Software transformations to reduce instruction memory power consumption using a loop buffer

Tom Vander Aa
Murali Jayapala
Francisco Barat
Geert Deconinck

ESAT,
K.U.Leuven, Belgium

Henk Corporaal
Francky Catthoor

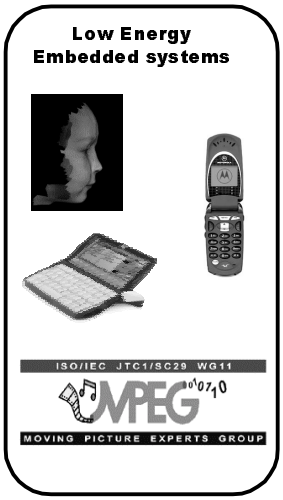
IMEC, Leuven,
Belgium

Overview

- Introduction and motivation
  - Low power
  - Multimedia applications and the loop buffer
- Optimization problem
  - Map loops onto the loop buffer
  - Energy function
- Software transformations
  - Goal
  - Examples and results
- Conclusions and future work
  - Related issues

ESAT/ACCA

Context



Low Power Embedded Systems

- Battery operated (low energy)
  - ⇨ 10-50 MOPS/mW
- Small
- Low cost
- Flexible
- Multimedia Applications
  - ⇨ Video, audio, wireless
  - ⇨ High performance
    - ✓ 10-100 GOPS
    - ✓ real-time constraints

ESAT/ACCA

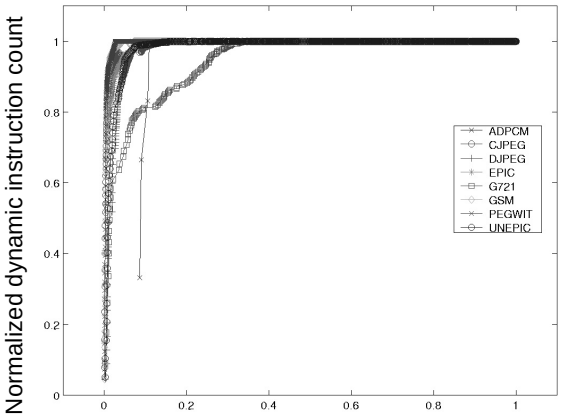
Context

Embedded processors

- Power Breakdown
  - ❑ 43 % of power in on-chip Memory
    - ⇨ StrongARM SA110: A 160MHz 32b 0.5W CMOS ARM processor
  - ❑ 40 % of power in internal memory
    - ⇨ C6x, Texas Instruments Inc.

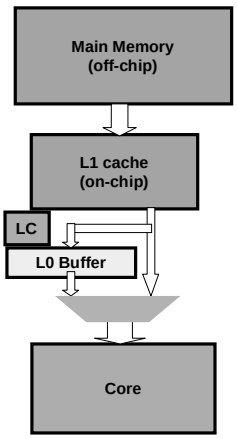
25-30% of power in Instruction Memory

Instruction Behavior



⇒ Within a program, few basic blocks or instructions take up most of the execution time ($IC_{dynamic}$)

Loop Buffer

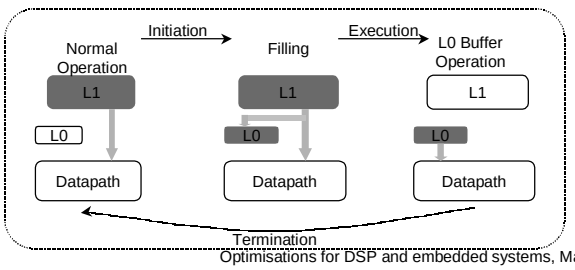


R.S. Bajwa et.al, "Instruction Buffering to Reduce Power in Processors for Signal Processing", IEEE Trans VLSI Systems, vol 5, no 4, 1997

L. H. Lee et.al, (M-CORE), "Instruction Fetch Energy Reduction Using Loop Caches for Applications with Small and Tight Loops", ISLPED 1999

- L0 Buffer: Buffer (< 1KB) + Local Controller (LC); [no tags]
- L0 / L1 access latency: 1 cycle
- Used only for *specific program segments* (innermost loops)
- Software control:

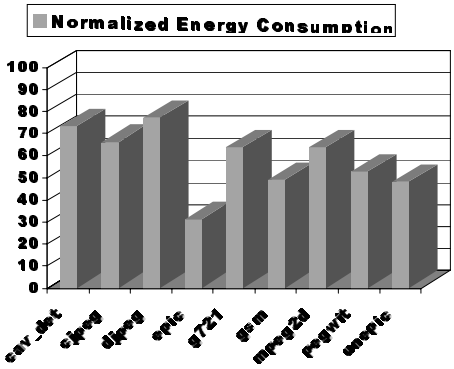
Special instruction *(lbon, sbb)* to map program segments to L0 buffer



ESAT/ACCA

Software controlled L0 buffers

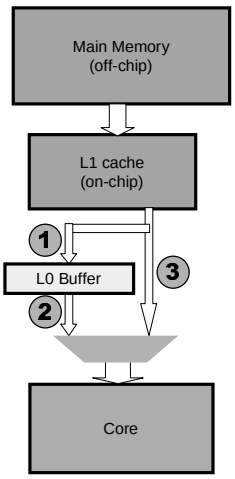
- Assumed Architecture
  - MIPS 4000 ISA
  - Single Issue Processor
  - L1 Cache
    - 16KB Direct Mapped
  - Loop Buffer (2KB)
    - Depth = 128 instructions
    - Width = 16 Bytes
- Tools
  - Simplescalar 2.0
  - Wattch Power estimator



- Loops with less than 128 instructions were hand-mapped onto the loop buffer

ESAT/ACCA

Energy Behaviour



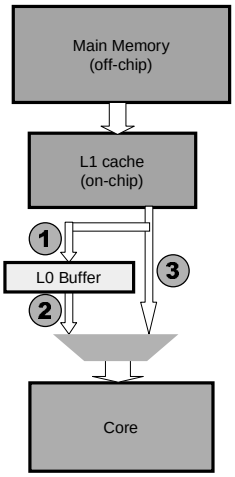
- Energy of the instruction memory hierarchy (lb and il1)

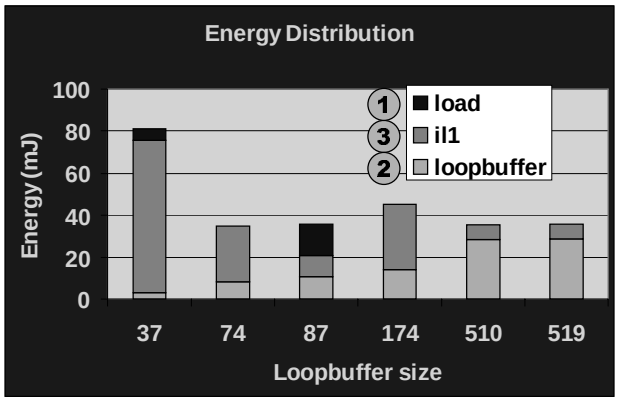
$$E = \sum_{l \in mapped}^{I} \underset{(1)}{N_{loads}(l)} \times E_{access}(il1) + \underset{(2)}{N_{exec}(l)} \times E_{access}(lb) + \sum_{l \in unmapped}^{I} \underset{(3)}{N_{exec}(l)} \times E_{access}(il1)$$

  - $E_{access}(lb) < E_{access}(il1)$
  - $E_{access}(lb)$ depends on the mapped loops
  - $N_{loads}(l)$ depends on the Control Flow Graph

ESAT/ACCA

Energy Behaviour

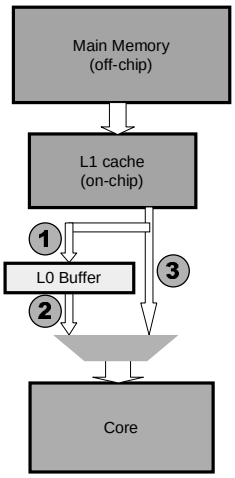




ESAT/ACCA

Code Transformations

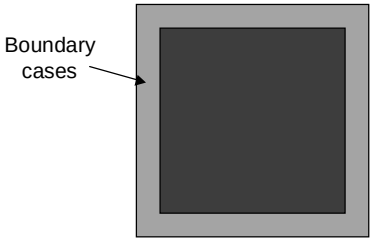
- goal: reduce energy
  - ❑ reduce the loop sizes
  - ❑ reduce accesses to il1
- to reduce the active code size
  - ⇨ loop peeling
  - ⇨ loop splitting
  - ⇨ code hoisting
  - ⇨ loop creation
- to reduce loads from l1 cache
  - ⇨ factorisation
  - ⇨ loop interchange
- Classical code optimisations borrowed from HPC



ESAT/ACCA

Loop Peeling (I)

- Loop peeling
  - Removes boundary cases
- Benefits
  - Reduces the loop size
  - Reduces the needed loop buffer size
- Example:
  - Cavity Detector
    - Medical imaging application

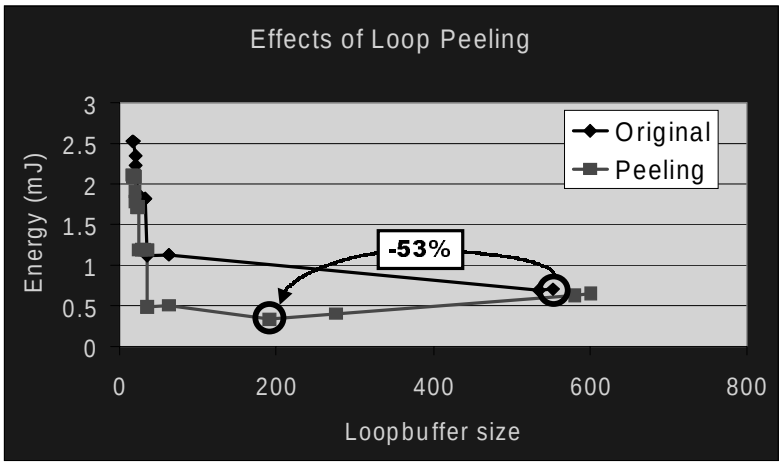



```
FOR i = 0 to no_of_rows
   FOR j = 0 to no_of_columns
      IF i < 10 THEN
         /* special case */
      END IF
      /* normal case */
   END FOR
END FOR
```

```
FOR i = 0 to 9
   FOR j = 0 to no_of_columns
      /* special case */
   END FOR
END FOR
FOR i = 10 to no_of_rows
   FOR j = 0 to no_of_columns
      /* normal case */
   END FOR
END FOR
```


ESAT/ACCA

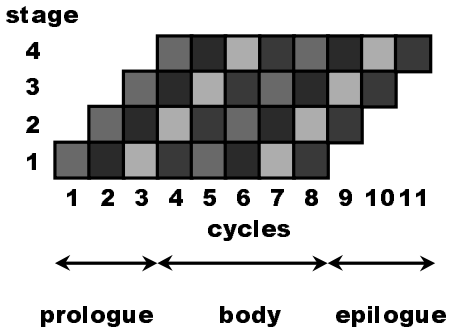
Loop Peeling (II)



ESAT/ACCA

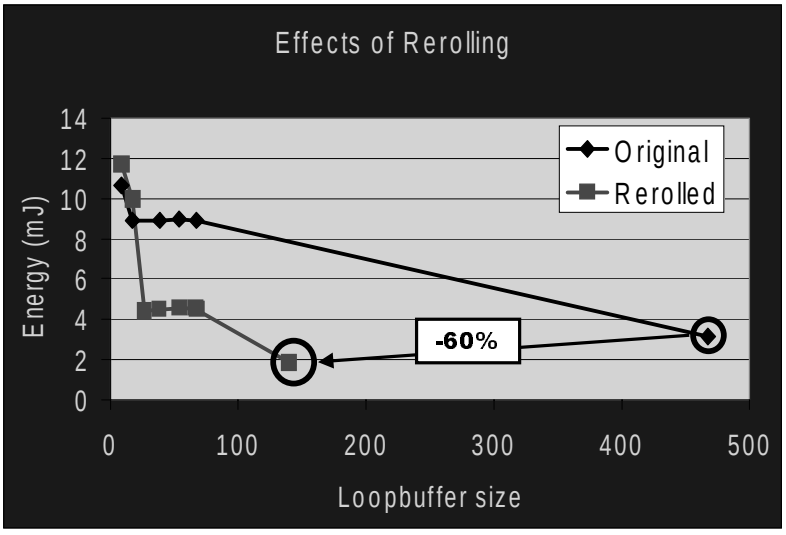
Loop Rerolling (I)

- Make trade off between
  - Software pipelining
    - No code duplication inside the loop
    - Limited by recurring dependencies
  - Loop Unrolling
    - Code size explosion
    - Better ILP, if recurring dependencies
- Benefits
  - Reduces the loop buffer size
  - Reduces E_{access}(lb)
- Example
  - Blowfish encryption



ESAT/ACCA

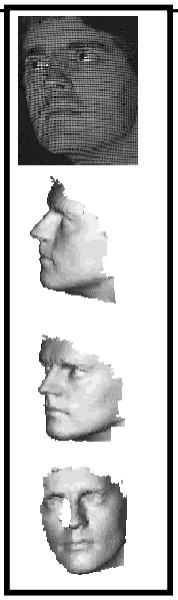
Loop Rerolling



ESAT/ACCA

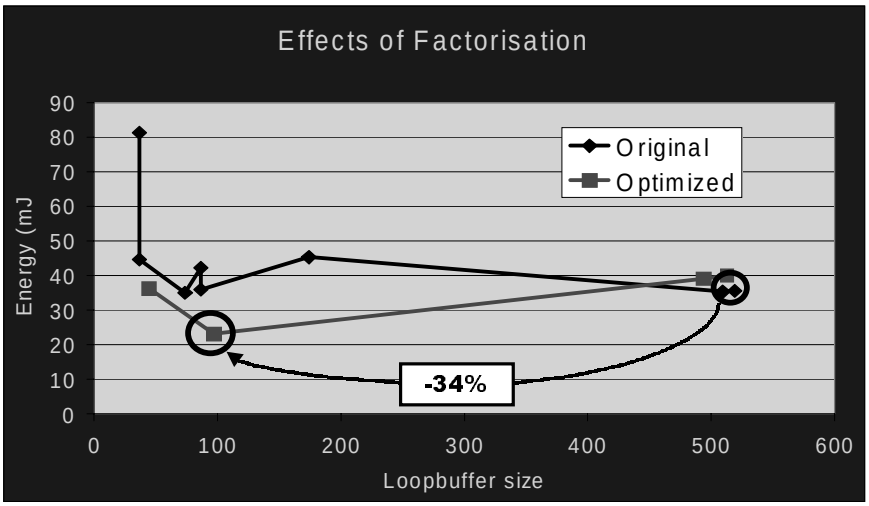
Factorization (I)

- Factorization
  - Finds common code in the loop buffer
    - Make them equal with if/then/else constructs
    - Create a function call
- Benefits
  - Reduces $N_{loads}(l)$
- Example
  - 3D Image Reconstruction Algorithm



ESAT/ACCA

Factorization (II)



ESAT/ACCA

Related Work

- Data memory hierarchy power
  - ⇨ Should also be handled
  - ⇨ Methods and tools are known (DTSE)
- Compiler support for loop buffers
  - ❑ Code layout [Bellas, et al]
  - ❑ Complementary transformation [Sias, et al]
- Impact of the transformations
  - ❑ On data memory
    - ⇨ Unchanged, since accesses are not reordered (yet)
  - ❑ On processor performance
  - ❑ Evaluation framework
    - ⇨ Enhanced VLIW compiler
    - ⇨ Based on Trimaran framework
    - ⇨ Wattch power models

ESAT/ACCA

Conclusion

- L0 Buffer Organization
  - ❑ Multimedia applications have high locality in small program segments
  - ❑ An additional small L0 buffer should be used
  - ❑ Standard implimentations: loop buffer not effciently used
- Software transformations are needed
  - ❑ Loop peeling
  - ❑ Loop rerolling
  - ❑ Factorisation
  - ❑ Loop splitting
- Up to 80% of energy reduction in the instruction memory hierarchy
  - ❑ Evaluation framework
- What's next?
  - ❑ Automation of the transformations