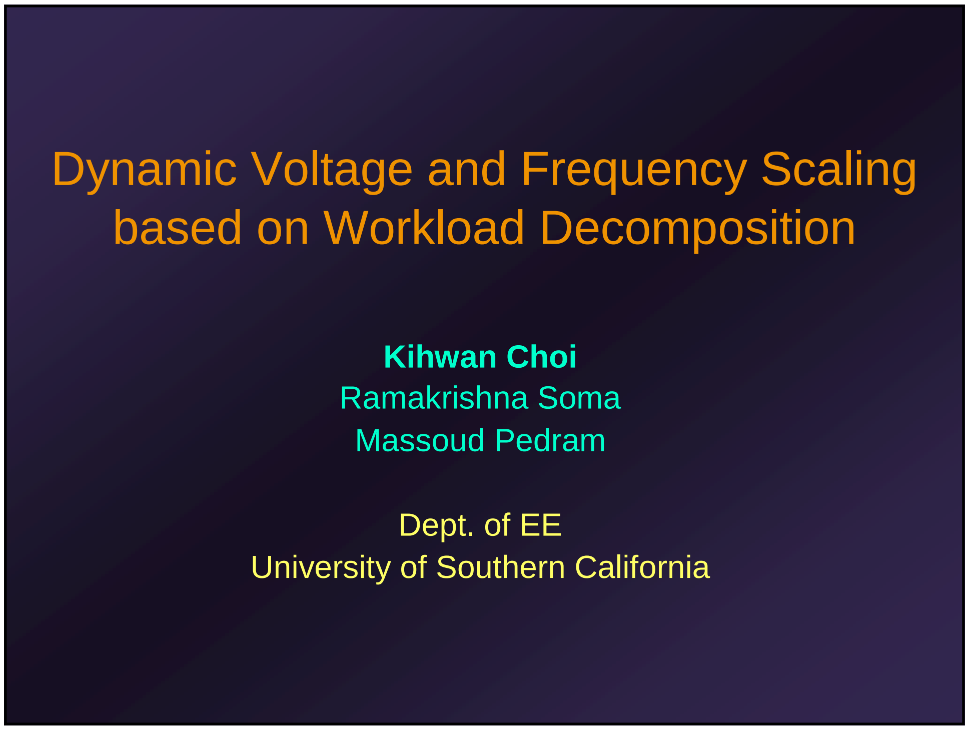
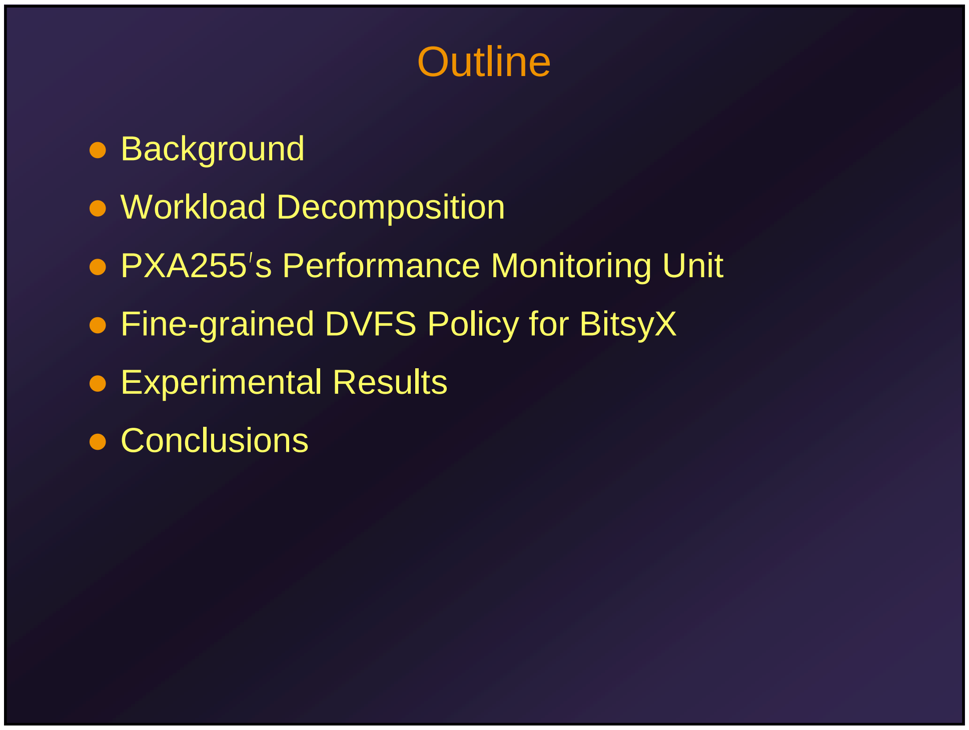


Dynamic Voltage and Frequency Scaling based on Workload Decomposition

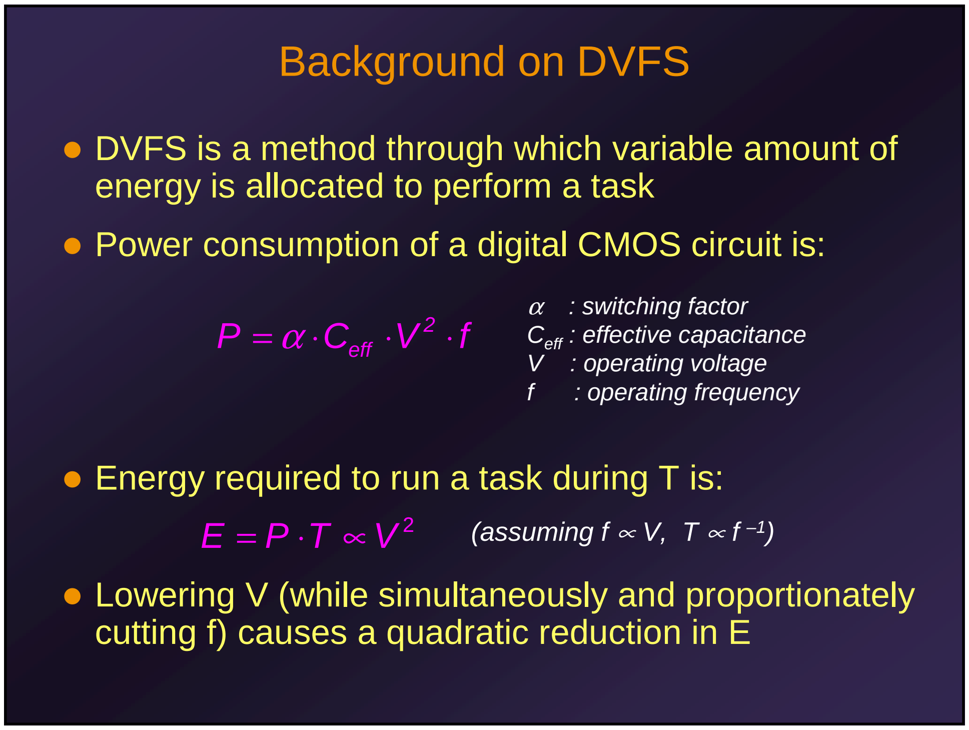
Kihwan Choi
Ramakrishna Soma
Massoud Pedram

Dept. of EE
University of Southern California



Outline

- Background
- Workload Decomposition
- PXA255′s Performance Monitoring Unit
- Fine-grained DVFS Policy for BitsyX
- Experimental Results
- Conclusions



Background on DVFS

- DVFS is a method through which variable amount of energy is allocated to perform a task
- Power consumption of a digital CMOS circuit is:

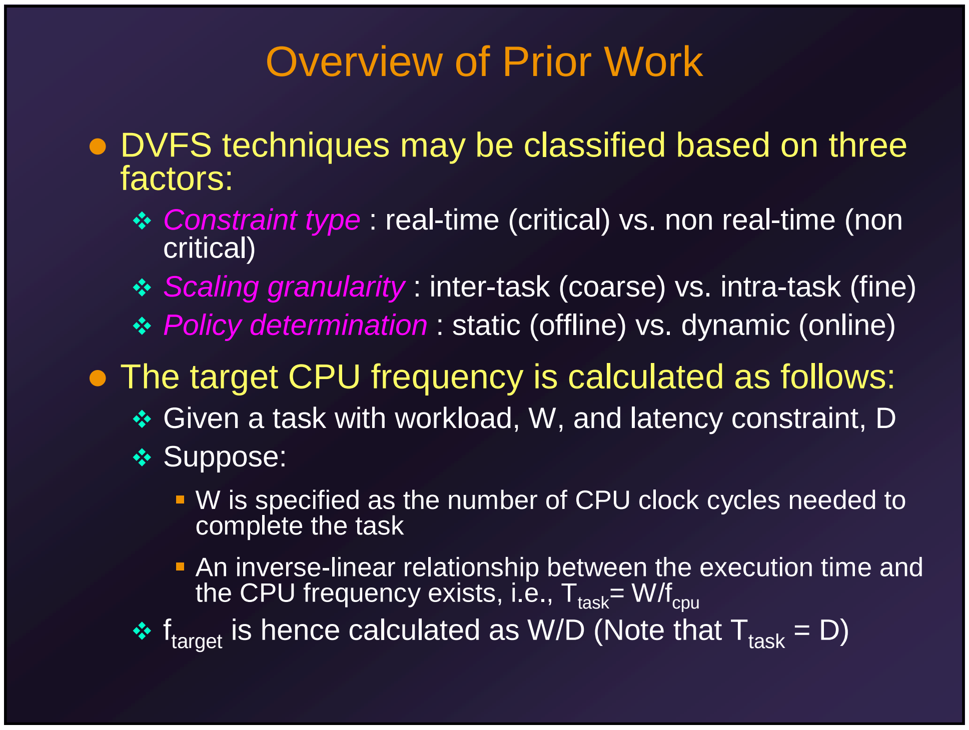
$$P = \alpha \cdot C_{eff} \cdot V^2 \cdot f$$

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

- Energy required to run a task during T is:

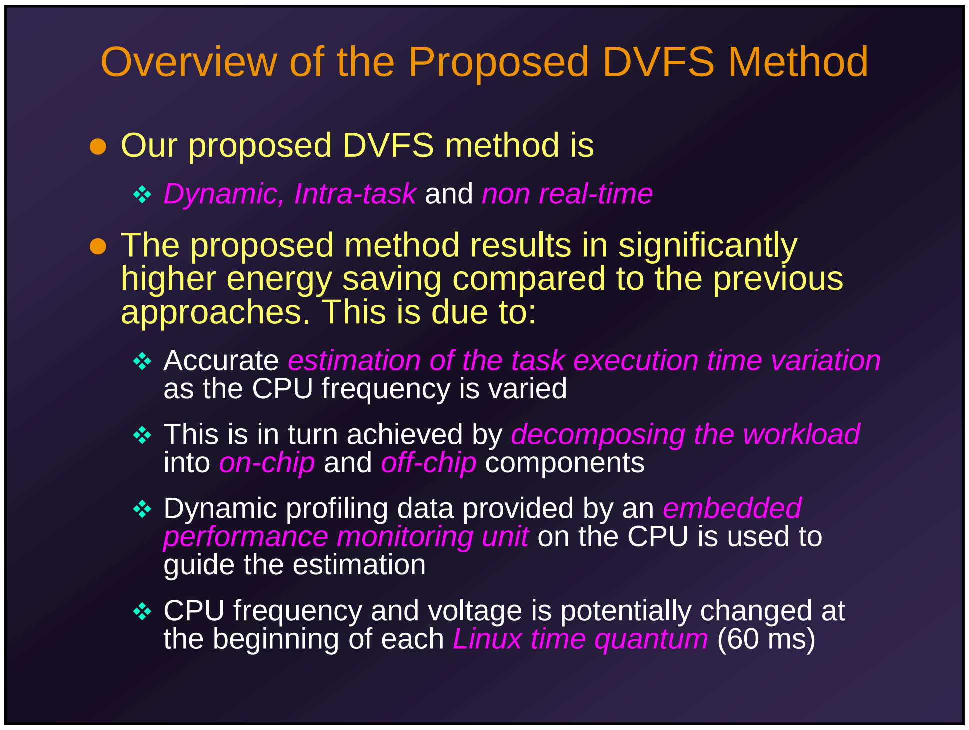
$$E = P \cdot T \propto V^2 \qquad (\text{assuming } f \propto V,\ T \propto f^{-1})$$

- Lowering V (while simultaneously and proportionately cutting f) causes a quadratic reduction in E



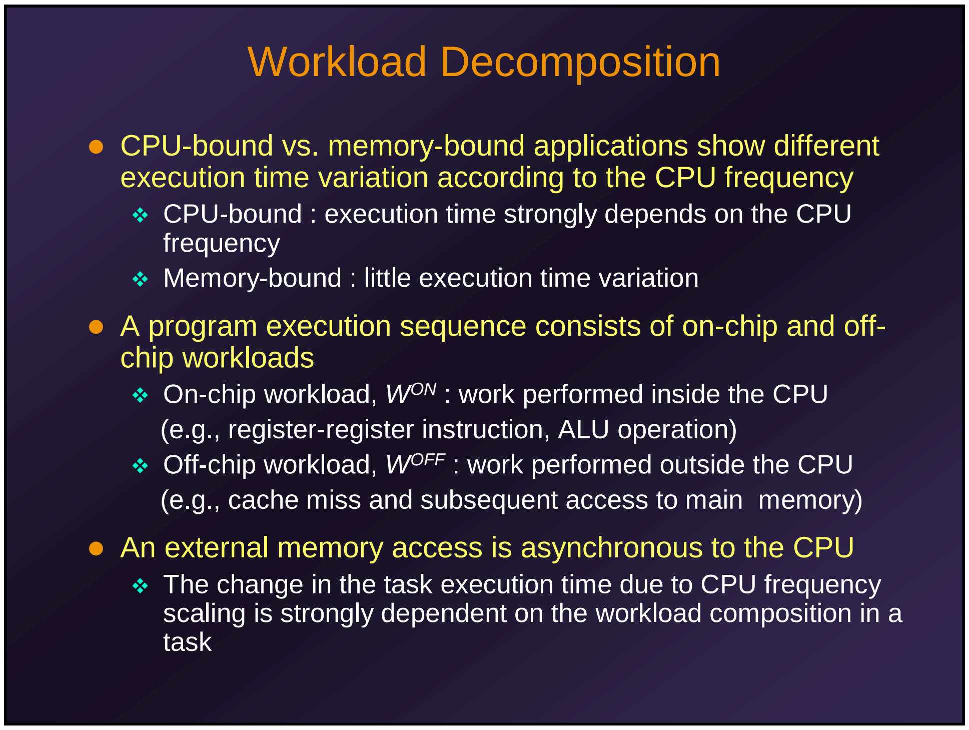
Overview of Prior Work

- DVFS techniques may be classified based on three factors:
  - *Constraint type* : real-time (critical) vs. non real-time (non critical)
  - *Scaling granularity* : inter-task (coarse) vs. intra-task (fine)
  - *Policy determination* : static (offline) vs. dynamic (online)
- The target CPU frequency is calculated as follows:
  - Given a task with workload, W, and latency constraint, D
  - Suppose:
    - W is specified as the number of CPU clock cycles needed to complete the task
    - An inverse-linear relationship between the execution time and the CPU frequency exists, i.e., $T_{task} = W/f_{cpu}$
  - f_{target} is hence calculated as W/D (Note that T_{task} = D)



Overview of the Proposed DVFS Method

- Our proposed DVFS method is
  - *Dynamic, Intra-task* and *non real-time*
- The proposed method results in significantly higher energy saving compared to the previous approaches. This is due to:
  - Accurate *estimation of the task execution time variation* as the CPU frequency is varied
  - This is in turn achieved by *decomposing the workload* into *on-chip* and *off-chip* components
  - Dynamic profiling data provided by an *embedded performance monitoring unit* on the CPU is used to guide the estimation
  - CPU frequency and voltage is potentially changed at the beginning of each *Linux time quantum* (60 ms)

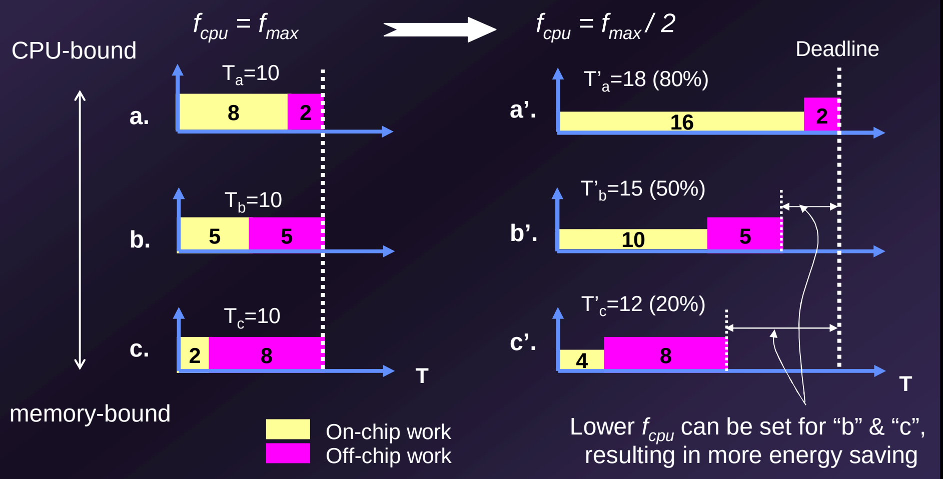


Workload Decomposition

- CPU-bound vs. memory-bound applications show different execution time variation according to the CPU frequency
  - CPU-bound : execution time strongly depends on the CPU frequency
  - Memory-bound : little execution time variation
- A program execution sequence consists of on-chip and off-chip workloads
  - On-chip workload, W^{ON} : work performed inside the CPU (e.g., register-register instruction, ALU operation)
  - Off-chip workload, W^{OFF} : work performed outside the CPU (e.g., cache miss and subsequent access to main memory)
- An external memory access is asynchronous to the CPU
  - The change in the task execution time due to CPU frequency scaling is strongly dependent on the workload composition in a task

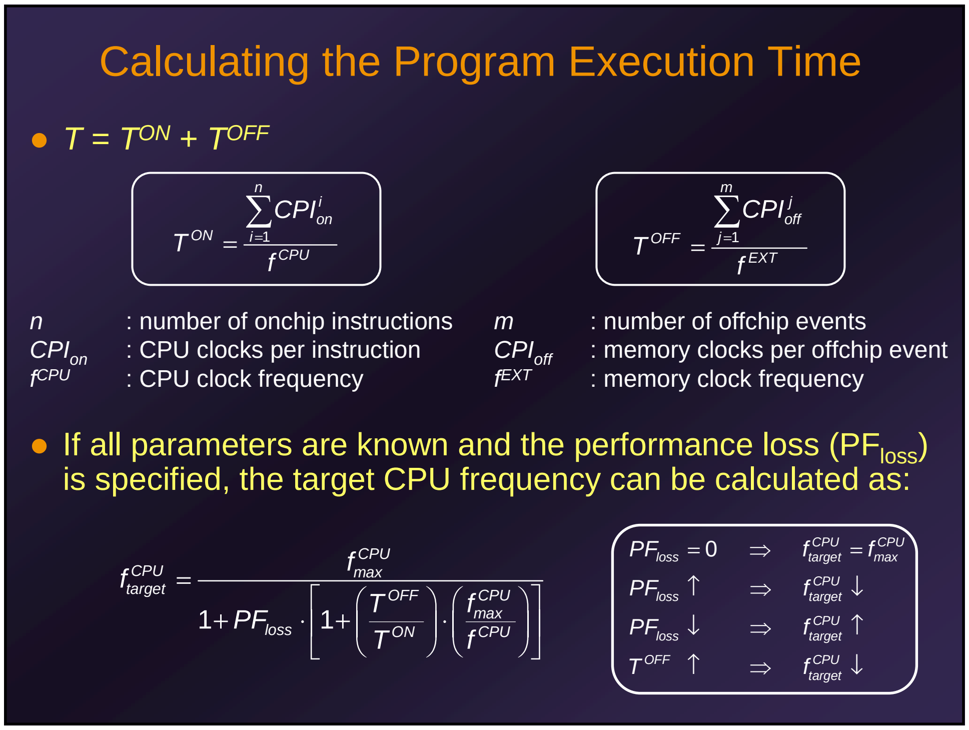
Energy Saving as a Function of Application Type

- CPU energy can be saved with lower performance loss for memory-bound application programs



Components of the Program Execution Time

- The amount of CPU and memory workloads for an application program must be determined
- Execution time of a program is the sum of the On-chip (CPU work) and the Off-chip Latency (memory work)
  - $T = T^{ON} + T^{OFF}$
- T^{ON} : varies with the CPU frequency
  - Cache hit
  - Stall due to data dependency
  - TLB hit, …
- T^{OFF}: is invariant with the CPU frequency
  - Access to external memory such as SDRAM and frame buffer memory, which is in turn due to a cache miss



Calculating the Program Execution Time

- $T = T^{ON} + T^{OFF}$

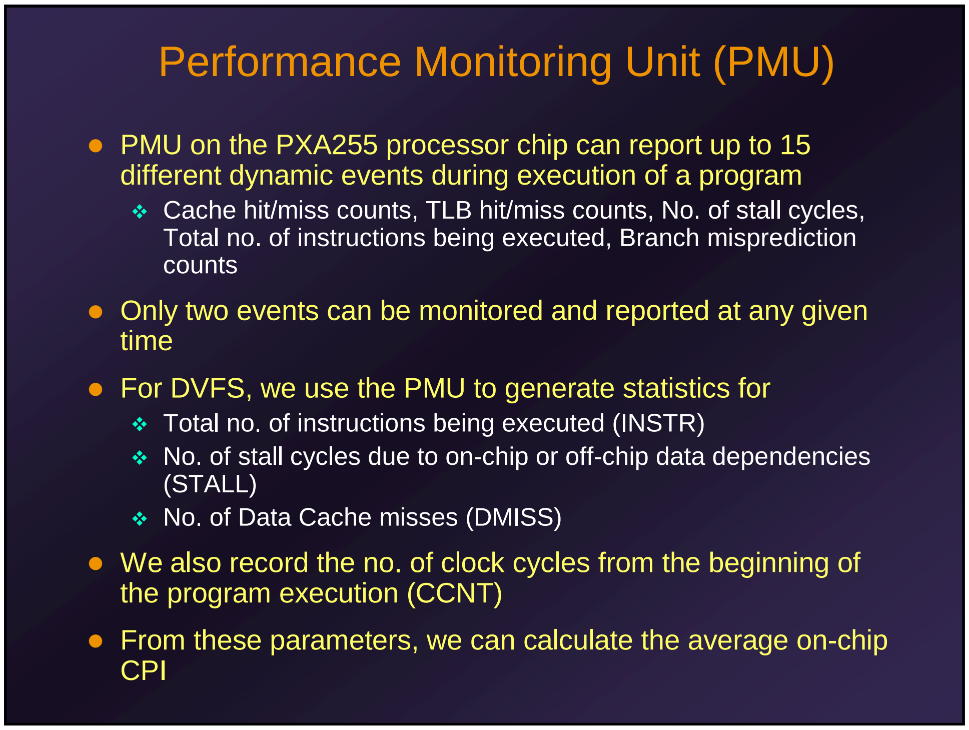
$$\boxed{T^{ON} = \frac{\sum_{i=1}^{n} CPI_{on}^{i}}{f^{CPU}}} \qquad \boxed{T^{OFF} = \frac{\sum_{j=1}^{m} CPI_{off}^{j}}{f^{EXT}}}$$

n	: number of onchip instructions	m	: number of offchip events
CPI_{on}	: CPU clocks per instruction	CPI_{off}	: memory clocks per offchip event
f^{CPU}	: CPU clock frequency	f^{EXT}	: memory clock frequency

- If all parameters are known and the performance loss (PF_{loss}) is specified, the target CPU frequency can be calculated as:

$$f_{target}^{CPU} = \frac{f_{max}^{CPU}}{1 + PF_{loss} \cdot \left[1 + \left(\frac{T^{OFF}}{T^{ON}}\right) \cdot \left(\frac{f_{max}^{CPU}}{f^{CPU}}\right)\right]}$$

$$\boxed{\begin{array}{lcl} PF_{loss} = 0 & \Rightarrow & f_{target}^{CPU} = f_{max}^{CPU} \\ PF_{loss} \uparrow & \Rightarrow & f_{target}^{CPU} \downarrow \\ PF_{loss} \downarrow & \Rightarrow & f_{target}^{CPU} \uparrow \\ T^{OFF} \uparrow & \Rightarrow & f_{target}^{CPU} \downarrow \end{array}}$$



Performance Monitoring Unit (PMU)

- PMU on the PXA255 processor chip can report up to 15 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
- Only two events can be monitored and reported at any given time
- For DVFS, we use the PMU to generate statistics for
  - Total no. of instructions being executed (INSTR)
  - No. of stall cycles due to on-chip or 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)
- From these parameters, we can calculate the average on-chip CPI

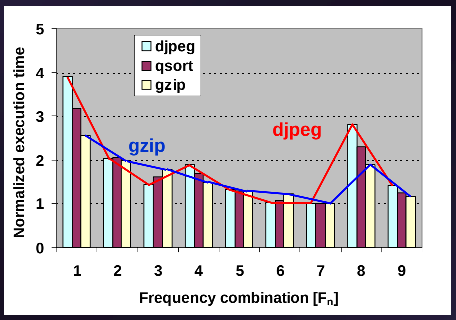
Frequency Settings in BitsyX

- PXA255 can operate from 100MHz to 400MHz, with a core supply voltage of 0.85V to 1.3V
- Nine frequency combinations (f^{CPU}, f^{INT}, f^{EXT})
- Internal bus connects the core and other functional blocks inside the CPU
- External bus is connected to SDRAM (64MB)

Freq. Set	CPU (MHz)	Internal bus (MHz)	External bus (MHz)	
F_1	100	50	100	
F_2	200	50	100	
F_3	300	50	100	
F_4	200	100	100	
F_5	300	100	100	
F_6	400	100	100	
F_7	400	200	100	**Highest Performance Setting**
F_8	133	66	133	
F_9	265	133	133	

Execution Time and Frequency Settings

- Execution time variation over different frequency combinations - “djpeg”, “qsort”, and “gzip”
  - “djpeg” is CPU-bound (strongly dependent on f^{CPU})
  - “gzip” is memory-bound (f^{INT} & f^{EXT} dependent)



Freq. Set	CPU (MHz)	Internal bus (MHz)	External bus (MHz)
F_1	100	50	100
F_2	200	50	100
F_3	300	50	100
F_4	200	100	100
F_5	300	100	100
F_6	400	100	100
F_7	400	200	100
F_8	133	66	133
F_9	265	133	133

Modeling the Execution Time in BitsyX

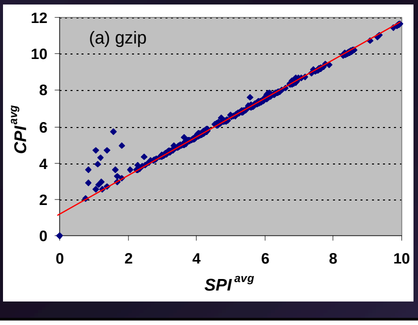
- *T^{OFF} is strongly dependent on the f^{EXT}. However, f^{INT} also affects T^{OFF}*
- Example: when a D-cache miss occurs, two operations are performed:
  - Data fetch from the external memory (f^{EXT})
  - Data transfer to the CPU core where the cache-line and destination register are updated (f^{INT})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

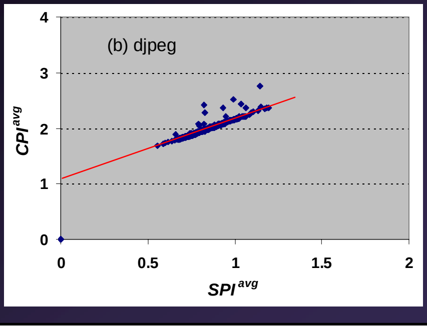
$$T^{OFF} = \frac{\alpha \cdot W^{OFF}}{f^{INT}} + \frac{(1-\alpha) \cdot W^{OFF}}{f^{EXT}}$$

- An α value of ~0.35 was obtained for tested applications
  - The error in predicting the execution time was less than 3% for all nine frequency settings

Plot of CPI vs. SPI

- We define *SPI* as ratio of the number of stall cycles to the total instruction count
  - $CPI^{avg} = CCNT / INSTR$, during a time quantum
  - $SPI^{avg} = STALL / INSTR$, during a time quantum
- A regression equation is used to model the CPI^{avg} vs. SPI^{avg} relation: $CPI^{avg} = b*SPI^{avg} + c$
- The intercept c produces the average on-chip CPI without any stall cycles, CPI_{on}^{min}

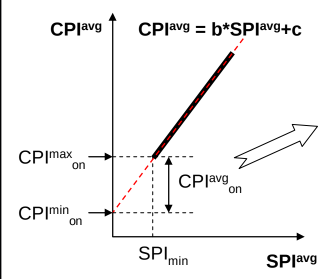




Calculating CPI_{on}^{avg}

- SPI^{avg} accounts for all stall events: $SPI^{avg} = SPI_{on}^{avg} + SPI_{off}^{avg}$
- We also have: $CPI_{on}^{avg} = CPI_{on}^{min} + SPI_{on}^{avg}$
- Let *DPI* denote the D-cache miss count per instruction, i.e.,
  - ❖ *DPI^{avg} = DMISS / INSTR,* during a quantum
- Let *dpi2spi(DPI_k)* denote a discrete function that maps DPI_k to SPI_{on}^{avg}. Then: $CPI_{on}^{avg} = CPI_{on}^{min} + dpi2spi(DPI_k)$

Note that when there are many D-cache miss events, there is a higher probability of off-chip accesses (although a D-cache miss does not always result in an off-chip access.)



CPI^{avg}_{on}(DPI)	DPI
CPI^{max}_{on}	$DPI \le K_1$
CPI^{min}_{on}+DF*(n-1)	$K_1 < DPI \le K_2$
● ● ●	● ● ●
CPI^{min}_{on}+DF*2	$K_{n-2} < DPI \le K_{n-1}$
CPI^{min}_{on}+DF*1	$K_{n-1} < DPI \le K_n$
CPI^{min}_{on}	$DPI > K_n$

DF = $(CPI^{max}_{on} - CPI^{min}_{on})/n$

K_n is constant: $K_1 < K_2 < \ldots < K_n$

Determining the Optimal Frequency Setting

- After calculating CPI_{on}^{avg} for the current quantum, *i*, the on-chip and off-chip execution times are calculated as follows:

$$T_i^{ON} = \frac{N_i \cdot CPI_{on,i}^{avg}}{f_i^{CPU}} \qquad T_i^{OFF} = T_i - T_i^{ON}$$

- Next we choose a frequency setting for the quantum i+1, F^{opt}_{i+1}, that satisfies the following equation :

$$T^{i+1}_{F^{opt}_{i+1}} \le (1 + PF_{loss}) \cdot T^{i}_{F_{max}}$$

$T^{i+1}_{F^{opt}_{i+1}}$: The expected execution time of quantum i+1 at F^{opt}_{i+1}

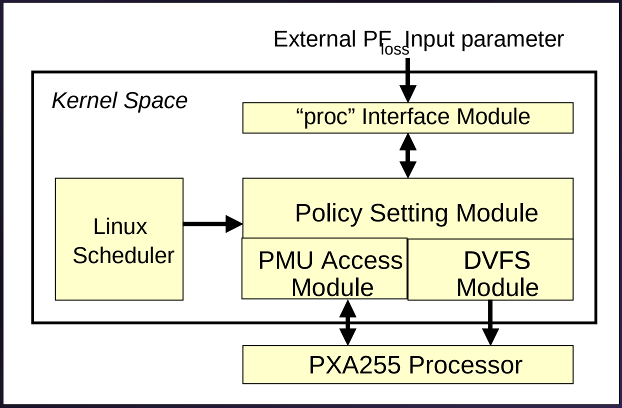
$T^{i}_{F_{max}}$: The execution time of quantum i at F_{max}

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

The Software Architecture

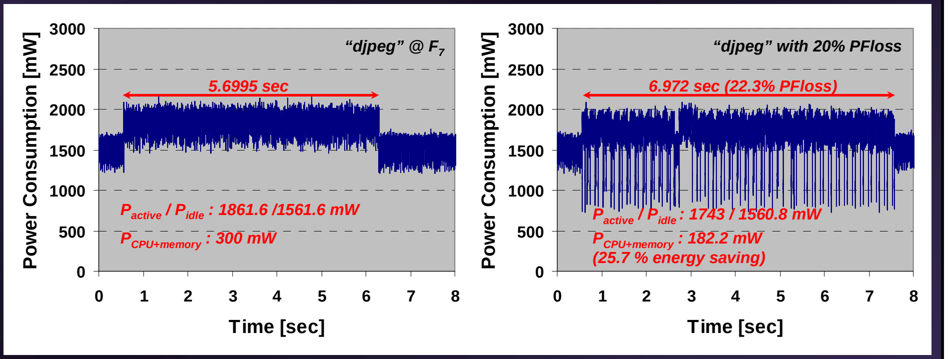
- The software architecture comprises of a proc interface module and a policy setting module tightly linked with the linux scheduler, the PMU, and the Freq. and voltage control circuitry on the BitsyX board



Energy vs. Execution Time Tradeoff

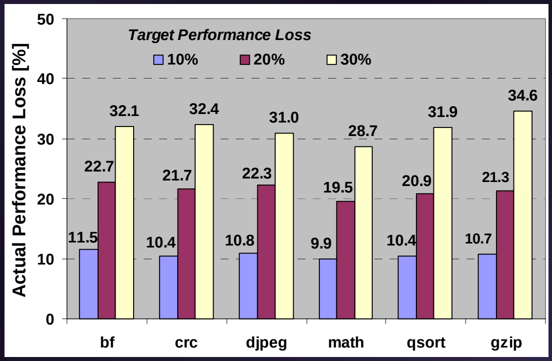
- For the *djpeg* application, a 25% total energy saving in the CPU plus memory subsystem is achieved at the cost of a 22% performance loss

without DVFS　　　　　　with DVFS



Actual vs. Target PF_{loss} Factors

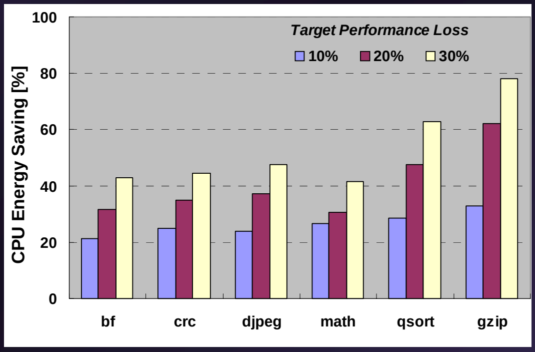
- The difference between the actual and the target performance loss factors is small (about 10% in the worst case)



CPU-bound　　　　　　Memory-bound

CPU Energy Saving Results

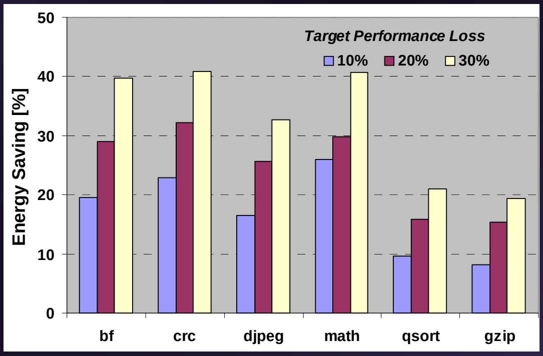
- With a 20% performance loss bound, 55% and 25% CPU energy savings were achieved for the memory-bound and the CPU-bound applications, respectively



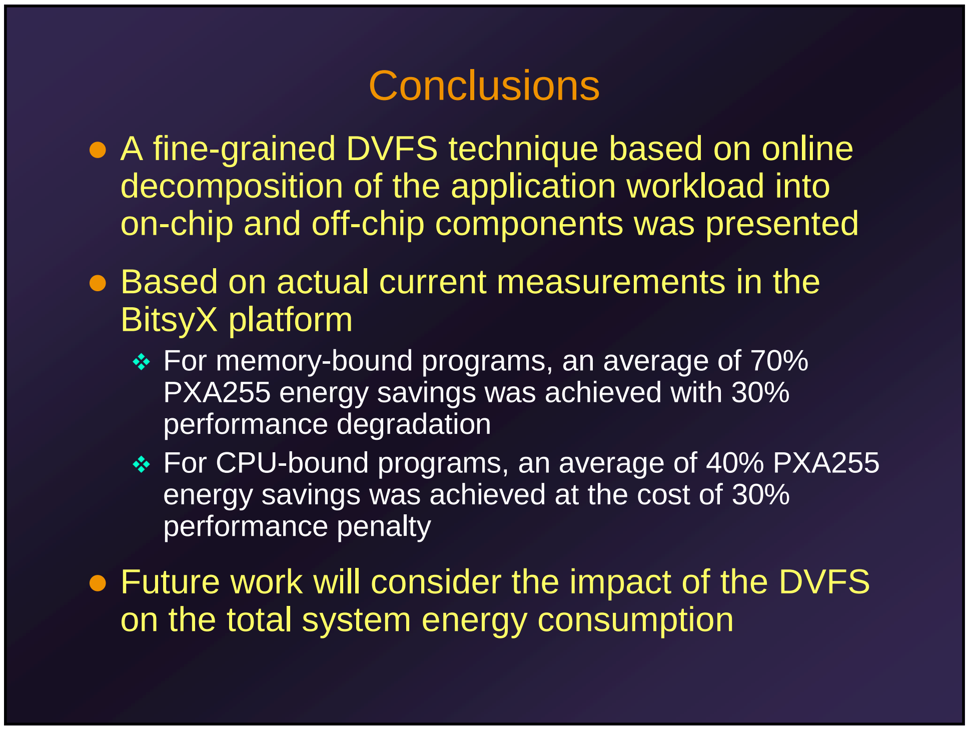
CPU-bound → Memory-bound

CPU + Memory Energy Saving Results

- With a 20% performance loss bound, 20% and 40% CPU energy savings were achieved for the memory-bound and the CPU-bound applications, respectively



CPU-bound → Memory-bound



Conclusions

- A fine-grained DVFS technique based on online decomposition of the application workload into on-chip and off-chip components was presented
- Based on actual current measurements in the BitsyX platform
  - For memory-bound programs, an average of 70% PXA255 energy savings was achieved with 30% performance degradation
  - For CPU-bound programs, an average of 40% PXA255 energy savings was achieved at the cost of 30% performance penalty
- Future work will consider the impact of the DVFS on the total system energy consumption