

Software-level Power-Aware Computing

Lecture 4

Lecture Organizations

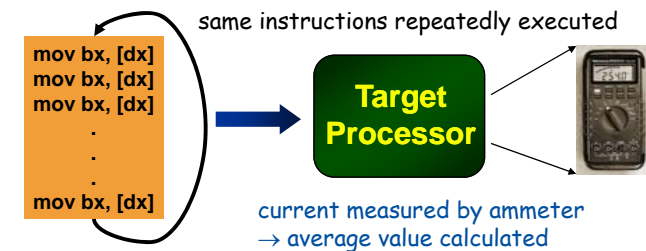
- Lecture 1:
 - Introduction to Low-power systems
 - Low-power binary encoding
 - Power-aware compiler techniques
- Lectures 2 & 3
 - Dynamic voltage scaling (DVS) techniques
 - OS-level DVS: Inter-Task DVS
 - Compiler-level DVS: Intra-Task DVS
 - Application-level DVS
 - Dynamic power management
- Lecture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

Software Power Estimation & Optimization

- Modeling-based Approaches
 - Instruction-level power modeling (V. Tiwari)
 - Employs *base energy cost* of each instruction
 - Instruction-level analysis and optimization
 - Component-based power modeling
 - Wattch, SimPower
- Measurement-based Approaches
 - SES : SNU Energy Scanner
 - Cycle-level power measurement on the target board
 - PowerScope (J. Flinn)
 - Function-level analysis w/ DMM
 - ePRO : energy PRofiler and Optimizer
 - Function-level analysis and optimization w/ DAQ

Instruction-level Power Estimation (1)

- Base energy cost



- For variations in base energy costs due to operands and addresses, average base cost values are employed

Instruction-level Power Estimation (2)

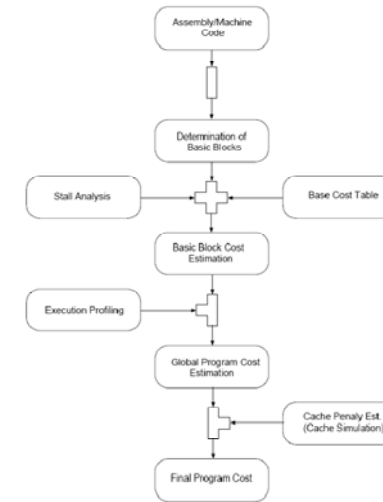
- Inter-Instruction Effects
 - Effects of circuit state : switching activity in a circuit
 - Avg. overhead through extensive experiments between pairs of instructions
 - Effect of resource constraints : pipeline stall and write buffer stalls
 - Avg. energy cost of each stall experimentally determined
 - Effect of cache misses
 - Avg. energy penalty of cache miss cycles

$$E_p = \sum_i (B_i \times N_i) + \sum_{i,j} (O_{i,j} \times N_{i,j}) + \sum_k E_k$$

E_p : average energy cost for a program $O_{i,j}$: the circuit state overhead
 B_i : the base cost of each instruction $N_{i,j}$: the no. of times the pair is executed
 N_i : the number of times executed E_k : other instruction effects

Instruction-level Power Estimation (3)

- Energy estimation framework for a program



Instruction-level Power Optimization

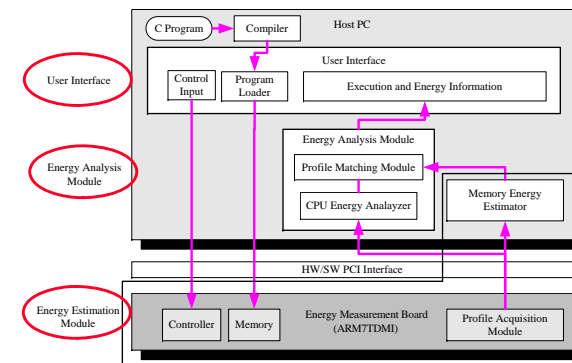
- Instruction Reordering
 - A technique to reduce the circuit state overhead
 - Instructions are scheduled in order to minimize the estimated switching in the control path
 - For 486DX, energy saving only up to 2%, but for a DSP up to 33.1%
- Energy cost driven code generation
 - Instructions with memory operands have much higher currents than those with register operands
 - Better utilization like optimal global register allocation of temporaries and frequently used variables

RESULTS OF ENERGY OPTIMIZATION OF SORT AND CIRCLE

Program	hlcc.asm	hht1.asm	hht2.asm	hht3.asm
Avg. Current (mA)	525.7	534.2	507.6	486.6
Execution Time (μsec)	11.02	9.37	8.73	7.07
Energy (10 ⁻⁶ J)	19.12	16.52	14.62	11.35
Program	clcc.asm	cht1.asm	cht2.asm	cht3.asm
Avg. Current (mA)	530.2	527.9	516.3	514.8
Execution Time (μsec)	7.18	5.88	5.08	4.93
Energy (10 ⁻⁶ J)	12.56	10.24	8.65	8.37

SES : SNU Energy Scanner (1)

- Overall Structure

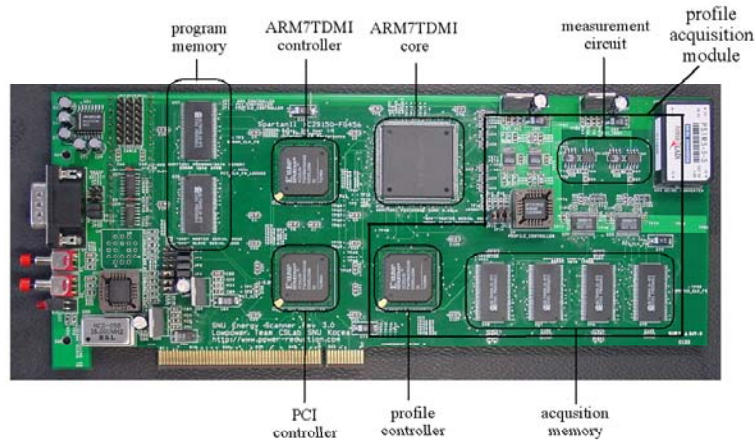


- Summary

- On-board, cycle-level power measurement
- Source code related energy analysis

SES : SNU Energy Scanner (2)

- Energy Measurement H/W Module



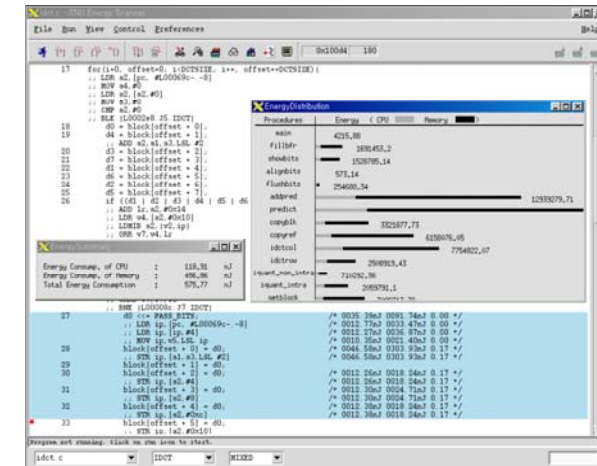
Low Power SW.4

9

J. Kim/SNU

SES : SNU Energy Scanner (3)

- Energy Analyzer GUI



Low Power SW.4

10

J. Kim/SNU

SES : SNU Energy Scanner (4)

- Pros
 - No additional measurement device (like DMM or DAQ) necessary
 - Cycle-level accuracy and timeliness
 - Source code related energy analysis
 - C program function or instruction level
- Cons
 - No portability
 - For each processor, new hardware and program are necessary
 - time, cost, and effort!!
 - No exact performance-energy correlation
 - Performance is not measured

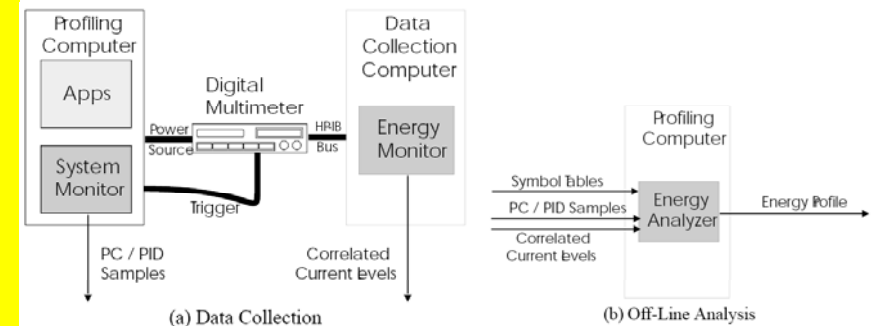
Low Power SW.4

11

J. Kim/SNU

PowerScope (1)

- Overall Structure



- Summary

- Power measurement w/ external device (DMM)
- Source code related energy analysis

Low Power SW.4

12

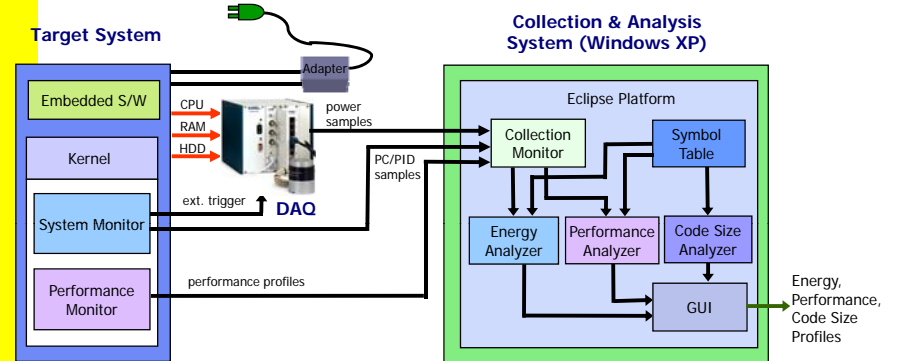
J. Kim/SNU

Powerscope (2)

- Pros
 - Portability
 - Employs Linux LKM (Loadable Kernel Module) and DMM
 - Moderate accuracy but fast measurement
 - Source code related energy analysis
 - C program function level
- Cons
 - Overhead
 - Sampling trigger feedback between target b'd and DMM
 - varying interrupt handling time
 - Long profile function path
 - No performance–energy correlation
 - Performance is not measured

ePRO : energy PРоfiler and Optimizer (1)

Overall Structure



Summary

- Power measurement w/ external device (DAQ)
- Source code related energy, performance, and code size analysis
- Automatic compiler–level optimization

ePRO : energy PРоfiler and Optimizer (2)

- Overview
 - Automatized tool which analyzes and optimizes software energy and performance based on measurement
- Function details
 - Performance Analysis
 - Function–level performance indices
 - Energy Analysis
 - Function–level energy consumption
 - Device–level energy consumption
 - Energy Optimization
 - Energy–optimal compiler option selection
 - Integrated Development Environment (IDE)
 - Plug–in of Eclipse

ePRO : energy PРоfiler and Optimizer (3)

Performance Analysis

- Using XScale processor's PMU
- CPI (Cycles Per Instruction) I–cache/D–cache efficiency
- Instruction fetch latency
- Data/bus request buffer : D–cache buffer stall
- Stall/writeback statistics
- I–TLB/D–TLB efficiency

Function	Source	Code...	Energy Consu...	Energy Ratio	Execution Tim...	I-Cache Miss...	D-Cache Miss...	CPI
divector	nrutil.c:60	64	0.00	0.00	7.82	0.1540	2.6846	5.566
fnrft	nrutil.c:59	6172	9,033.39	0.0022	6,585.05	0.0022	0.0128	1,786
fnrft	nrutil.c:486	536	7,741.11	0.004	832.95	0.004	0.0085	1,681
free_dmatrix	nrutil.c:132	42	0.00	0.00	0.1745	0.0000	0.0000	3,001
free_dvector	nrutil.c:123	29	0.00	0.00	15.20	0.1624	4,9190	10,725
free_dvector	nrutil.c:98	29	0.00	0.00	4.4736	1.8555	9.161	9,161
free_dvector	nrutil.c:106	29	0.00	0.00	0.1905	0.0000	0.0000	2,57
free_dvector	nrutil.c:114	29	0.00	0.00	5.4490	5.5225	19.34	19,34
golden	nrutil.c:95	516	0.00	0.00	11.20	0.0000	2.0906	2,35
ivector	nrutil.c:33	64	0.00	0.00	0.1754	1.6125	3.21	3,21
ivector	nrutil.c:35	64	0.00	0.00	7.4543	6.5760	18.17	18,17
main	main_jrnl.c:19	516	0.00	0.00	76.79	0.0000	0.2548	2,70
memor	nrutil.c:11	100	0.00	0.00	0.0000	0.0000	0.0000	0,00
phlfn	nrutil.c:638	488	55,103.28	0.0014	6,116.69	0.0014	0.5121	1,78
phlgr	nrutil.c:521	112	31.74	0.0023	4.37	0.0023	2.2641	2,92
total	total	760	1,164.87	0.0000	142.00	0.0000	1,78	1,78

ePRO : energy PROfiler and Optimizer (4)

- Function-level Energy Consumption Analysis

Function	Source	Code	Energy Consu...	Energy Ratio (%)
_TOTAL			74,545.70	
amphi	fmft.c:804	164	42.96	
bracket	fmft.c:510	304	232.16	
dindex	fmft.c:717	717	0.00	
dmatrix	nrutil.c:72	192	0.00	
dsort	fmft.c:569	400	0.00	
dvector	nrutil.c:90	64	0.00	
fmft	fmft.c:59	610	9,033.28	
four1	fmft.c:466	536	7,741.11	
free_dmatrix	nrutil.c:132	52	0.00	
free_dvector	nrutil.c:123	20	0.00	
free_ivector	nrutil.c:98	20	0.00	
free_ivector	nrutil.c:106	20	0.00	
free_ivector	nrutil.c:114	20	0.00	
golden	fmft.c:565	516	0.00	
ivector	nrutil.c:93	64	0.00	

- Device-level Energy Consumption Analysis
 - CPU, RAM, FLASH, HDD, etc

Device Name	Energy Consumption (mJ)	Energy Ratio	Percentage (%)
_TOTAL	678.84		100.00
CPU	123.90		18.25
Flash	43.60		6.42
HDD	410.03		60.40
RAM	101.31		14.92

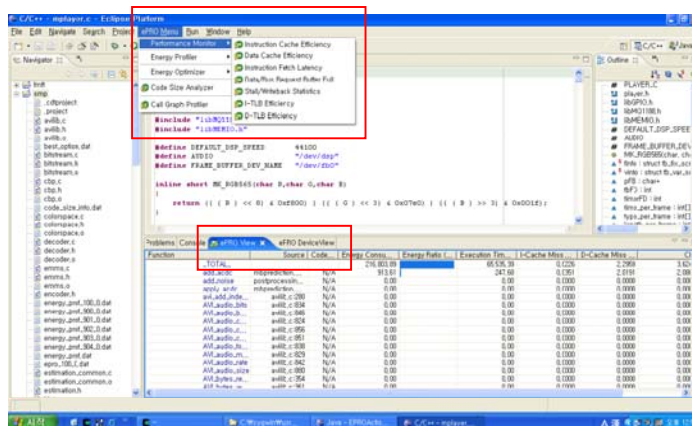
ePRO : energy PROfiler and Optimizer (5)

- Energy Optimization
 - CL-OSE (Compiler-Level Optimal Space Exploration) : Selects the energy-optimal options time-efficiently for the target program among the all the available compiler options

Source	OPT OPTION 01 (mJ)	OPT OPTION 02 (mJ)	OPT OPTION 03 (mJ)	BEST OPTION (-O2) (mJ)
_TOTAL	238,421.90	66,683.56	68,565.36	66,683.56
amphi	fmft.c:604	0.00	0.00	0.00
bracket	fmft.c:518	156.05	93.32	93.32
dindex	fmft.c:717	0.00	0.00	0.00
dmatrix	nrutil.c:72	0.00	0.00	0.00
dsort	fmft.c:569	0.00	0.00	0.00
dvector	nrutil.c:90	0.00	0.00	0.00
fmft	fmft.c:59	182,843.75	12,408.76	12,408.76
four1	fmft.c:466	6,455.51	6,739.21	6,739.21
free_dmatrix	nrutil.c:132	0.00	0.00	0.00
free_dvector	nrutil.c:123	0.00	0.00	0.00
free_ivector	nrutil.c:98	0.00	0.00	0.00
free_ivector	nrutil.c:106	0.00	0.00	0.00
free_ivector	nrutil.c:114	0.00	0.00	0.00
golden	fmft.c:565	31.13	31.73	207.27
ivector	nrutil.c:93	0.00	0.00	0.00

ePRO : energy PROfiler and Optimizer (6)

- Integrated Development Environment
 - Employs Eclipse's plug-in function

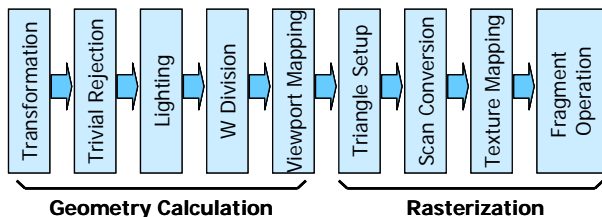


ePRO : energy PROfiler and Optimizer (7)

- Pros
 - Portability
 - Employs Linux LKM and DAQ assembly
 - Program function-level energy, performance, and code size analysis
 - Automatized compiler-level energy optimization
- Cons
 - Overhead
 - System behavior sampling overhead
 - Limited to a processor with PMU (Performance Monitoring Unit) : e.g. XScale
 - No support for multiple processes till now

3D Graphics Pipeline

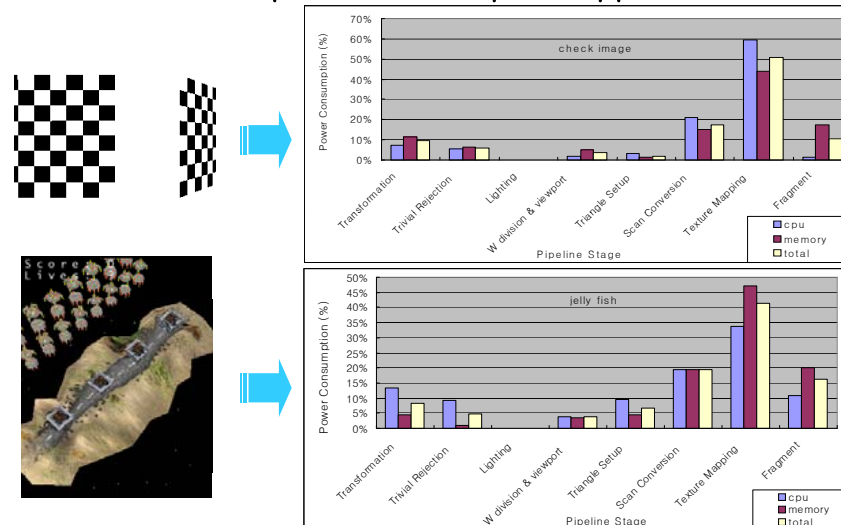
- 3D Graphics Pipeline
 - Geometry Calculation
 - Calculation of geometric data of objects
 - Rasterization
 - Converting an object on a screen



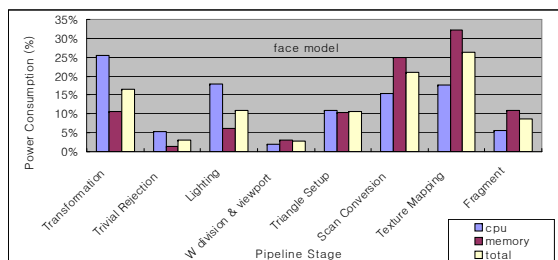
<3D Graphics Pipeline>

Power Breakdown of 3D Graphics

- Power Consumption of 3D Graphics Application



Face model

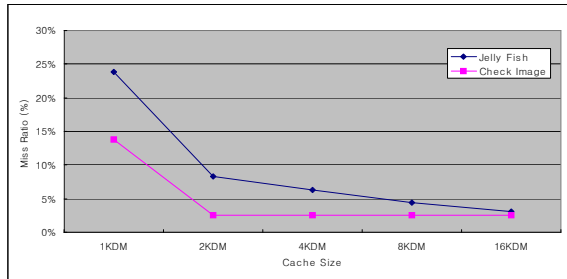


A Low-Power Texture Mapping (1)

- Previous Work
 - “A Low-Power Content-Adaptive Texture Mapping Architecture for Real-Time 3D Graphics”, Jeongseon Euh et al, PACS’02.
 - Adaptive texture mapping
 - based on a model of human visual perception (HVP)
 - DVS is applied to the interpolation block
 - “Trading Efficiency for Energy in a Texture Cache Architecture”, Iosif Antochi et al, MPCs’02
 - Mobile devices cannot afford large texture cache
 - Due to gate count limitation and low power consumption
 - 128~512 bytes texture cache between the graphics accelerator and texture memory

A Low-Power Texture Mapping (2)

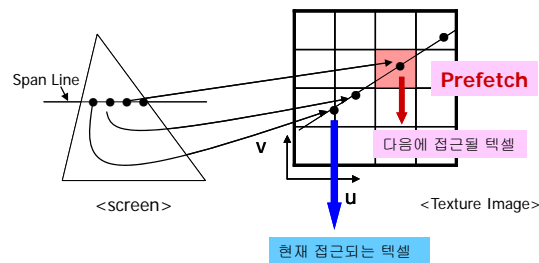
- A Low-Power Texture Mapping Technique for Mobile 3D Graphics
 - A small texture cache can increase the miss ratio
 - The technique to preserve performance is needed
 - Prefetching
 - Victim cache



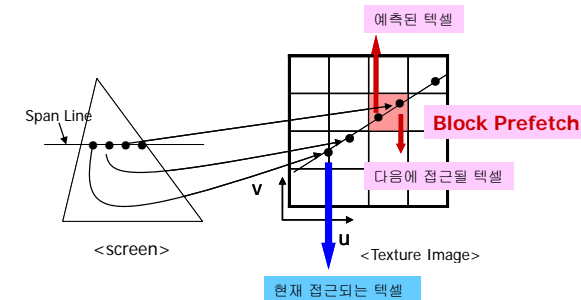
A Low-Power Texture Mapping (3)

- Prefetch techniques
 - Technique 1: Prediction of next texels
 - Division is required due to “perspective correction”
 - Technique 2: Prediction of next blocks
 - Assuming that derivatives are not changed
 - Division is eliminated
 - Technique 3: Prediction of next blocks based on direction of texture map access
 - Simple, but less exact than technique 2
- A small fully associative prefetch buffer is used

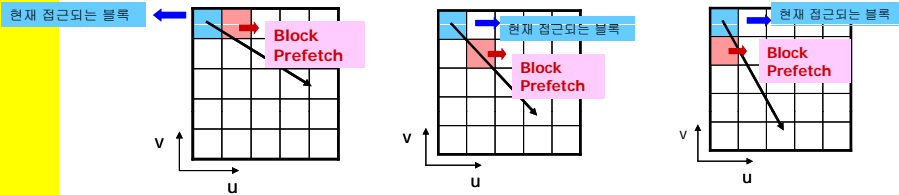
Technique 1



Technique 2



Technique 3

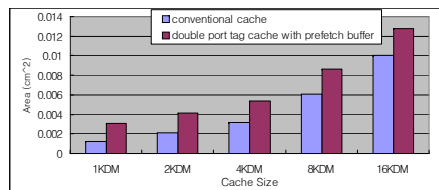


A Low-Power Texture Mapping (4)

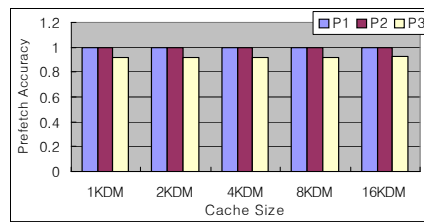
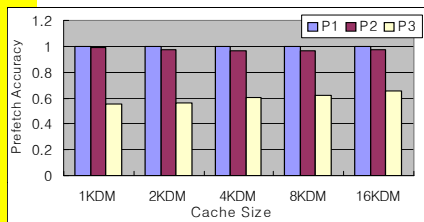
- Using victim cache
 - Sizes of texture images are powers of two
 - Conflict misses can occur between blocks
 - Especially in the small texture cache
 - Blocks are reused in processing of next spanline
 - Victim cache can reduce conflict misses
 - Prefetch buffer performs as the victim cache
 - Evicted blocks are moved into the prefetch buffer

A Low-Power Texture Mapping (5)

- Experimental Results
 - Area Reduction

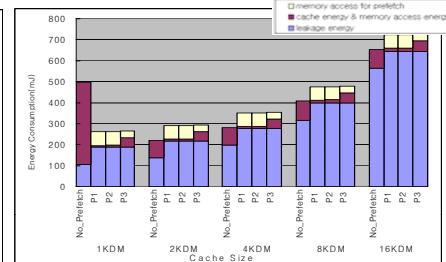
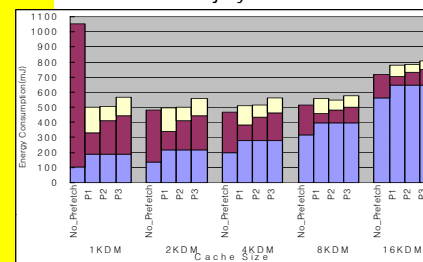
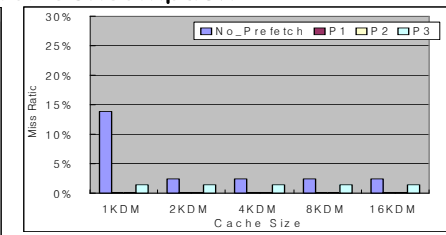
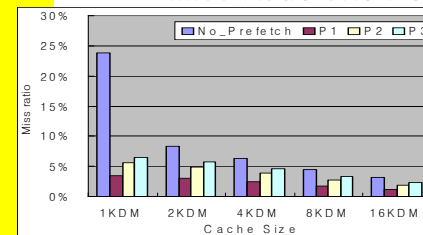


- Prefetch Accuracy



A Low-Power Texture Mapping (6)

- Miss Ratio and Power Consumption



Researches on 3D Graphics (1)

- “An Effective Pixel Rasterization Pipeline Architecture for 3D Rendering Processors”, Woo-Chan Park et al, IEEE Transactions on Computers `03
 - Avoid unnecessary texture mapping for obscured fragments
 - Reduce the miss penalties of the pixel cache by prefetching scheme
- “Design and Implementation of Low-Power 3D Graphics SoC for Mobile Multimedia Applications”, Ramchan Woo, PHD thesis, KAIST `04
 - Implementing full-3D pipeline with texture mapping and special effects

Researches on 3D Graphics (2)

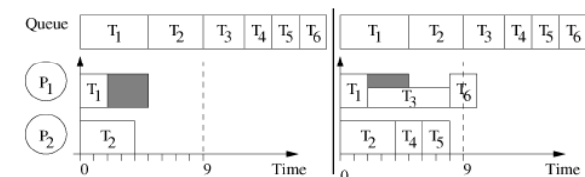
- “GaalBench: A 3D Graphics Benchmark Suite for Mobile Phones”, Iosif Antochi et al. LCTES`04
 - A set of 3D graphics workloads representative for mobile devices
- “Power-Aware 3D Computer Graphics Rendering”, Jeongseon Euh, Journal of VLSI Signal Processing`05
 - Low power system based on Approximate Graphics Rendering (AGR)
 - Power savings are examined for stages
 - Shading
 - Texture mapping

Low-Power Techniques for Multiprocessors

- DVS techniques for Multiprocessors
 - Slack reclamation
 - Condition-aware scheduling
 - dist-PID
- Power-Aware Parallelism Optimization
 - Static & Dynamic optimizing
- Local Memory Management

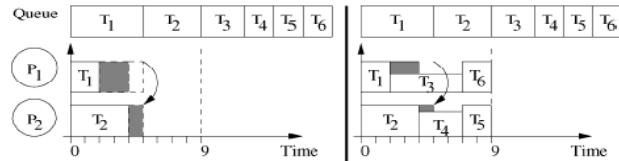
Slack Reclamation (1)

- Problem definition
 - Greedy slack reclamation
 - Any slack is used to reduce the speed of next task on same processor
 - It cannot guarantee deadline
 - Example: 6 tasks in 2 processors
 - $\Gamma = \{T_i(\text{WCET}, \text{AET}) \mid T_1(5,2), T_2(4,4), T_3(3,3), T_4(2,2), T_5(2,2), T_6(2,2)\}$, Deadline=9
 - T_3 uses up its time, T_6 misses the deadline



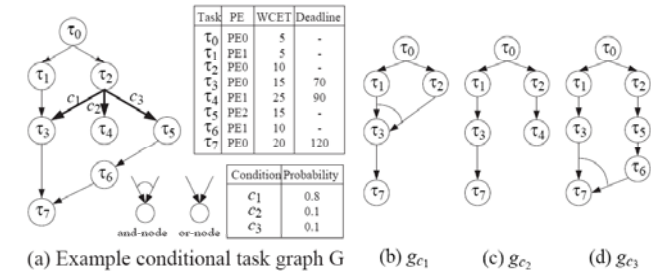
Slack Reclamation (2)

- Shared slack reclamation [Zhu03]
 - Share the slack with other processors
 - Split slack into multiple parts
 - Slack sharing example (see figure (b))
 - Slack1: Two time units before T_2 's finish time (based on T_2 's WCET)
 - Slack2: One time units after T_2 's finish time
 - Share slack2 with P_2
 - All tasks meet deadlines



Condition-Aware Scheduling (1)

- Task scheduling for conditional task graph
 - Conditional Task Graph (CTG)
 - Various task sequences depending on the conditions
 - Require power-aware scheduling technique considering conditions



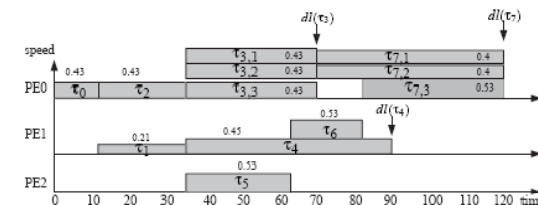
Condition-Aware Scheduling (2)

- Condition-Aware scheduling [Shin03]
 - Step 1: Task ordering
 - Use the schedule table: <start time, clock speed>
 - Depending the condition value, each task has different start time and clock speed

task \ condition	true	c_1	c_2	c_3			
τ_0	0	0.5					
τ_1	10	0.25					
τ_2	10	0.5					
τ_3		30	0.39	30	0.39	30	0.38
τ_4			30	0.42			
τ_5					30	0.5	
τ_6					60	0.5	
τ_7		68.6	0.39	68.6	0.39	80	0.5

Condition-Aware Scheduling (3)

- Condition-Aware scheduling [Shin03]
 - Step 2: Task stretching
 - Use probabilities of conditions from profile information
 - Minimize $\sum E(\tau) \text{Prob}(\tau)$
 - Optimize for high probability conditions
 - $c_1 \gg c_2, c_3$



DVFS in MPSoC (1)

- Local-DVFS
 - Decide the frequency of the each processor only using the local information.
 - Do not use the information of the other processors.
 - Higher frequency as more tasks in the task queue.
- Limitations of the local-DVFS
 - If a processor is executed with lower frequency, it can hurt the performance of the other processor because of the dependency.

DVFS in MPSoC (2)

- dist-DVFS [Juang05]
 - Decide the frequency of the each processor using the global information.
- Operation steps
 - Estimate the future task queue occupancy
 - Identify the critical-path-tasks (with the highest queue occupancy)
 - Decide the frequency of the each processor not hurting the performance of the critical-path-tasks

Optimizing Parallelism (2)

- The number of processors that generate the best execution time for each loop nest

Benchmark Name	N1	N2	N3	N4	N5	N6	N7	N8	N9
3step-log	1	1	5						
adi	4	5							
app	1	1	1						
biscm	1	1	2	4					
btrix	2	1	7	6	1	3	8		
eflux	2	3							
full-search	2	2	6						
hier	1	1	3	3	2	1	5		
lms	2	1	2	2					
n-real-updates	4	4	4						
parallel-hier	3	3	1	1	2				
tomcat	2	1	3	1	2	4	1	8	2
tsf	1	7	2	4					

Loop nest

Number of processors for the best execution time

- Using only a small subset of processors out of 8 processors
- This is a strong motivation for shutting down unused processors.

Optimizing Parallelism (3)

- Designing an effective parallelization strategy for an on-chip multiprocessor
 - Mechanism
 - Dynamic approach
 - The number of processors for each loop nest is decided at run time.
 - Static approach
 - The number of processors for each loop nest is decided at compile time.
 - Policy
 - Criterion to decide the number of processors
 - Execution time, energy and so on.

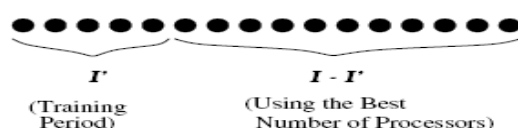
Optimizing Parallelism (4)

- Procedure
 - Determine the number of processors from mechanism and policy
 - Insert activation / deactivation call in the code
 - Optimize the code
- Optimization
 - Current active/idle status of processors is maintained as much as possible
 - To minimize overhead from on/off
 - We have to pre-activate the processors,
 - If the processor will be used in next loop
 - Not to hurt the performance

Runtime Code Parallelization (1)

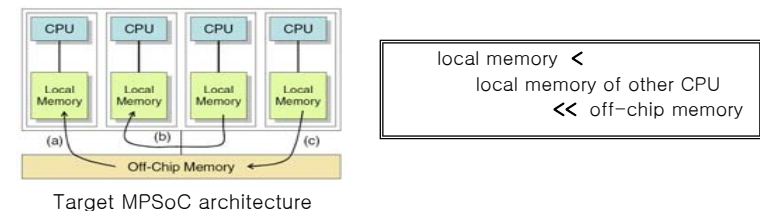
- A run-time strategy for determining the best number of processors to use [Kandemir03]
 - Dynamic mechanism
 - To minimize energy and execution time
 - Need some help from H/W and compiler.

Run-time Code Parallelization (2)

- Parallelization based on training
 - Each dot represents an iteration
- 
- Training period
 - Find the optimal number of processors
 - Using the number of processors determined
- Extra Optimization
 - Minimize training iteration based on history.
 - Utilize the past history, avoid redundant training.

Local Memory Management (1)

- Latency of memory access



- Frequent off-chip memory access can be very costly from both performance and energy perspectives
- Propose local memory management scheme for low cost [Chen05]

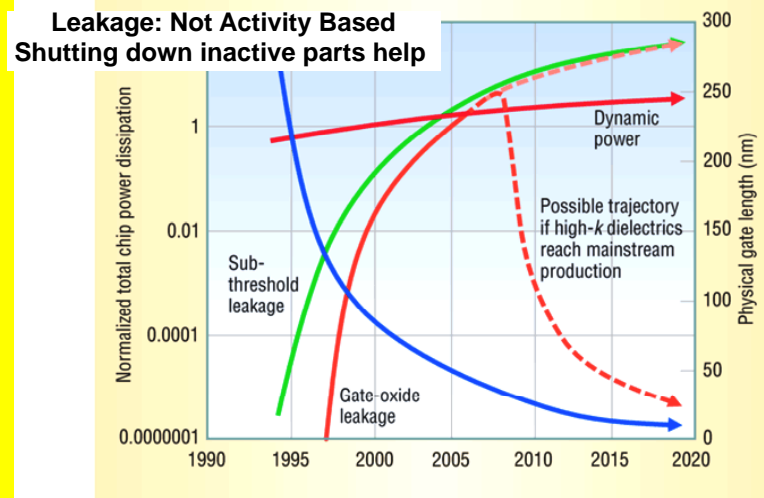
Local Memory Management (2)

- Access pattern of the data block is analyzed by compiler
- Software-managed memory is used
- When a data block is stored in the local memory of the processor,
 - Even though the data block is predicted not to be used any more by the processor,
 - If the data block is predicted to be used by another processor, keep the data block in the local memory.

References

- [Chen05] Guilin Chen, Guangyu Chen, Ozcan Ozturk and Mahmut Kandemir, "Exploiting Inter-Processor Data Sharing for Improving Behavior of Multi-Processor SoCs", In Proc. of Annual Symposium on VLSI, 2005.
- [Juang05] Philo Juang and Qiang Wu, "Coordinated, Distributed, Formal Energy Management of Chip Multiprocessors", In Proc. of ISLPED, 2005.
- [Kadayif05] Ismail Kadayif, Mahmut Kandemir, Guilin Chen and Ozcan Ozturk, "Optimizing Array-Intensive Applications for On-Chip Multiprocessors", IEEE Trans. on Parallel and Distributed Systems, 2005.
- [Kandemir03] M Kandemir, W Zhang, M Karakoy, "Runtime code parallelization for on-chip multiprocessors", In Proc. of DATE, 2003.
- [Shin03] Dongkun Shin and Jihong Kim, "Power-Aware Scheduling of Conditional Task Graphs in Real-Time Multiprocessor Systems", In Proc. of ISLPED, 2005.
- [Zhu03] Dakai Zhu, Rami Melhem, and Bruce R. Childers, "Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multiprocessor Real-Time Systems", IEEE Trans. on Parallel and Distributed Systems, Vol.14, No.7, July 2003.

Leakage Current



(source: Kim et al., IEEE Computer, Dec, 2003)

Subthreshold Leakage, I_{sub}

$$I_{sub} \sim e^{(-Vt/Va)} (1 - e^{(-V/Va)})$$

where V_a is the thermal voltage

- How to reduce I_{sub}
 - Turn off the supply voltage
(-) loss of state
 - Increase the threshold voltage
(-) loss of performance

Leakage Power Reduction

- State-Destructive vs State-Preserving
- Application-Sensitive vs. Application-Insensitive

Dynamic Resizing of Instruction Cache

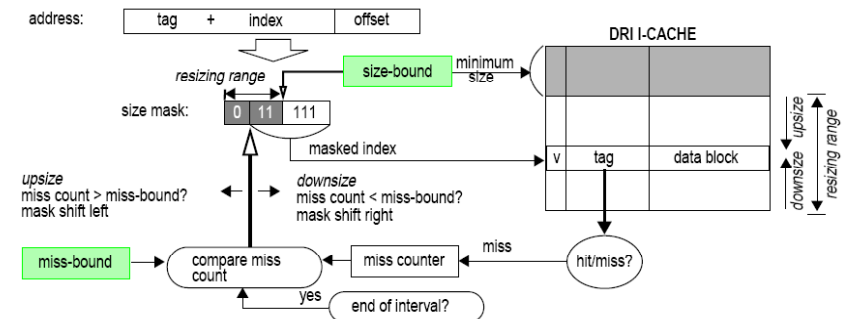
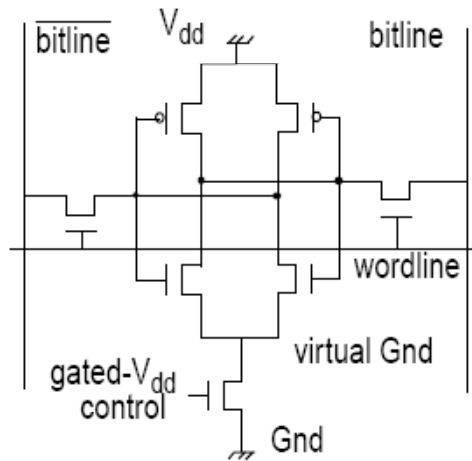


FIGURE 1: A DRI i-cache's anatomy.

[Powell, ISLPED00]

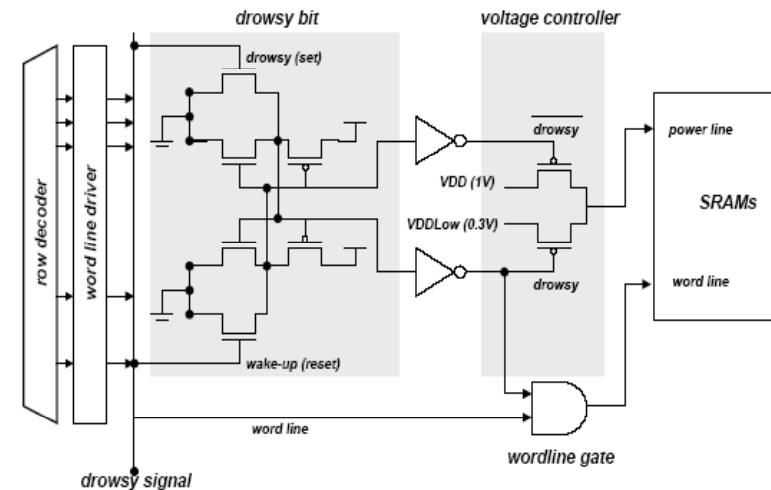
Gated-V_{dd}



- State Destructive, Application Insensitive

Drowsy Cache

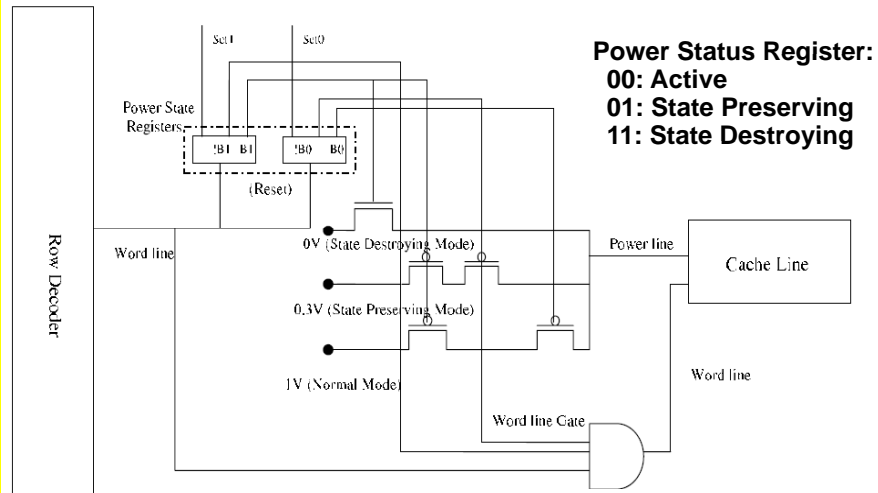
[Flautner, 2002]



- State Preserving, Application Insensitive

Compiler-Directed Approach

[Zhang, MICRO-35]



Power Status Register:
00: Active
01: State Preserving
11: State Destroying

Two special instructions for power state changes