Linux Power Management on a Fine-Grain Power-Gating Processor

LINUXCON JAPAN 2012

Hiroaki Kobayashi

Department of Computer and Information Sciences

Tokyo University of Agriculture and Technology

Outline

- Fine-grain power gating technology
- Geyser processor
- Linux power management
- Evaluation
- Summary

Background

- Increased power consumption of processors
  - Dynamic Power
  - **Leakage Power (Static Power)**
- Fine-grain power gating technology
  - The basic strategy is **to turn off the power to idle modules**
  - Break Even Time (BET)

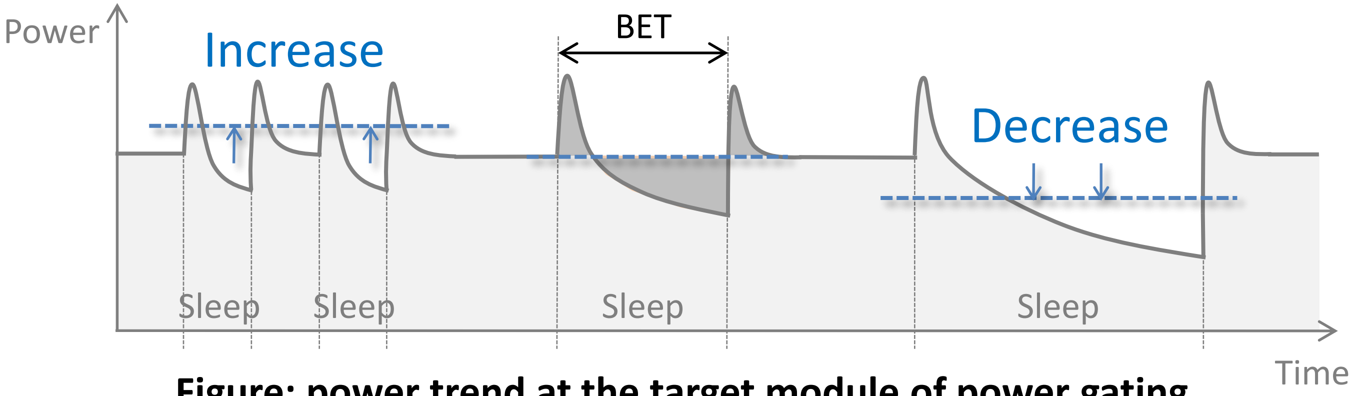


Figure: power trend at the target module of power gating

Geyser Processor

- MIPS R3000 architecture
- Fine-grain power gating technology
  - 4 functional units: **ALU**, **SHIFT**, **MULT**, and **DIV**
    - ✓Turned off automatically after execution
    - ✓Wakened up in a few nanoseconds
- Power gating control interfaces
  - Power gating control register
  - Instruction with power gating direction

Objective

- To enhance the power-saving effect of fine-grain power gating by **optimizing the timing of turning off** the power to functional units

Linux Power Management

- Linux 2.6.35 was ported to the Geyser processor
- Power Management
  - **Power gating control**: enhancing the power-saving effect by optimizing the timing of turning off the power to functional units
  - 2 types of power gating control
    - ✓Performance counter-based power gating control
    - ✓Power gating control by dynamic code adaptation

Performance Counter-Based Power Gating Control

Architectural supports

Power gating policy register

Power gating policies:

1. The target unit is turned off **every after execution**
2. The target unit is turned off **when cache misses occur**
3. The target unit is **always kept active**

12 11	10 9	8 7	6 5
DIV	MULT	SHIFT	ALU

Figure: power gating policy register

Performance counter

It counts sleep cycles: How **long/many times** does the functional unit sleep?

Basic strategy

At every scheduling intervals, a scheduler does…

1. Monitors the usage of functional units

   How long/many times did each functional unit sleep?

2. Calculates the BET miss rate

   BET miss rate: the rate of sleep cycles which does not exceed the BET

3. Decides a next power gating policy

   Based on a current policy, the BET miss rate, and a current chip temperature

Overview

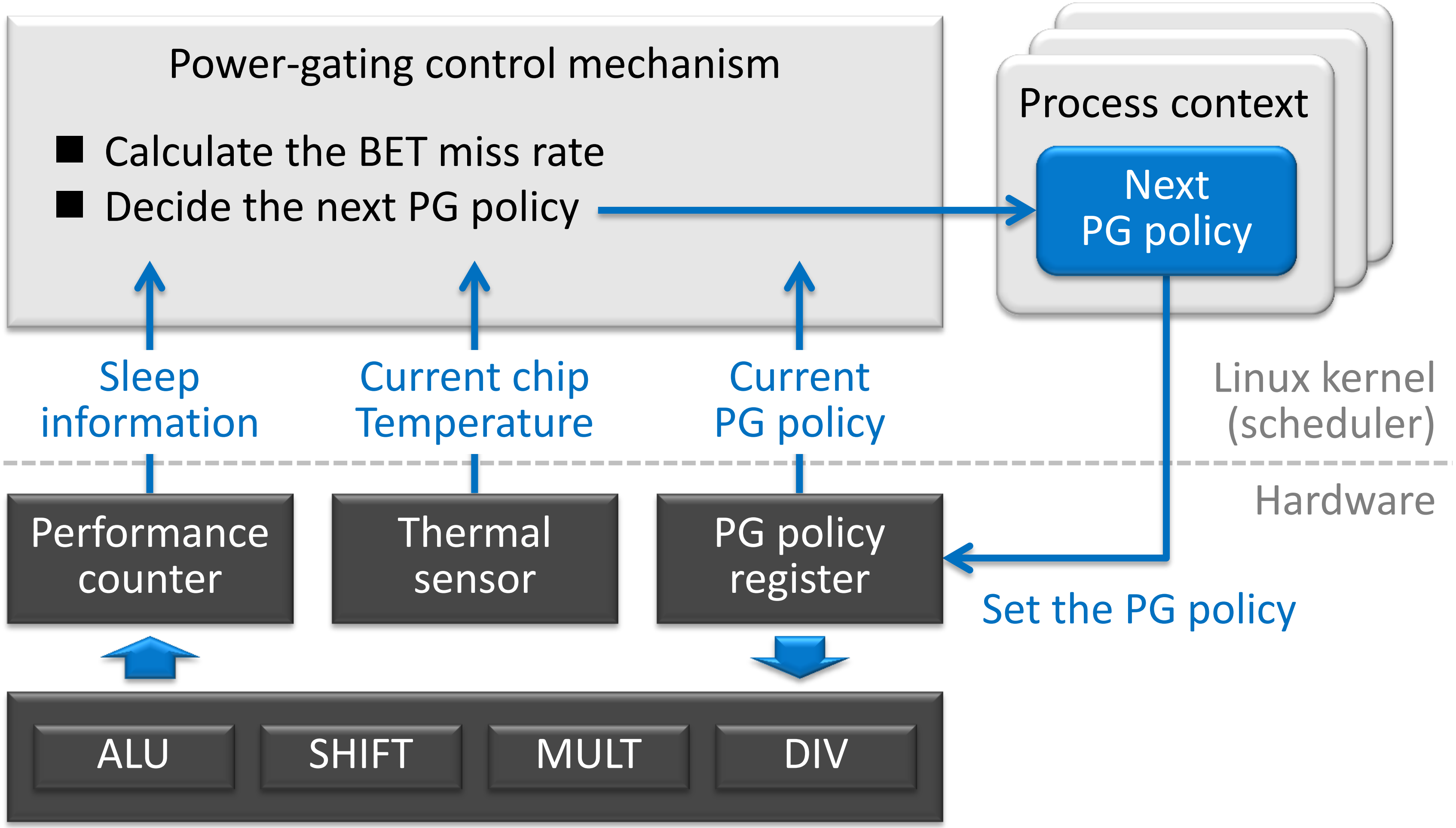


Figure: an overview of the performance counter-based PG control

Implementation

- Additional features
  - Process scheduler
    - ✓Monitors the usage of functional units
    - ✓Monitors the chip temperature
    - ✓Calculates the BET miss rate
    - ✓Decides power gating policies
  - Process context
    - ✓Contains power gating policies
  - Others
    - ✓Performance counter controller

Evaluation

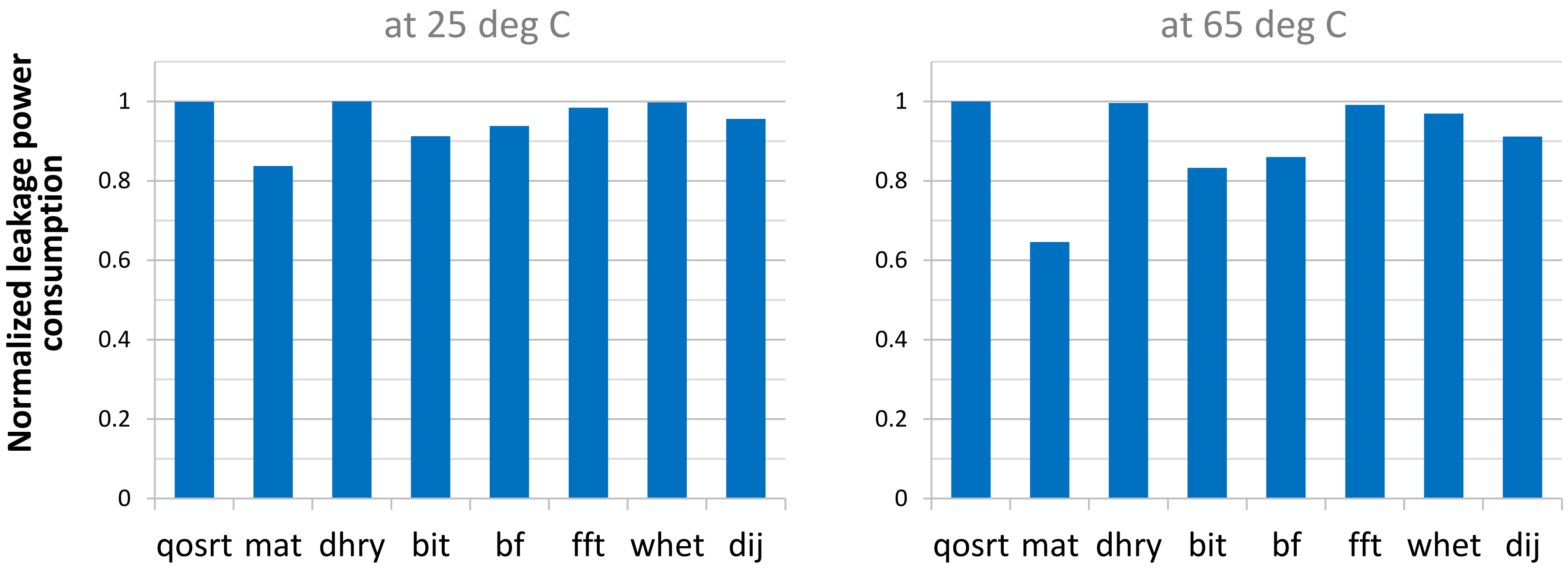
■ Experimental Environment

- Built on a FPGA evaluation platform (Xilinx ML501)
  - ✓Linux is working on the ASIC but still under test
- CPU : Geyser (MIPS R3000)
- OS : Linux-2.6.35
- Benchmarks : MiBench

Picture: an evaluation platform

Results

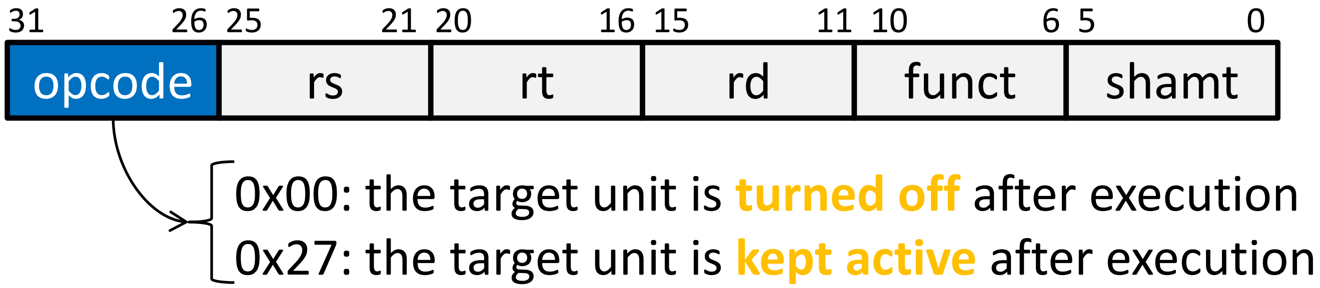
- Reduction of leakage power consumption
  - 7% on Avg. and 35% on Max.
  - with 6% performance overhead
    - ✓Can be cut by changing the specification of the counter



Power Gating Control
by
Dynamic Code Adaptation

Architectural supports

- Instruction with power gating direction
  - Extended MIPS R type instruction
  - Power gating direction: **whether the target unit is turned off or not**



31 26	25 21	20 16	15 11	10 6	5 0
opcode	rs	rt	rd	funct	shamt

0x00: the target unit is **turned off** after execution
0x27: the target unit is **kept active** after execution

Figure: instruction format

Basic Strategy

1. Static code analysis

   A compiler predicts idle cycles of functional units

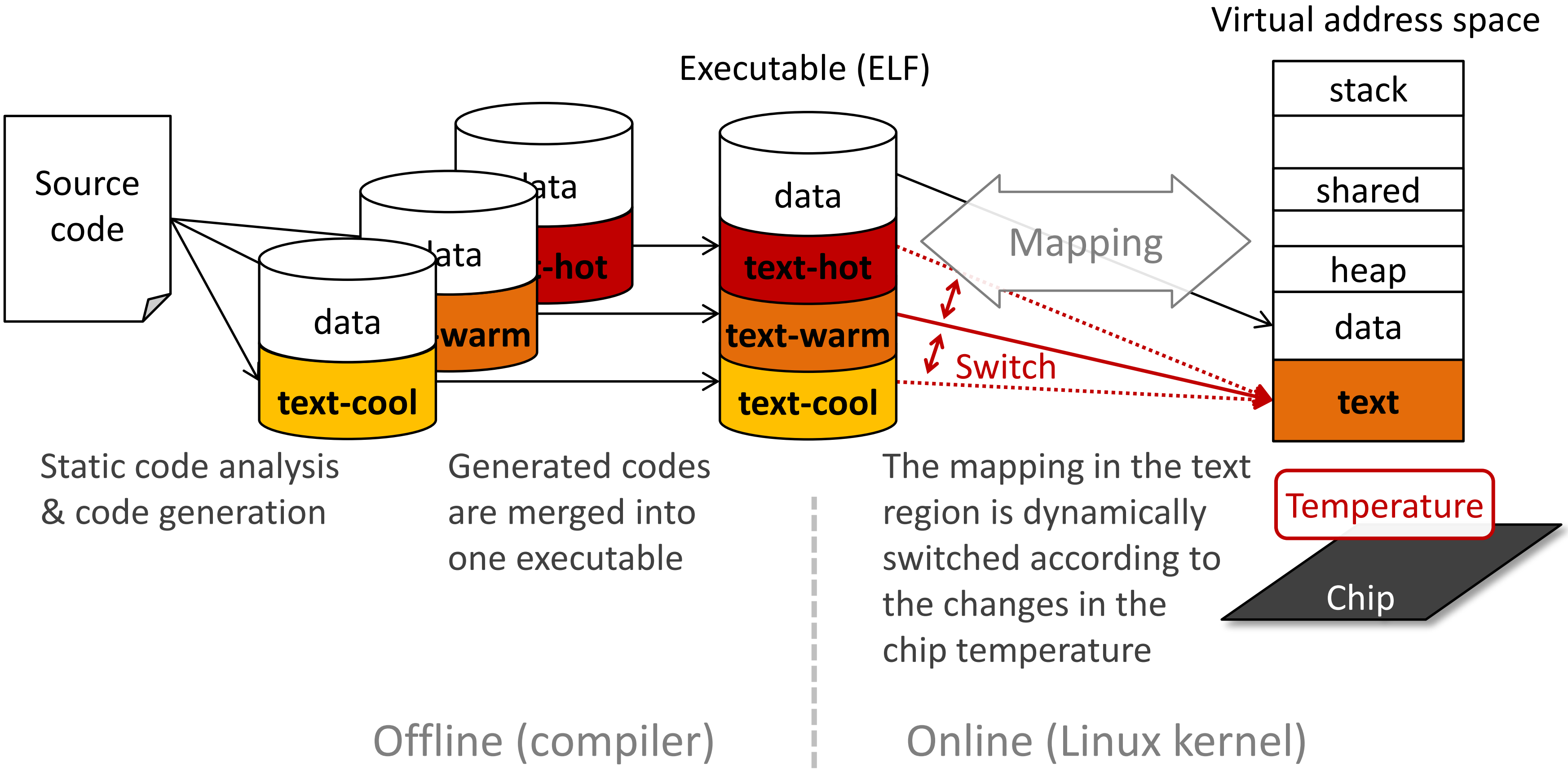
2. Code generation

   A compiler adds power gating directions into instructions and generates energy-optimized code for some chip temperatures

3. Dynamic code adaptation

   The Linux kernel monitors the chip temperature and selectively executes an appropriate code for runtime chip temperature

Overview



Static code analysis & code generation

Generated codes are merged into one executable

The mapping in the text region is dynamically switched according to the changes in the chip temperature

Offline (compiler)

Online (Linux kernel)

Figure: an overview of the PG control by dynamic code adaptation

Implementation

- Additional features
  - Process scheduler
    - ✓Monitors a chip temperature
    - ✓Decides whether switching the code or not
    - ✓Changes a mapping of text region
  - ELF loader
    - ✓Deals with multi text segment
  - Others
    - ✓An data structure which contains temperature information of a text segment

Evaluation

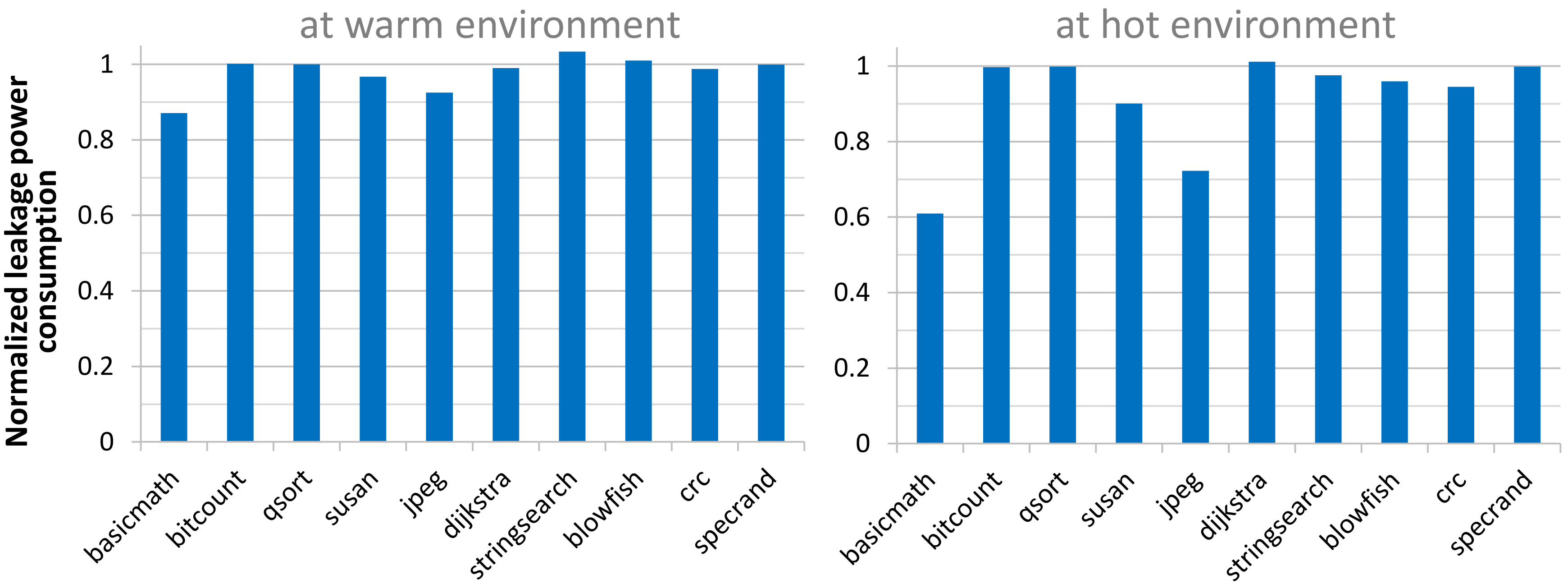
■ Experimental Environment

- Built on a FPGA evaluation platform (Xilinx ML501)
  - ✓Linux is working on the ASIC but still under test
- CPU : Geyser (MIPS R3000)
- OS : Linux-2.6.35
- Benchmarks : MiBench

Picture: an evaluation platform

Results

- Reduction of leakage power consumption
  - 9% on Avg. and 39% on Max.
  - With 0.3% performance overhead



Summary

- Geyser: a low power processor with fine-grain power-gating technology
  - The target is functional units
- Linux power management
  - Performance counter-based power gating control
    - ✓Saves ~35% of leakage power consumption
  - Power gating control by dynamic code adaptation
    - ✓Saves ~39% of leakage power consumption