Energy-Efficient Embedded System Design at 90nm and Below : A System-Level Perspective

Ishihara, Tohru
System LSI Research Center, Kyushu University

Energy-Efficient Embedded System Design at 90nm and Below
~ A System-Level Perspective ~

Tohru Ishihara

System LSI Research Center
Kyushu University

Agenda

- Introduction
- Software-Level Energy Characterization
- Process-Variation Aware Compilation

What is Embedded System?

- A combination of computer hardware and software
- A specialized computer system which is dedicated to a specific task

⬇

- Performance is not necessarily very important
- Can spent much time for compiler optimization
- The number of chips produced is 1/100 of general purpose processors

Mask Costs at 90nm & Below

- Mask cost doubles every 2 years

Year of Production	’05	’06	’07	’08	’09	’10	’11	’12	’13
MPU/ASIC Metal1 ½ Pitch (nm)	90	78	68	59	52	45	40	36	32
Mask Cost ($m)	1.5	2.2	3.0	4.5	6.0	9.0	12.0	18.0	24.0

ITRS 2005 prediction

- Suppose the number of chips per a mask set is 100,000

The mask cost per chip is more than $30

Fujitsu’s embedded processor

Chip price $27

Solutions

- Field Programmable Devices
  - Energy consumption is still high
  - 10x or 100x of ASIC chips
- Software and processors
  - The energy consumption is highly depending on the software running on the target processor
  - Software programmers do not pay attention to energy reduction

Agenda

- Introduction
- Software-Level Energy Characterization
- Process-Variation Aware Compilation

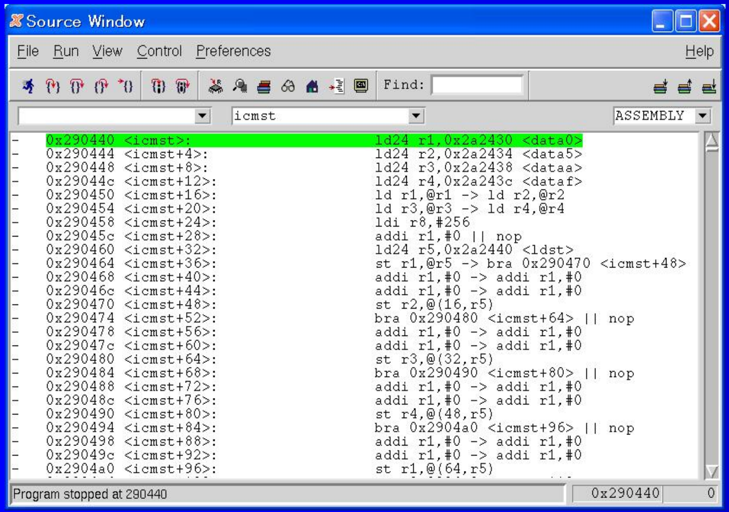
Motivation

- Energy consumption of embedded systems depend on the behavior of the software running on the hardware
- Most software programmers pay less attention to the energy issue than hardware designers do
- Software-level energy analysis is needed for reducing energy consumption of embedded systems

Our Goal

- Energy analysis framework which can be used in a software design phase

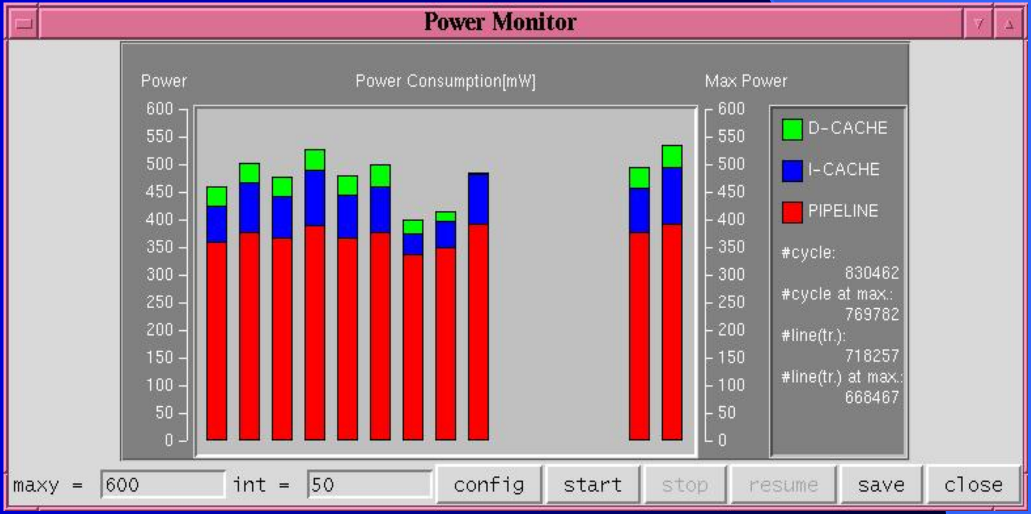
Work in cooperation with GDB



```
-     0x290440 <icmst>:                ld24 r1,0x2a2430 <data0>
-     0x290444 <icmst+4>:              ld24 r2,0x2a2434 <data5>
-     0x290448 <icmst+8>:              ld24 r3,0x2a2438 <dataa>
-     0x29044c <icmst+12>:             ld24 r4,0x2a243c <dataf>
-     0x290450 <icmst+16>:             ld r1,@r1 -> ld r2,@r2
-     0x290454 <icmst+20>:             ld r3,@r3 -> ld r4,@r4
-     0x290458 <icmst+24>:             ldi r8,#256
-     0x29045c <icmst+28>:             addi r1,#0 || nop
-     0x290460 <icmst+32>:             ld24 r5,0x2a2440 <ldst>
-     0x290464 <icmst+36>:             st r1,@r5 -> bra 0x290470 <icmst+48>
-     0x290468 <icmst+40>:             addi r1,#0 -> addi r1,#0
-     0x29046c <icmst+44>:             addi r1,#0 -> addi r1,#0
-     0x290470 <icmst+48>:             st r2,@(16,r5)
-     0x290474 <icmst+52>:             bra 0x290480 <icmst+64> || nop
-     0x290478 <icmst+56>:             addi r1,#0 -> addi r1,#0
-     0x29047c <icmst+60>:             addi r1,#0 -> addi r1,#0
-     0x290480 <icmst+64>:             st r3,@(32,r5)
-     0x290484 <icmst+68>:             bra 0x290490 <icmst+80> || nop
-     0x290488 <icmst+72>:             addi r1,#0 -> addi r1,#0
-     0x29048c <icmst+76>:             addi r1,#0 -> addi r1,#0
-     0x290490 <icmst+80>:             st r4,@(48,r5)
-     0x290494 <icmst+84>:             bra 0x2904a0 <icmst+96> || nop
-     0x290498 <icmst+88>:             addi r1,#0 -> addi r1,#0
-     0x29049c <icmst+92>:             addi r1,#0 -> addi r1,#0
-     0x2904a0 <icmst+96>:             st r1,@(64,r5)
```

Program stopped at 290440   0x290440   0

Bottleneck analysis from a software viewpoint

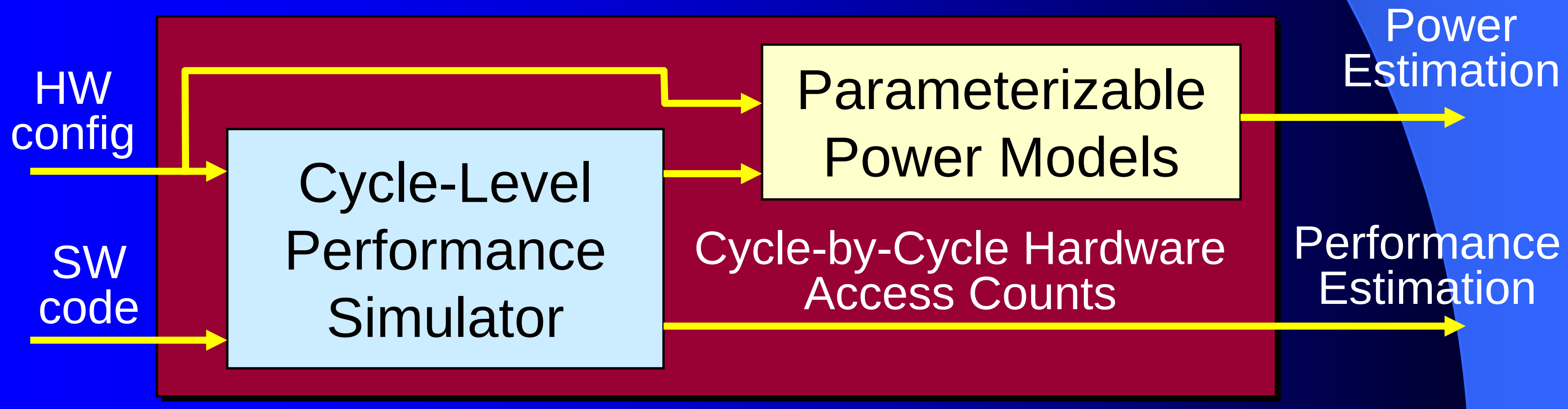


#cycle:
830462
#cycle at max.:
769782
#line(tr.):
718257
#line(tr.) at max.:
668467

maxy = 600   int = 50

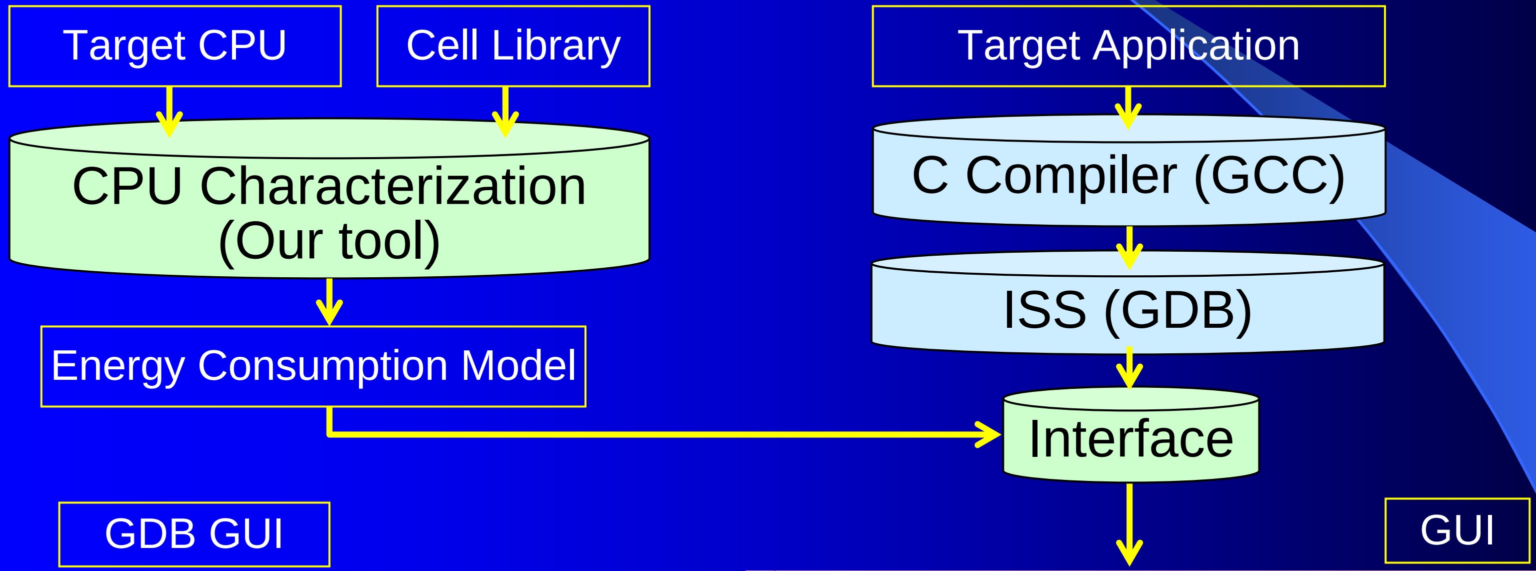
Existing Tools

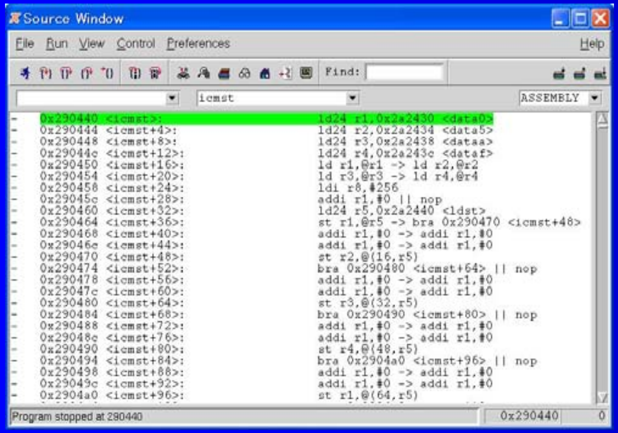
- SimplePower [Irwin@PSU]
  - Based on RTL simulation
  - Energy is calculated from activities of RTL blocks and predetermined energy values of the blocks
- Wattch [Brooks@Princeton]
  - Based on modified SimpleScalar
  - SimpleScalar is not popular among embedded system designers


HW config
SW code
Cycle-Level Performance Simulator
Parameterizable Power Models
Cycle-by-Cycle Hardware Access Counts
Power Estimation
Performance Estimation

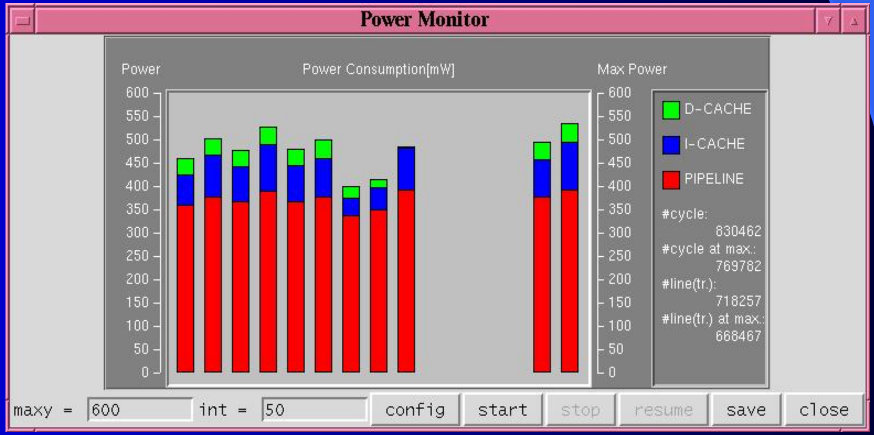
Our Tool

- Technology Independent Framework





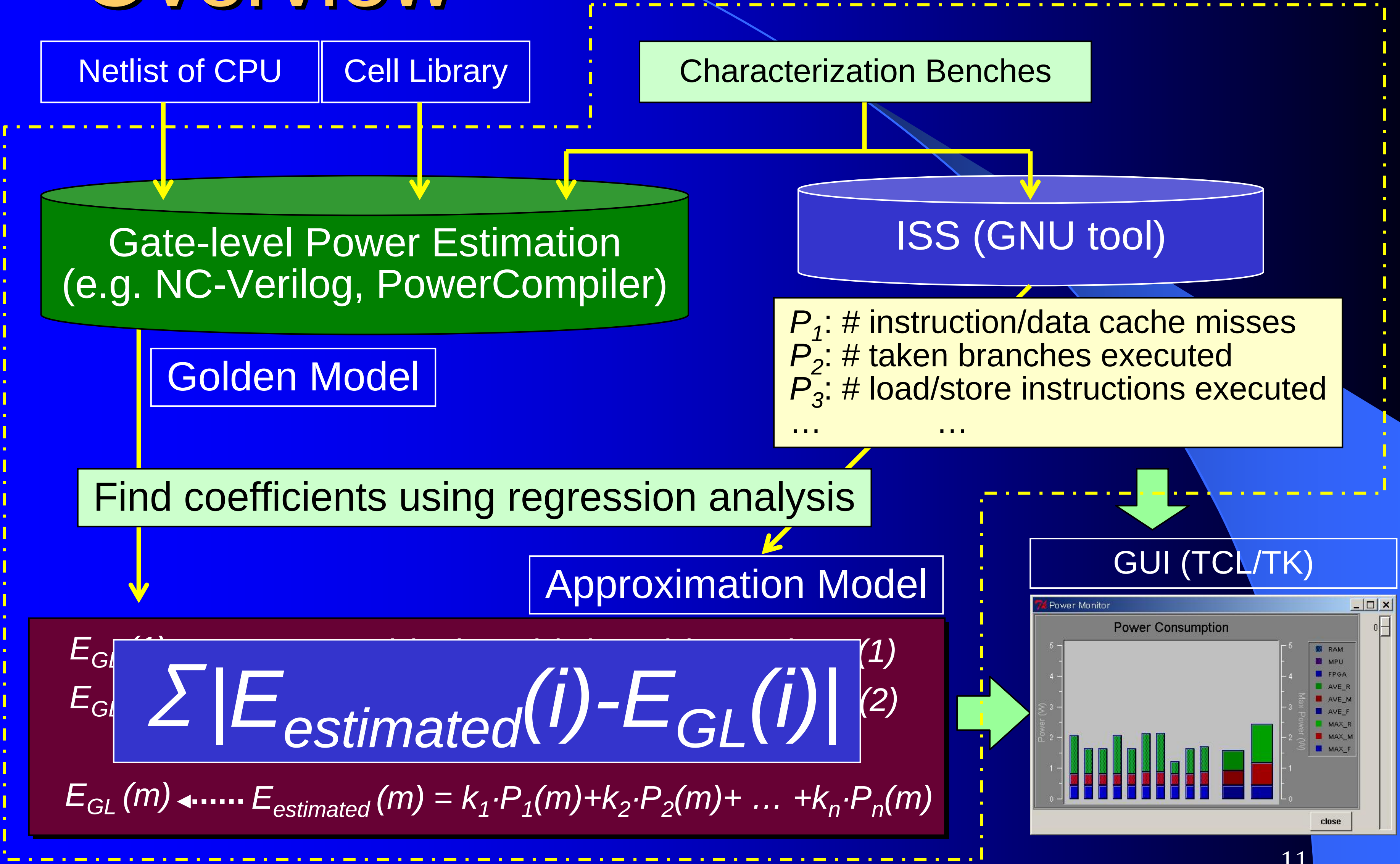
```
-     0x290440 <icmst>:         ld24 r1,0x2a2430 <data0>
-     0x290444 <icmst+4>:       ld24 r2,0x2a2434 <data5>
-     0x290448 <icmst+8>:       ld24 r3,0x2a2438 <dataa>
-     0x29044c <icmst+12>:      ld24 r4,0x2a243c <dataf>
-     0x290450 <icmst+16>:      ld r1,@r1 -> ld r2,@r2
-     0x290454 <icmst+20>:      ld r3,@r3 -> ld r4,@r4
-     0x290458 <icmst+24>:      ldi r8,#256
-     0x29045c <icmst+28>:      addi r1,#0 || nop
-     0x290460 <icmst+32>:      ld24 r5,0x2a2440 <ldst>
-     0x290464 <icmst+36>:      st r1,@r5 -> bra 0x290470 <icmst+48>
-     0x290468 <icmst+40>:      addi r1,#0 -> addi r1,#0
-     0x29046c <icmst+44>:      addi r1,#0 -> addi r1,#0
-     0x290470 <icmst+48>:      st r2,@(16,r5)
-     0x290474 <icmst+52>:      bra 0x290480 <icmst+64> || nop
-     0x290478 <icmst+56>:      addi r1,#0 -> addi r1,#0
-     0x29047c <icmst+60>:      addi r1,#0 -> addi r1,#0
-     0x290480 <icmst+64>:      st r3,@(32,r5)
-     0x290484 <icmst+68>:      bra 0x290490 <icmst+80> || nop
-     0x290488 <icmst+72>:      addi r1,#0 -> addi r1,#0
-     0x29048c <icmst+76>:      addi r1,#0 -> addi r1,#0
-     0x290490 <icmst+80>:      st r4,@(48,r5)
-     0x290494 <icmst+84>:      bra 0x2904a0 <icmst+96> || nop
-     0x290498 <icmst+88>:      addi r1,#0 -> addi r1,#0
-     0x29049c <icmst+92>:      addi r1,#0 -> addi r1,#0
-     0x2904a0 <icmst+96>:      st r1,@(64,r5)
Program stopped at 290440                          0x290440   0
```



#cycle: 830462
#cycle at max.: 769782
#line(tr.): 718257
#line(tr.) at max.: 668467

maxy = 600 int = 50

Overview



Netlist of CPU

Cell Library

Characterization Benches

Gate-level Power Estimation (e.g. NC-Verilog, PowerCompiler)

ISS (GNU tool)

P_1: # instruction/data cache misses
P_2: # taken branches executed
P_3: # load/store instructions executed
… …

Golden Model

Find coefficients using regression analysis

Approximation Model

GUI (TCL/TK)

E_{GL} (1) … (1)

E_{GL} … (2)

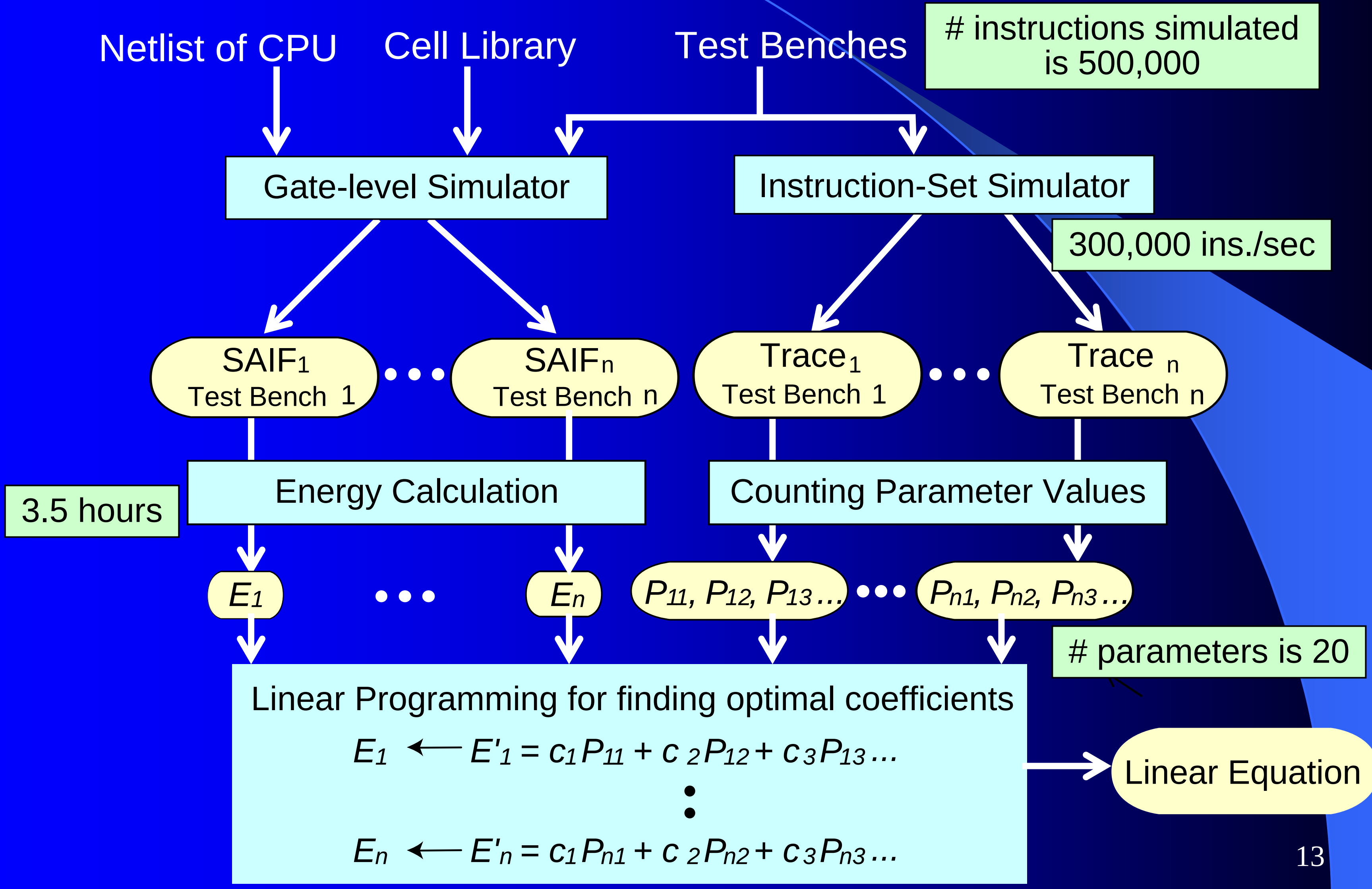
$$\Sigma|E_{estimated}(i)-E_{GL}(i)|$$

$E_{GL}(m)$ ◂⋯⋯ $E_{estimated}(m) = k_1 \cdot P_1(m)+k_2 \cdot P_2(m)+ \ldots +k_n \cdot P_n(m)$

Experimental Setup

- M32R-II or SH3-DSP and SDRAM (Micron)
  - 5-stage pipeline
  - 8KB 2-way I-Cache and D-Cache
  - 32KB SRAM
- GNU CC (e.g., m32r-linux-gcc)
- NC-Verilog from Cadence and PowerCompiler from SYNOPSYS for the Gate-Level Energy Estimation
- 0.18um Standard Cell Library
- System Power Calculator for the energy model of SDRAM
- GNU based ISS (e.g, m32r-linux-run)
- CPLEX from ILOG for solving Linear Programming

Detailed Characterization Flow



Netlist of CPU

Cell Library

Test Benches

# instructions simulated is 500,000

Gate-level Simulator

Instruction-Set Simulator

300,000 ins./sec

SAIF_1 Test Bench 1 • • • SAIF_n Test Bench n

Trace_1 Test Bench 1 • • • Trace_n Test Bench n

3.5 hours

Energy Calculation

Counting Parameter Values

E_1 • • • E_n

$P_{11}, P_{12}, P_{13} \ldots$ • • • $P_{n1}, P_{n2}, P_{n3} \ldots$

# parameters is 20

Linear Programming for finding optimal coefficients

$$E_1 \longleftarrow E'_1 = c_1 P_{11} + c_2 P_{12} + c_3 P_{13} \ldots$$

$$\vdots$$

$$E_n \longleftarrow E'_n = c_1 P_{n1} + c_2 P_{n2} + c_3 P_{n3} \ldots$$

Linear Equation

Experimental Results (M32R-II)

Benchmark Program	Error (%)		Standard Deviation of error Percentage
	Ave.	Max.	
JPEG	2.70	10.32	2.76
JPEG_opt	6.09	16.46	6.17
MPEG2	1.54	3.97	0.94
MPEG2_opt	1.78	5.15	0.96
compress	5.00	6.41	1.19
compress_opt	4.35	7.18	0.93
FFT	1.55	6.87	0.92
FFT_opt	1.45	5.59	0.89
DCT	1.42	8.58	0.72
DCT_opt	1.47	8.07	0.69
Total	2.74	16.46	1.62

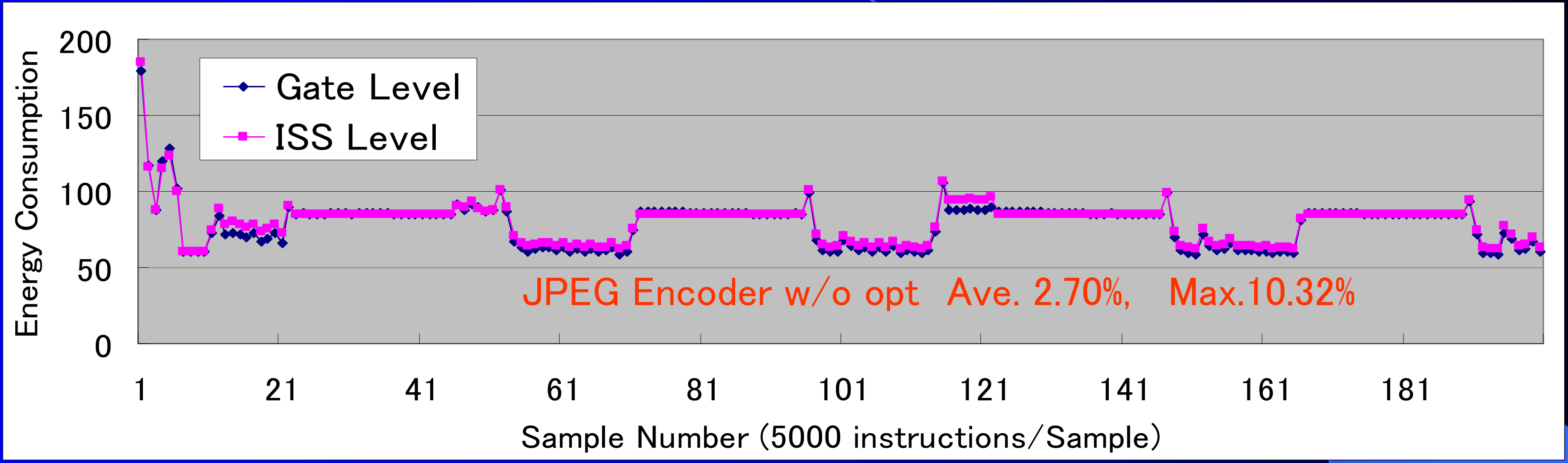
∗ _opt corresponds programs compiled with -O3 option

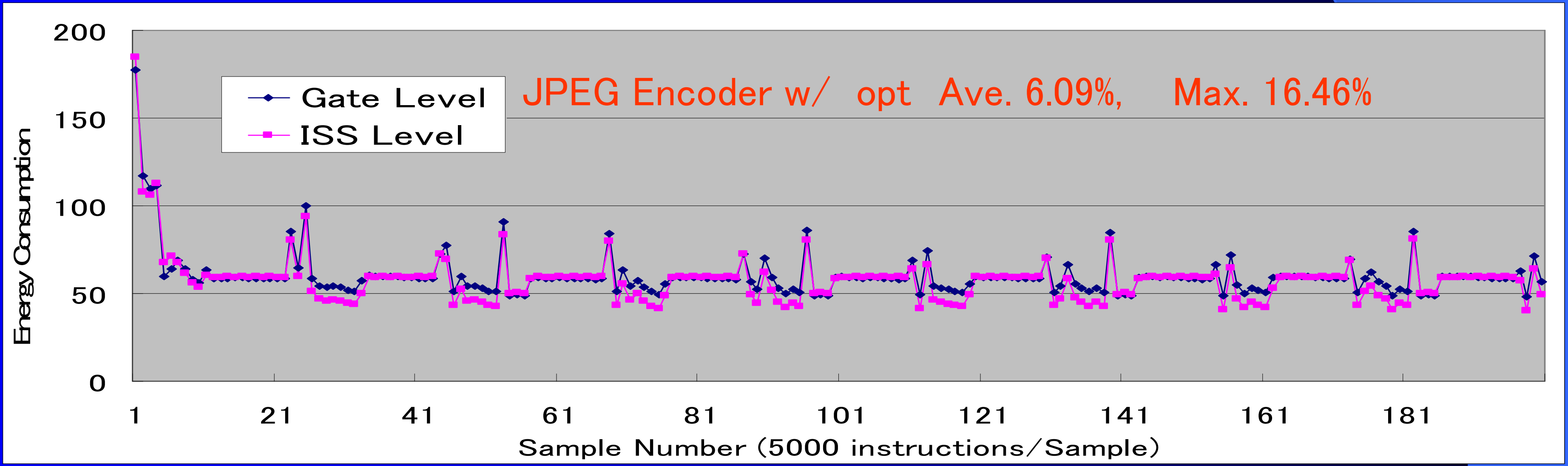
Experimental Results (SH3-DSP)

Benchmark Program	Error (%)		Standard Deviation of Error Percentage
	Ave.	Max.	
JPEG	3.61	11.23	2.37
JPEG_opt	5.09	15.70	3.73
MPEG2	3.43	5.93	1.20
MPEG2_opt	3.33	5.59	1.16
compress	8.50	10.67	0.84
compress_opt	1.48	15.23	1.22
FFT	2.89	5.87	1.07
FFT_opt	3.25	6.30	1.31
DCT	0.72	1.91	0.28
DCT_opt	0.99	2.38	0.40
Total	3.33	15.70	1.36

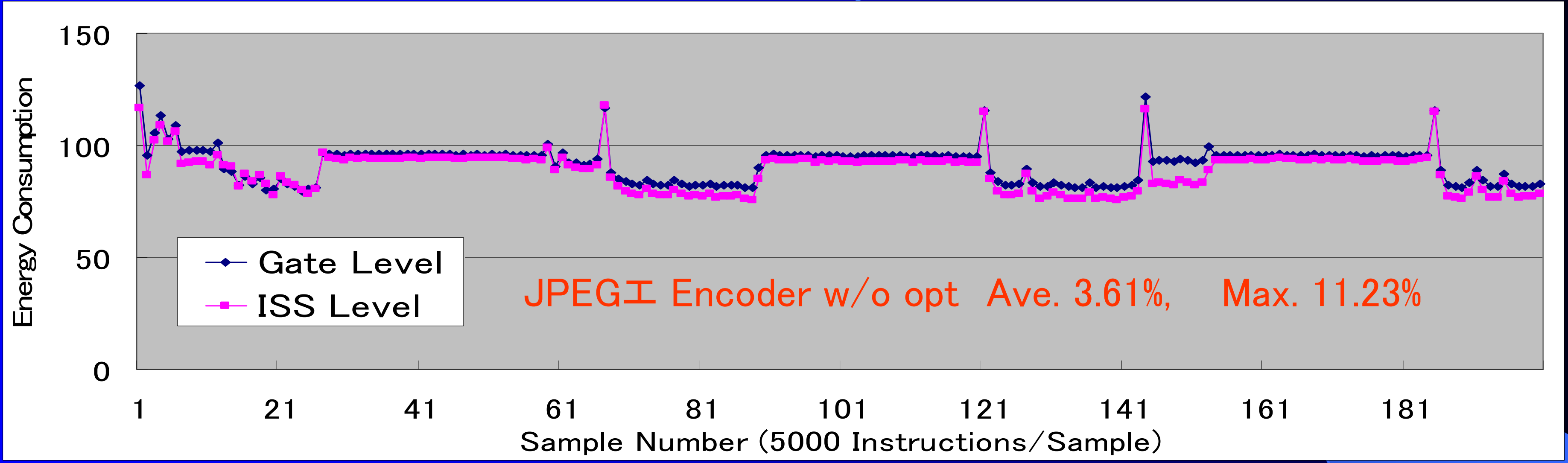
∗_opt corresponds programs compiled with -O3 option

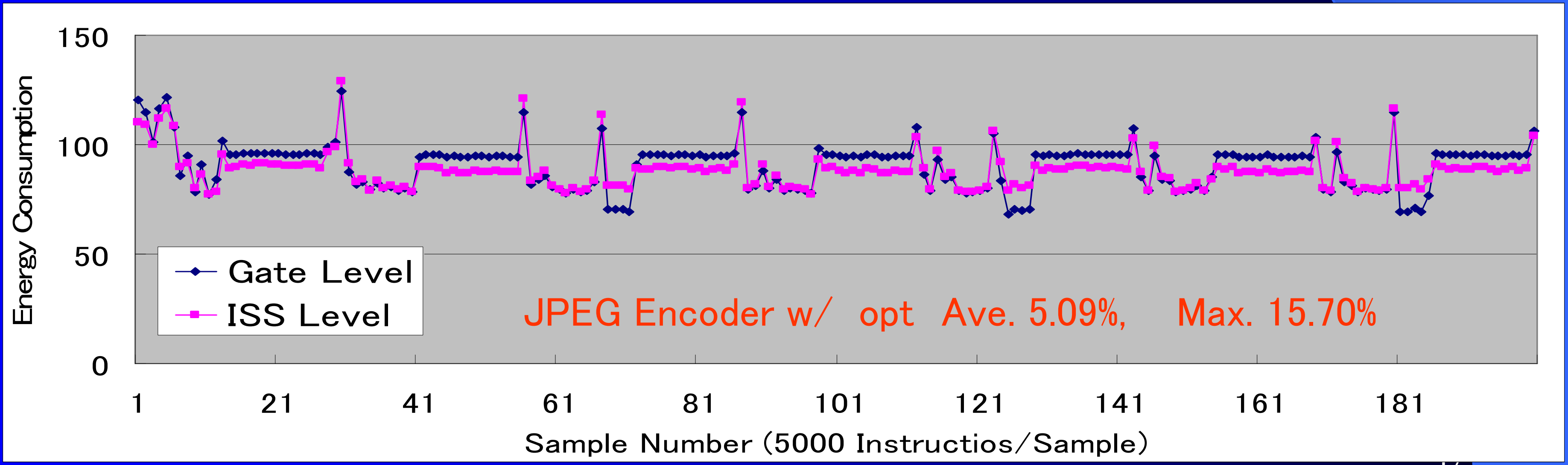
JPEG Encoder (M32R-II)





JPEG Encoder (SH3-DSP)





Summary

- Proposed an energy characterization framework
- The error of our approach is 3% on an average and 16% at the maximum case.

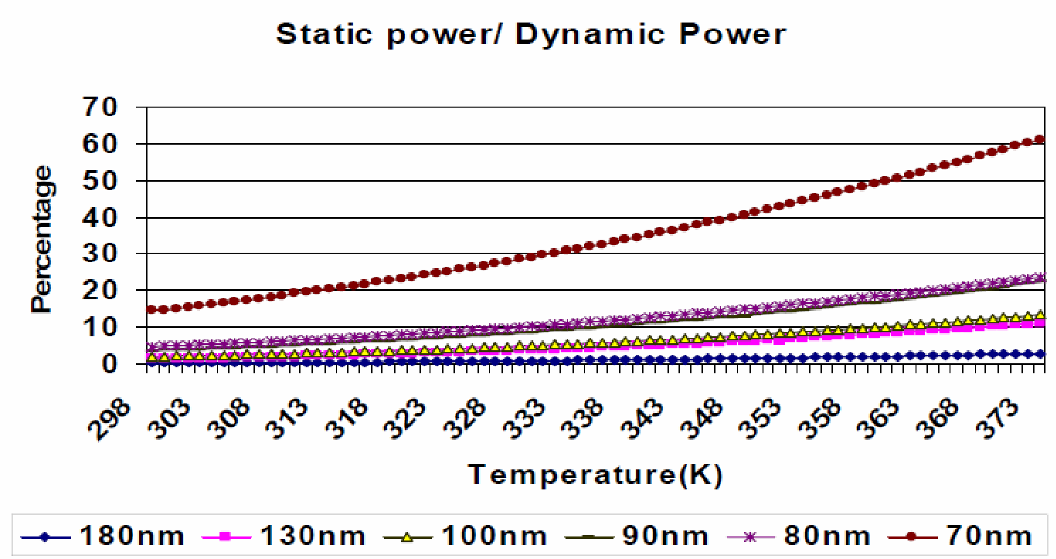
Future work

- Extending the current approach for targeting multi-core processors

Agenda

- Introduction
- Software-Level Energy Characterization
- Process-Variation Aware Compilation

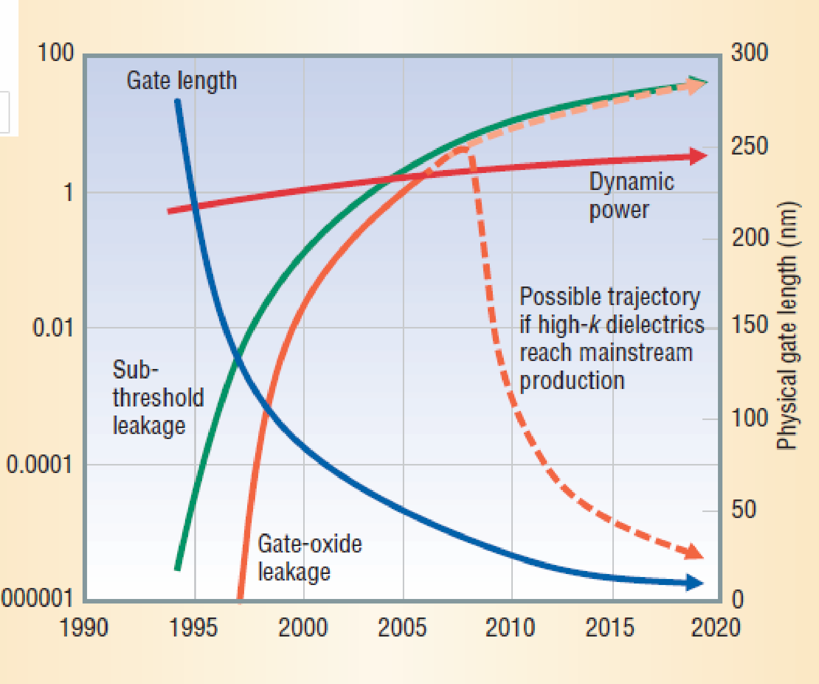
Leakage Energy



Source: J. Gonzalez and K. Skadron, “Power-Aware Design for High-performance Processors,” Tutorial in conjunction with 10th International Symposium on High-Performance Computer Architecture, Feb. 2004.

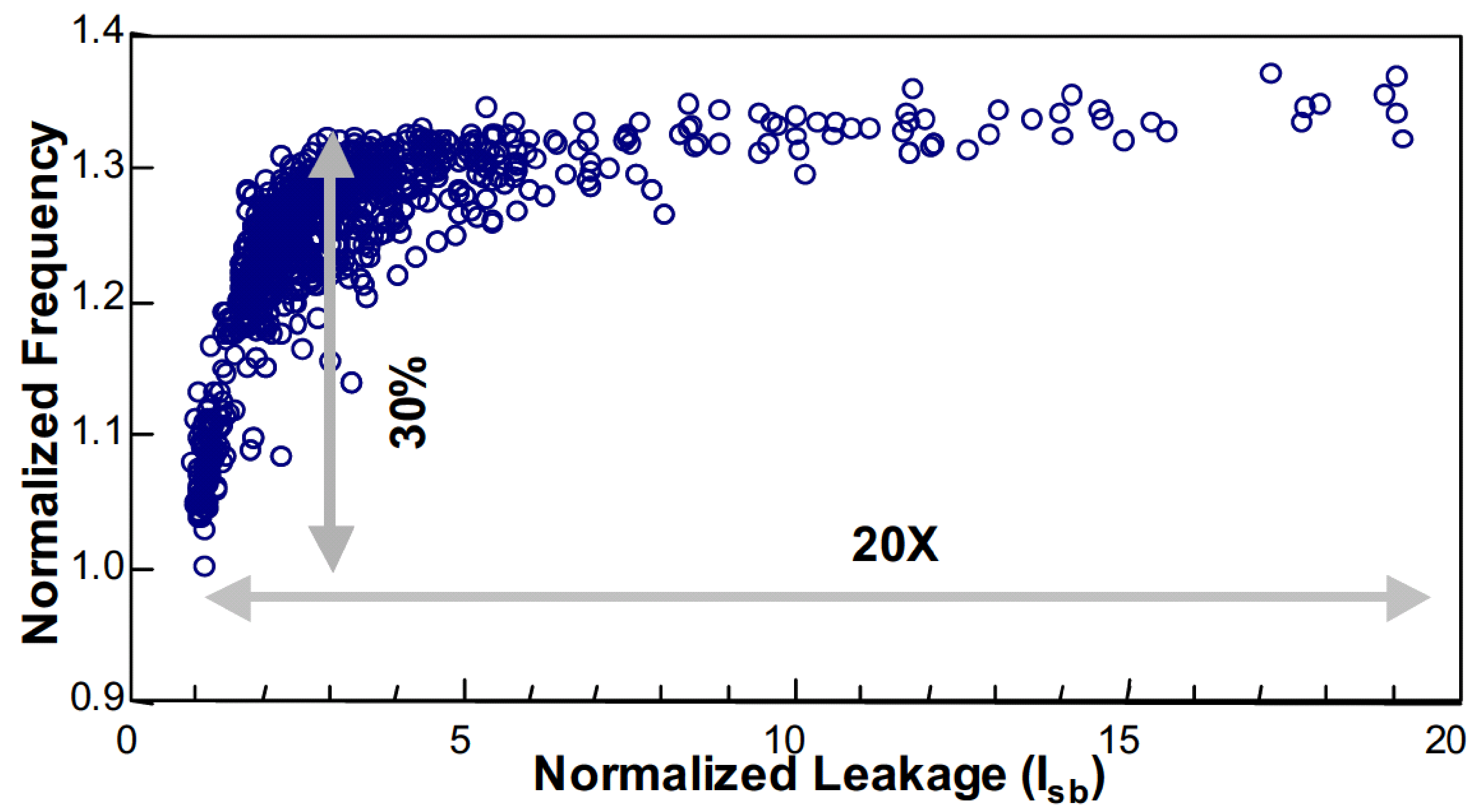
Leakage exponentially increases along with the chip temperature

Leakage exponentially increases as the transistor size shrinks



Source: N. S. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan, “Leakage Current: Moore’s Law Meets Static Power,” IEEE Computer, Vol. 36, No. 12, pp.68-75, Dec. 2003.

Process Variation



S. Borkar, Parameter variations and impact on circuits and microarchitecture, DAC 2003.

Motivation

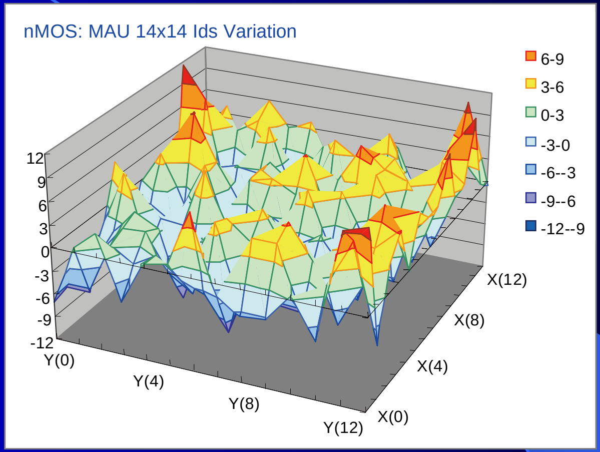
Large Intra-Die Variation

Current 3-sigma = 13%
Vth 3-sigma = 67mV

Variation is huge in small transistors

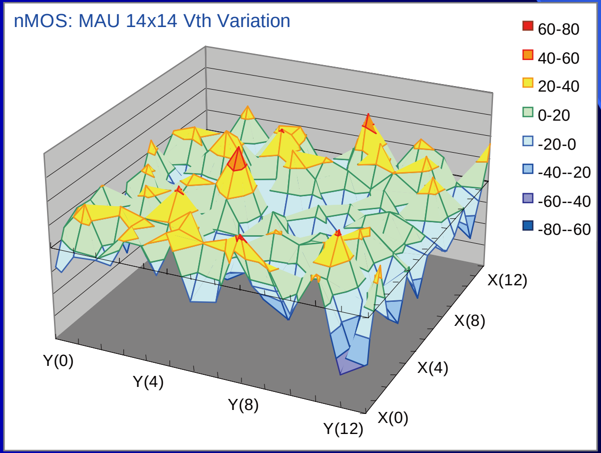
$$\sigma_{Vth} = \frac{q}{C_{ox}}\sqrt{\frac{N_a \cdot W_{dm}}{3\cdot L\cdot W}}$$

L, *W*: Effective channel length and width
q: electron charge
C_{ox}: oxide capacitance
N_a: substrate doping concentration
W_{dm}: maximum depletion width



L = 0.1um
W= 0.4um

Av. = 203.7uA
Sigma = 4.4%
min. = -11.4%
max. = 11.4%



L = 0.1um
W= 0.4um

Av. = 308.3uA
Sigma = 22.1mV
min. = -66.6mV
max. = 57.0mV

Eijiro Toyoda, “DFM: Device & Circuit Design Challenges”, Int’l Forum on Semiconductor Technology, 2004

Process Variation at 90nm

$$I_{Subthreshold} \propto \frac{W \cdot V_T^2}{T_{ox} \cdot L} \cdot \exp\left(\frac{-V_{th}}{\alpha \cdot V_T}\right)$$

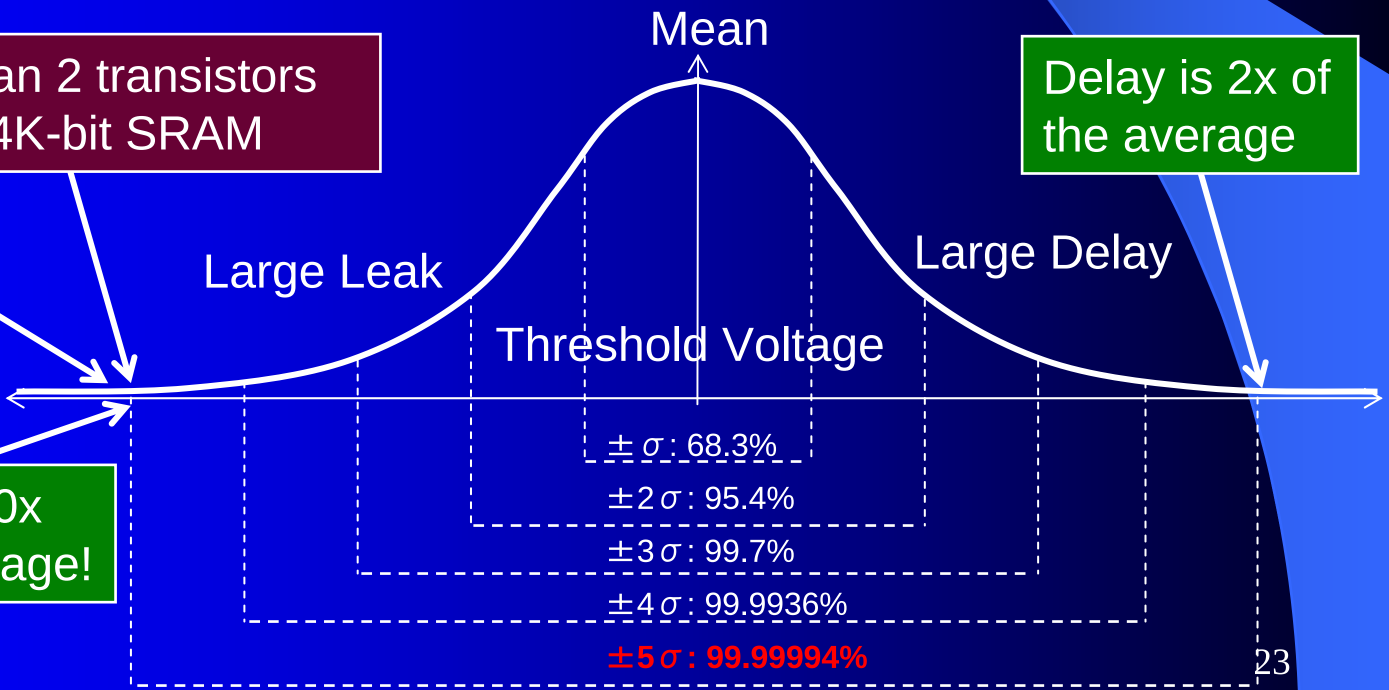
V_T: Thermal voltage (25mV@room temperature)

α : Sub-threshold factor (1.40~1.65)

T_{ox}: Oxide thickness

Year	min. L [nm]	[1]V_{TH} [V]	[2]V_{TH} [V]
2004	37 (90)	0.32	0.12
2005	32 (80)	0.33	0.09
2006	28 (70)	0.34	0.06

1: Low Operating Power Process 2: MPU process



More than 2 transistors out of 64K-bit SRAM

Delay is 2x of the average

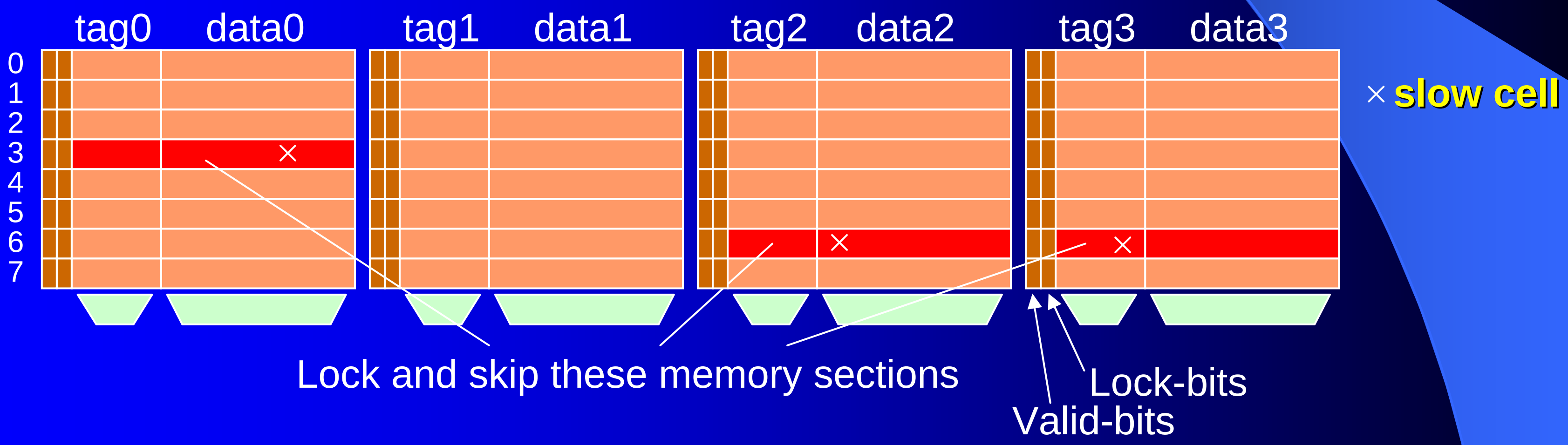
$5\sigma_{Vth}$=0.3V

Leakage is 1,500x higher than average!

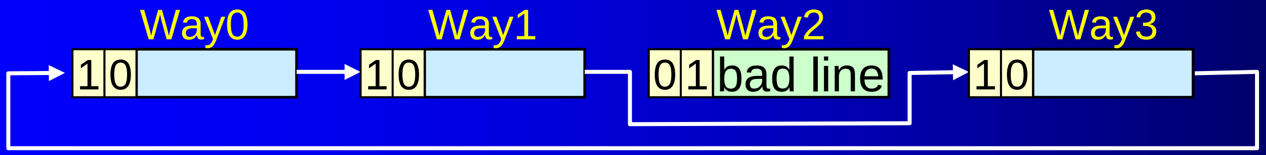
Marking Bad Cache-Lines

- Use an unused combination of existing flag bits to indicate a slow SRAM cell in a specific cache-line.
- Invalidate and skip a cache-line if it is marked.

4-way set-associative cache memory

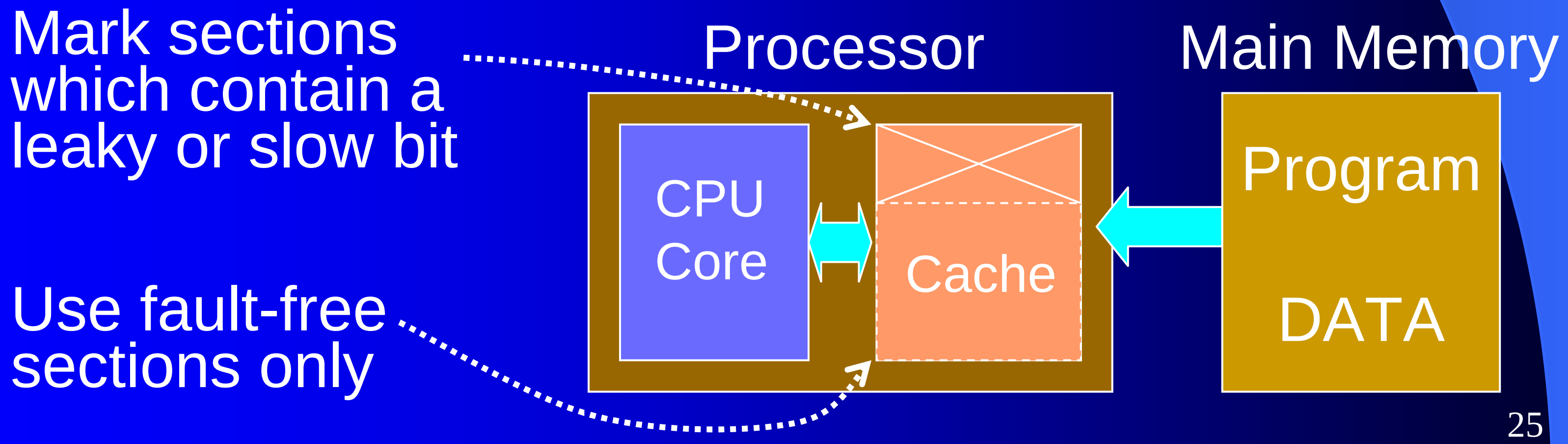


Cache replacement policy



Cache Miss Reduction

- Using a smaller cache memory does not affect the correct operation of a processor.
- The idea is to mark extremely slow cache-lines and to use fast cache-lines only.
- The problem is an increase of cache miss rate due to a reduced cache size.



Mark sections which contain a leaky or slow bit

Use fault-free sections only

Processor

CPU Core

Cache

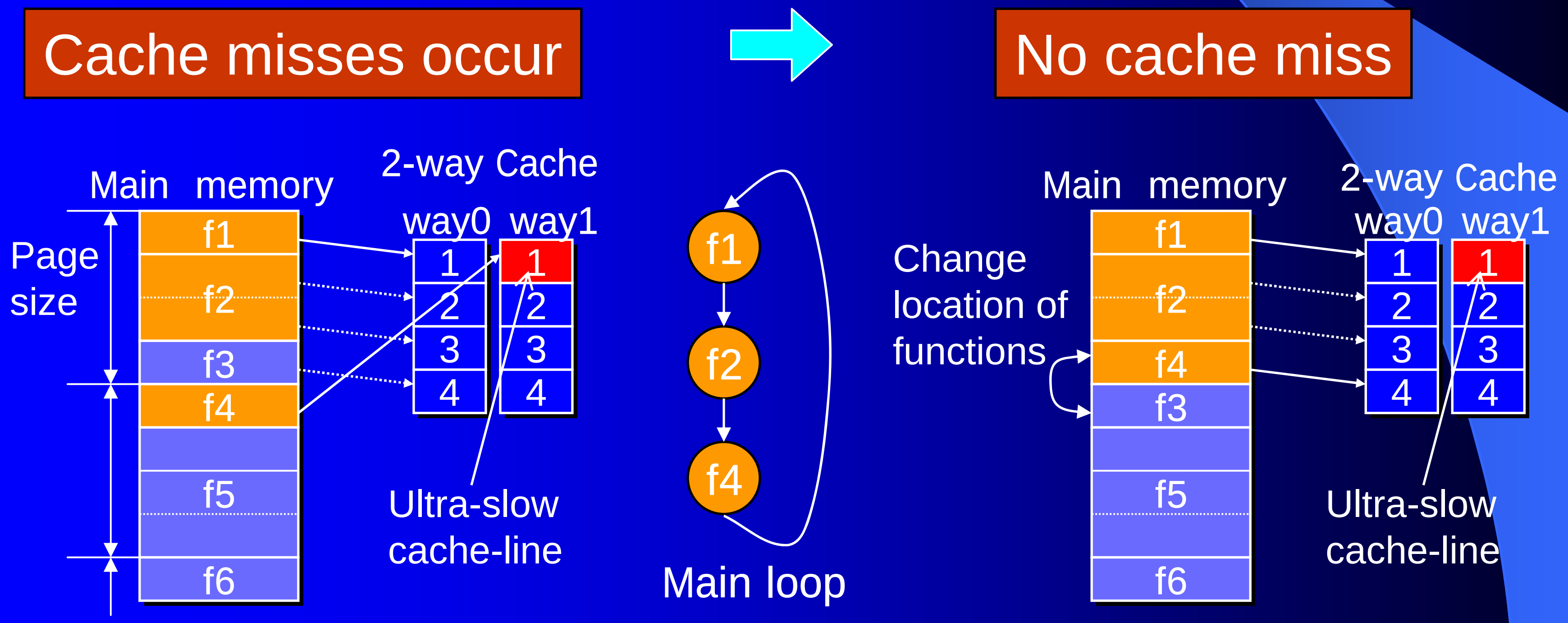
Main Memory

Program

DATA

Our Approach

- Modify the order of functions in the address space such that ultra-slow cache-lines are not accessed frequently.



Cache misses occur

No cache miss

Process Variation at 90nm

$$I_{Subthreshold} \propto \frac{W \cdot V_T^2}{T_{ox} \cdot L} \cdot \exp\left(\frac{-V_{th}}{\alpha \cdot V_T}\right)$$

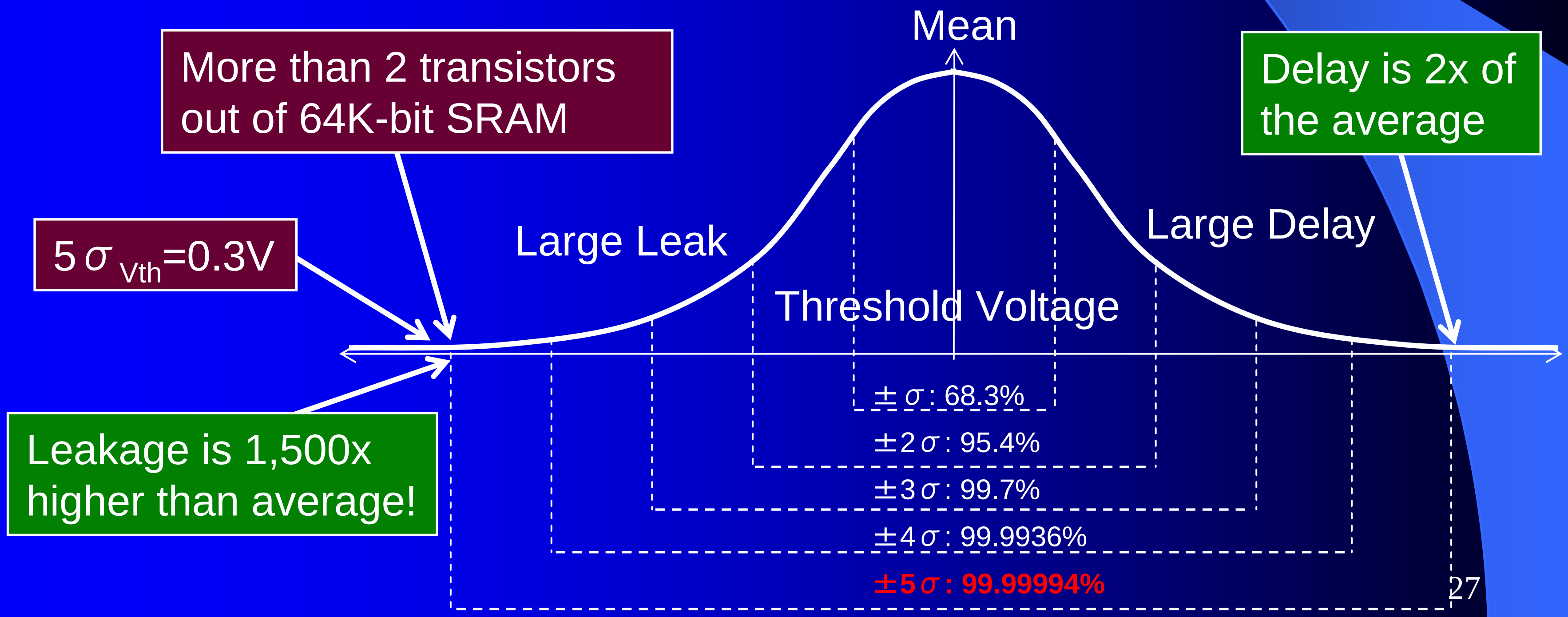
V_T: Thermal voltage (25mV@room temperature)

α : Sub-threshold factor (1.40~1.65)

T_{ox}: Oxide thickness

Year	min. L [nm]	$^1V_{TH}$ [V]	$^2V_{TH}$ [V]
2004	37 (90)	0.32	0.12
2005	32 (80)	0.33	0.09
2006	28 (70)	0.34	0.06

1: Low Operating Power Process 2: MPU process



More than 2 transistors out of 64K-bit SRAM

$5\sigma_{Vth}$=0.3V

Leakage is 1,500x higher than average!

Delay is 2x of the average

±σ : 68.3%

±2σ : 95.4%

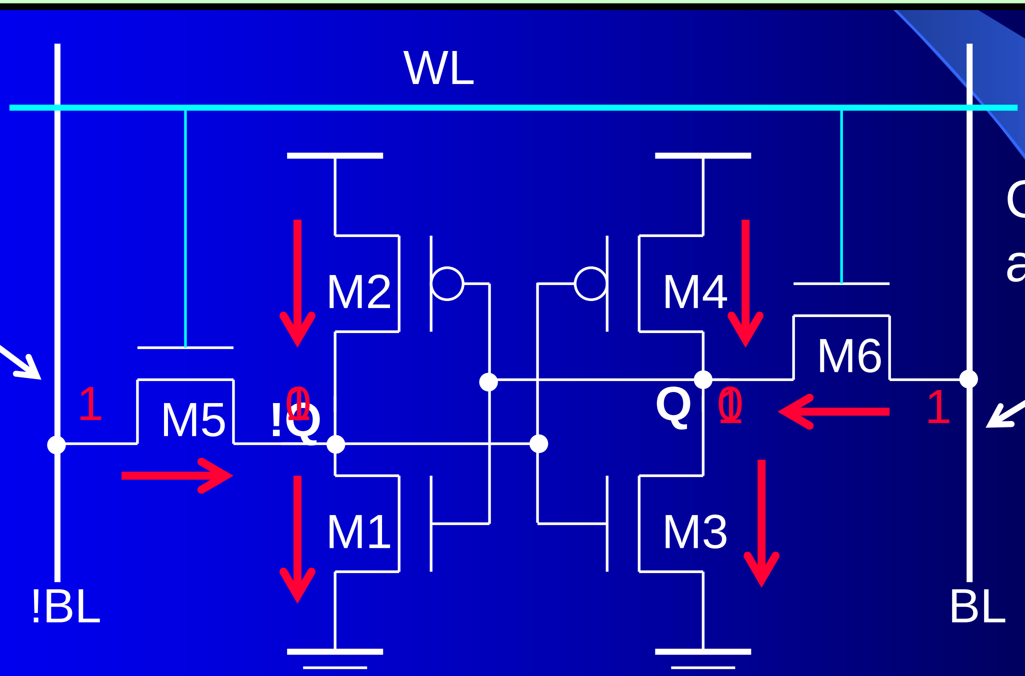
±3σ : 99.7%

±4σ : 99.9936%

±5σ : 99.99994%

Masking Leaky Cells

Leakage current of a SRAM cell depends on the logic value stored



Charged to Vdd at inactive mode

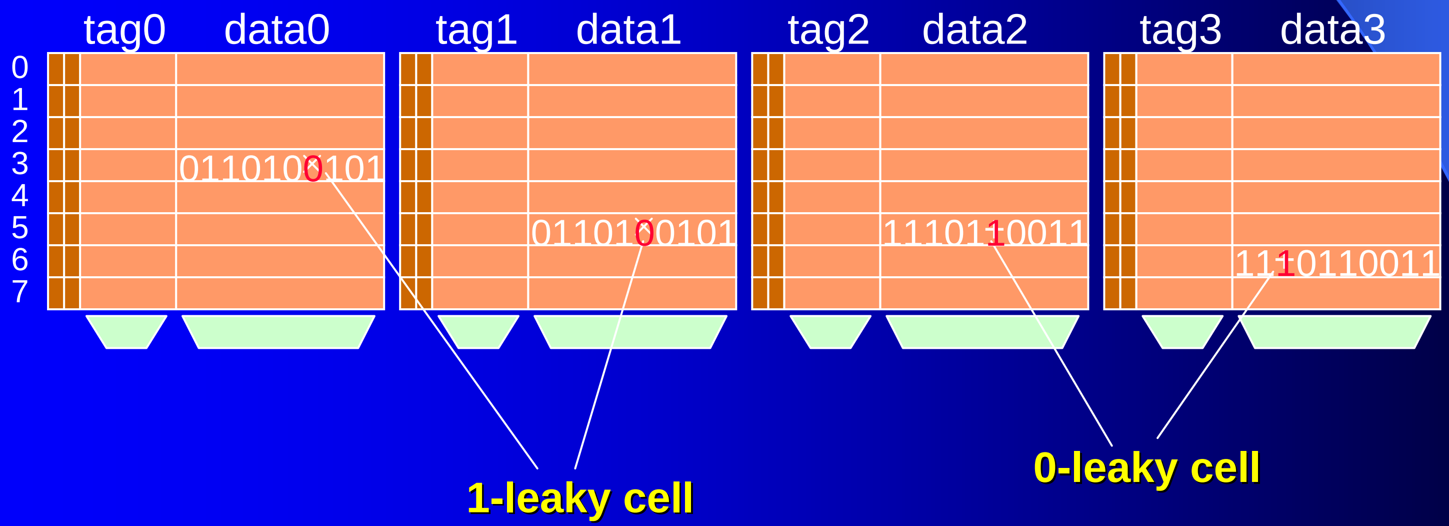
Charged to Vdd at inactive mode

If M2, M3, or M5 is leaky, the SRAM cell is 1-leaky
If M1, M4, or M6 is leaky, the SRAM cell is 0-leaky

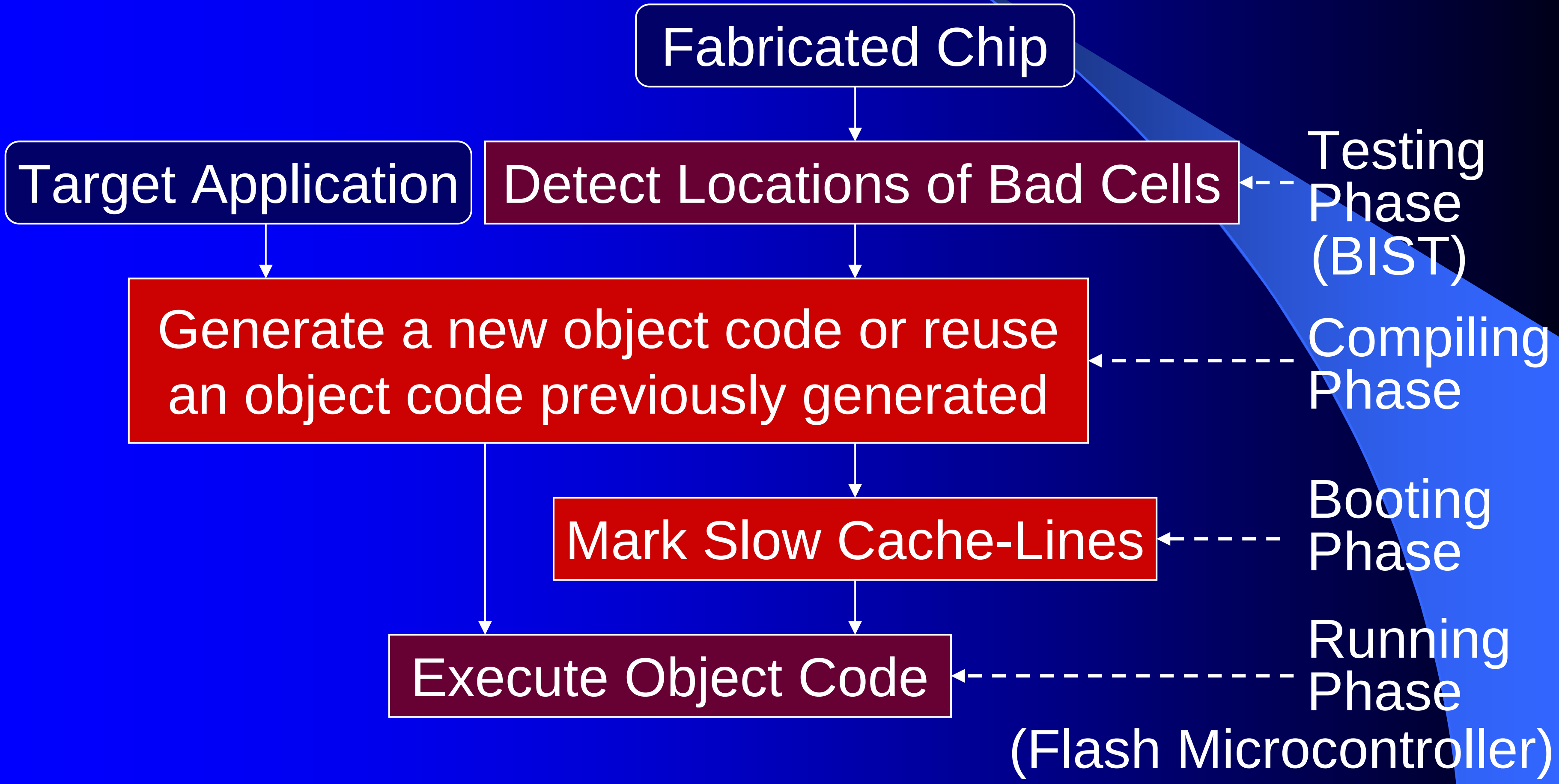
Masking Leaky Cache-Lines

- Modify the order of instruction codes considering binary expressions of the codes and locations of 0/1-leaky bits in a cache so that the total leakage current is minimized

4-way set-associative cache memory



The Flow



Fabricated Chip

Target Application

Detect Locations of Bad Cells

Testing Phase (BIST)

Generate a new object code or reuse an object code previously generated

Compiling Phase

Mark Slow Cache-Lines

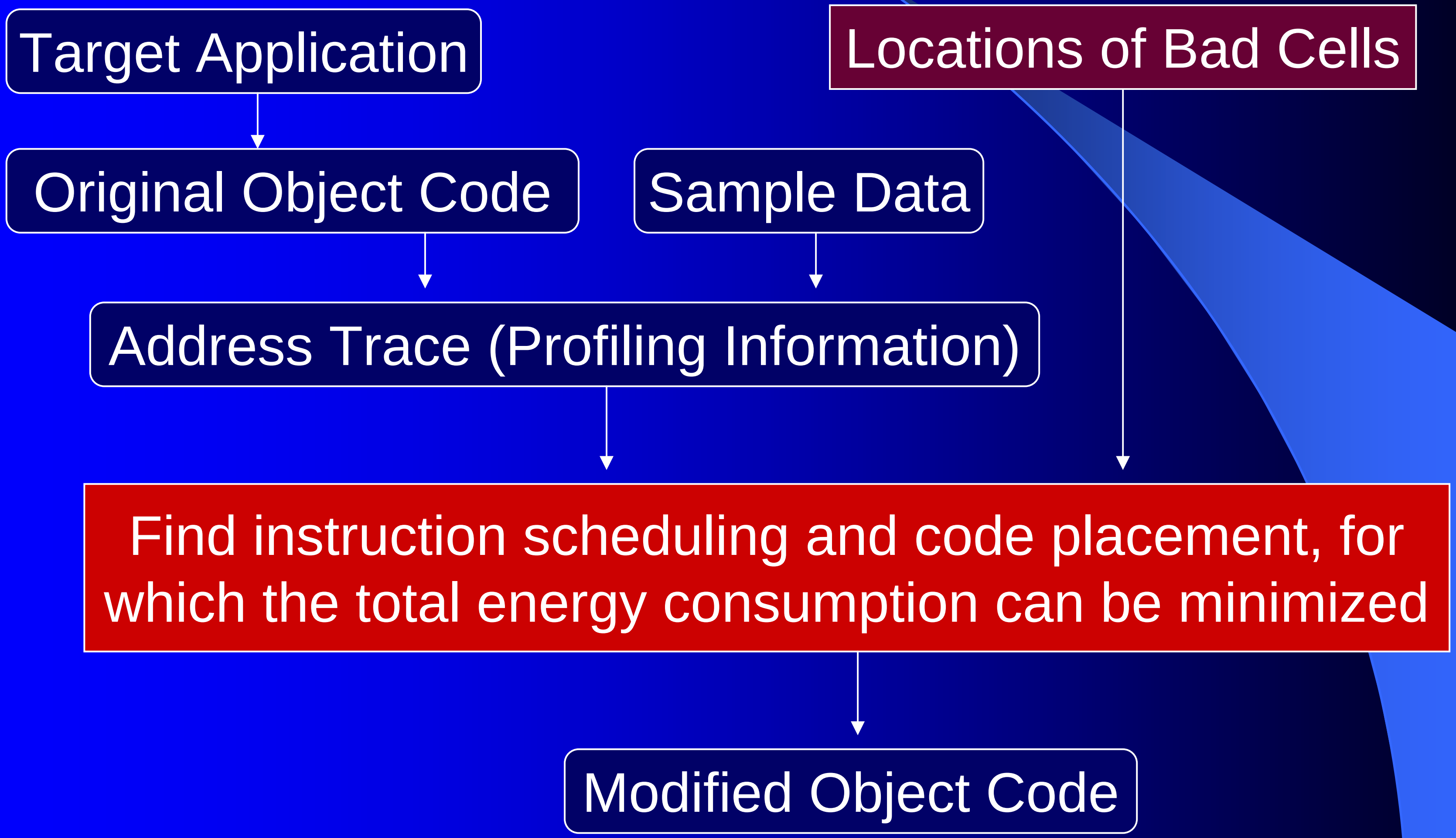
Booting Phase

Execute Object Code

Running Phase

(Flash Microcontroller)

Compiler Optimization Flow



Target Application

Locations of Bad Cells

Original Object Code

Sample Data

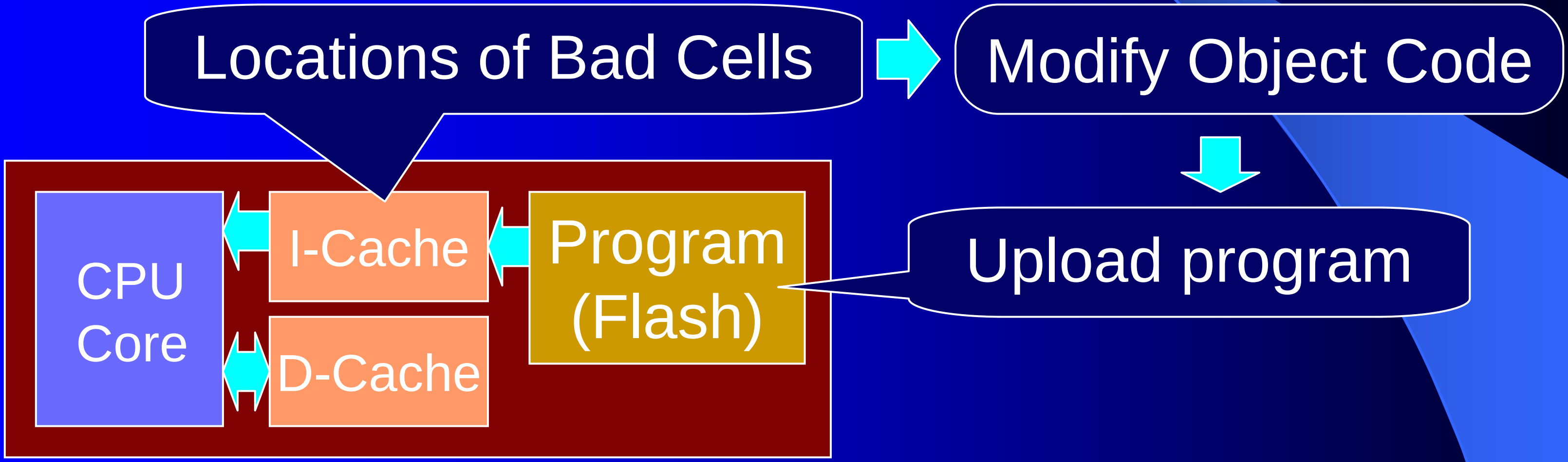
Address Trace (Profiling Information)

Find instruction scheduling and code placement, for which the total energy consumption can be minimized

Modified Object Code

New Paradigm

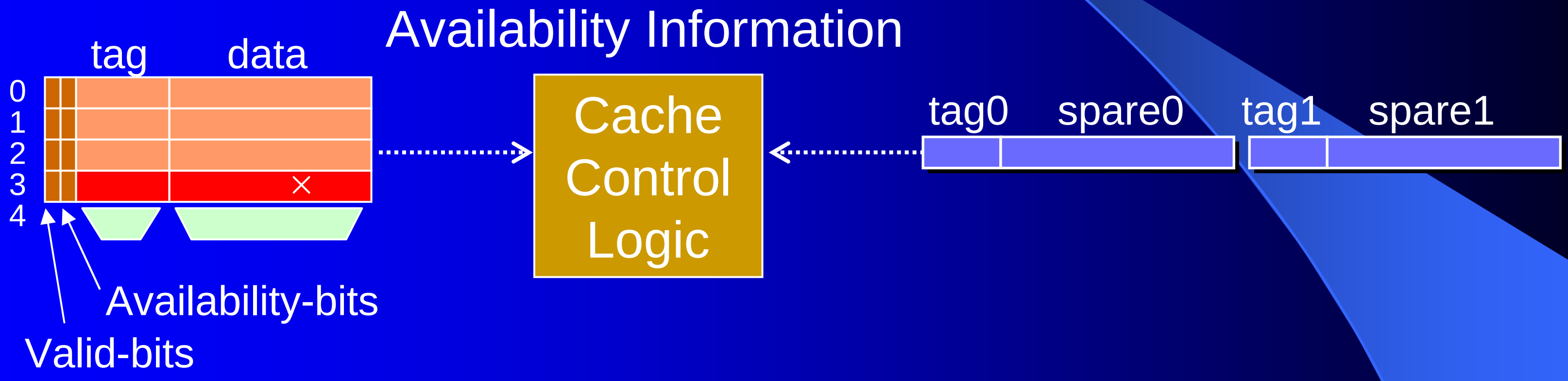
- Use different object codes for different chips



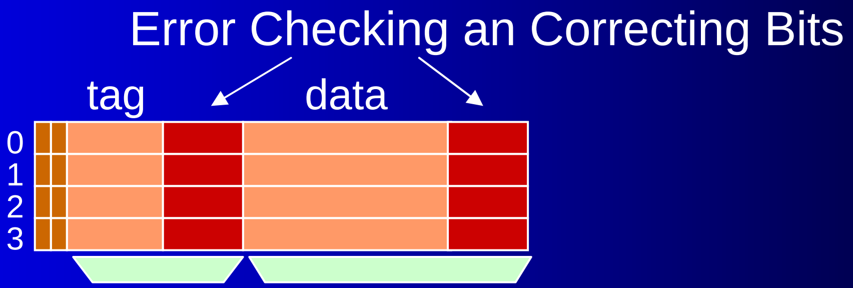
- Future work: Reducing the test cost

Previous Work (1/2)

- Vergos et al. proposed a technique using spare cache

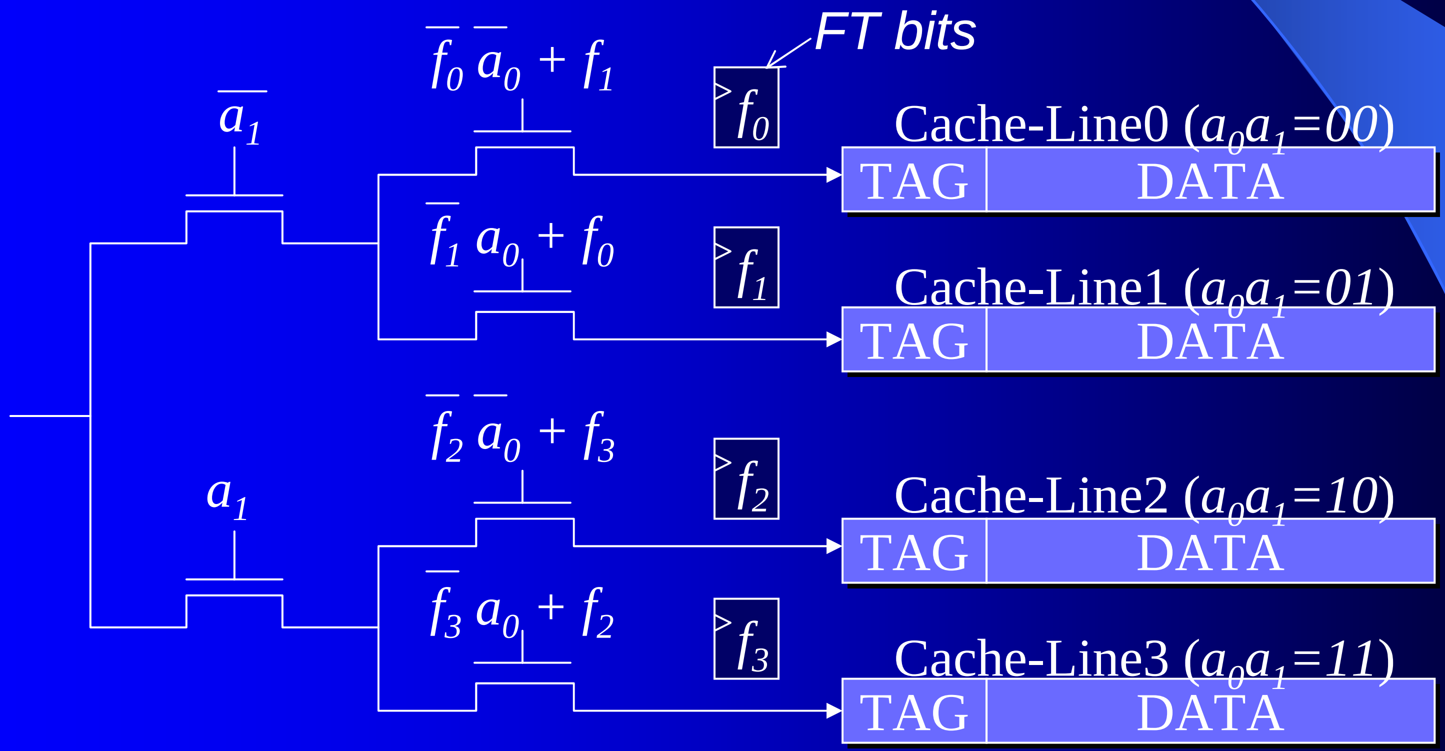


- Sohi proposed a technique using error correcting code



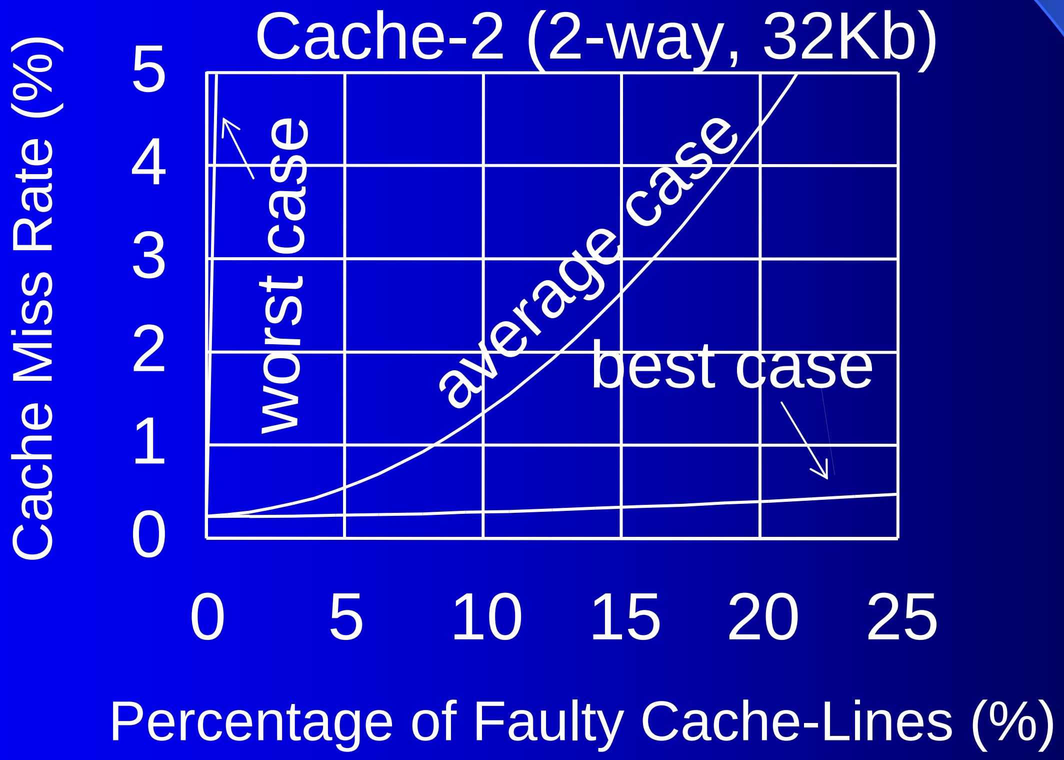
Previous Work (2/2)

- Shiravani et al. proposed *PADded* cache
  - Customize an address decoder so that faulty blocks will not be accessed.

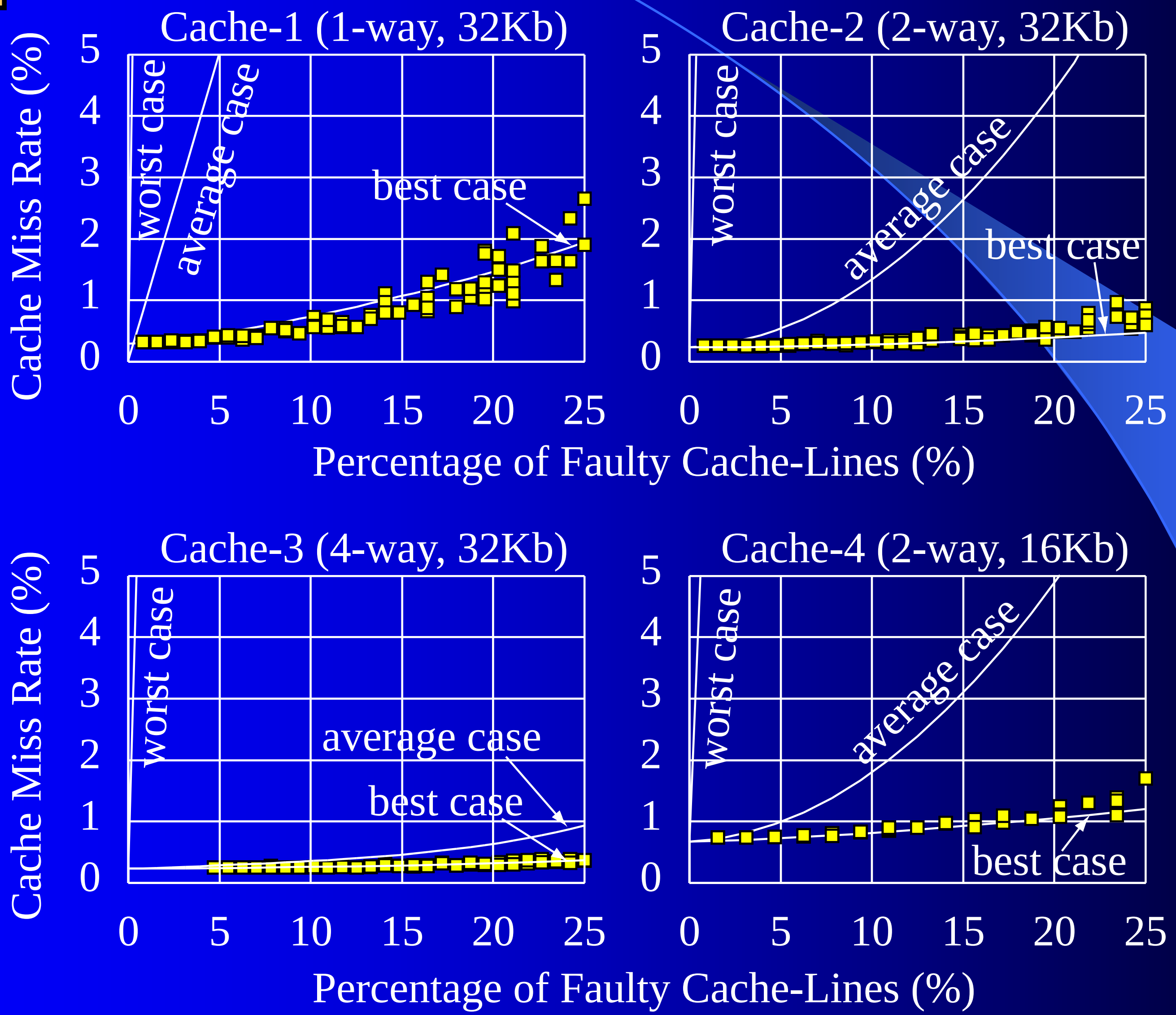

$\overline{f_0}\,\overline{a_0} + f_1$
FT bits
$\overline{a_1}$
f_0
Cache-Line0 (a_0a_1=00)
TAG
DATA
$\overline{f_1}\,a_0 + f_0$
f_1
Cache-Line1 (a_0a_1=01)
TAG
DATA
$\overline{f_2}\,\overline{a_0} + f_3$
a_1
f_2
Cache-Line2 (a_0a_1=10)
TAG
DATA
$\overline{f_3}\,a_0 + f_2$
f_3
Cache-Line3 (a_0a_1=11)
TAG
DATA

Experimental Setup

- Applied our technique to ARMv4T architecture
- Used three programs, *Compress*, *JPEG*, and *MPEG2*
- Considered three scenarios: best, typical, and worst



Experimental Results



Summary

- Cancel the degradation of cache hit-rate even in presence of 25% slow cache-lines.
- Worst case (5-sigma) delay can be reduced by 40%.
- No major HW modification is required.

Future work

- Implement an instruction scheduling algorithm for reducing leakage current

Conclusion

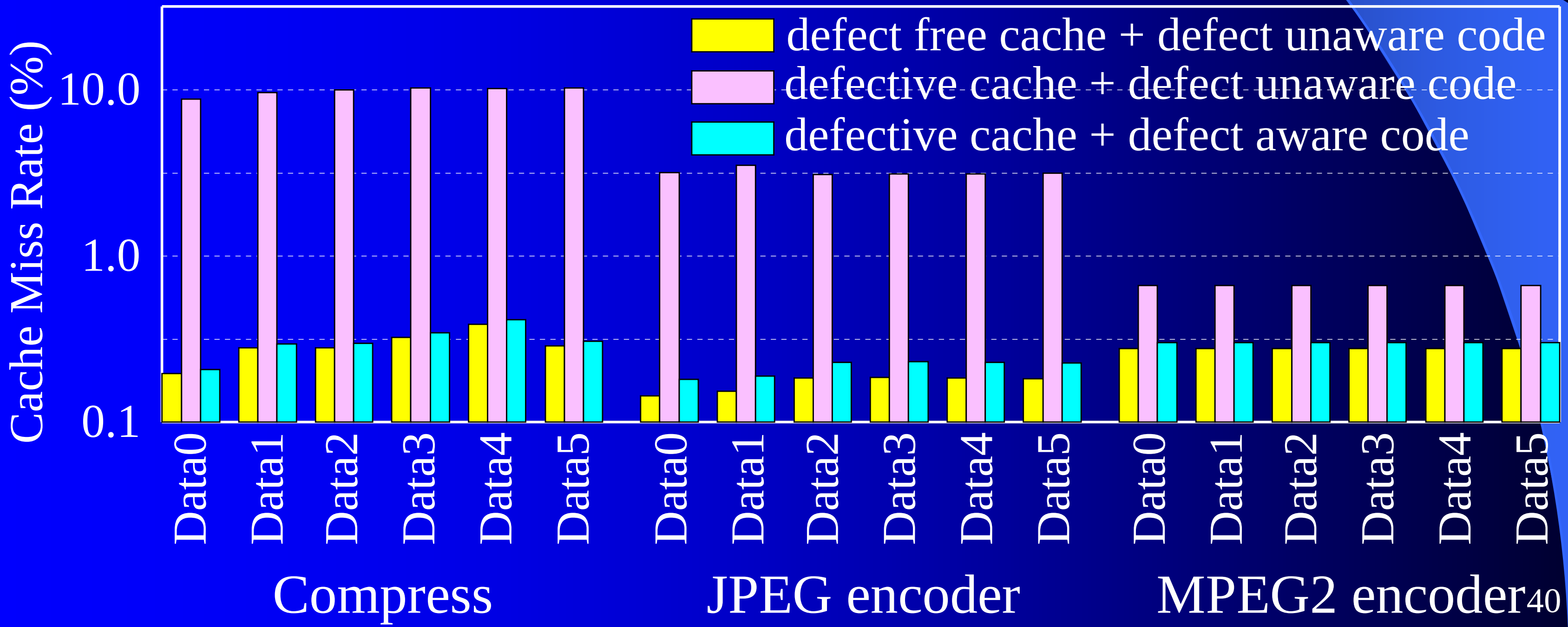
- Flexibility and customizability become much more important in future technologies.
- Process-variation-aware design at system-level is essential for saving energy.
- Hardware and software cooperation is very important.

1. L. Donghoon, T. Ishihara, M. Muroyama, H. Yasuura and F. Fallah, “An Energy Characterization Technique for Fast and Accurate Software Power Estimation (in Japanese)”, IPSJ Technical Report, March 2006
2. T. Ishihara and F. Fallah, “A Non-Uniform Cache Architecture for Low Power System Design”, ISLPED 2005, August 2005
3. T. Ishihara and F. Fallah, “A Code Placement Technique for Improving the Performance of Processors with Defective Caches”, IWLS 2005, June 2005
4. T. Ishihara and F. Fallah, “A Cache-Defect-Aware Code Placement Algorithm for Improving the Performance of Processors”, ICCAD 2005, November 2005

Thank you!!

Input Data Dependency

- Compared cache miss rates for 6 different input values.
- The optimized code for Data0 achieves very good results for other input values too.


Cache Miss Rate (%)
defect free cache + defect unaware code
defective cache + defect unaware code
defective cache + defect aware code
Data0
Data1
Data2
Data3
Data4
Data5
Compress
JPEG encoder
MPEG2 encoder