Power Optimization at Compilation Level

Yi Zhao

CSE231 Class Presentation

outline

• Introduction: Power Optimization in VLSI Chip Design
• Optimization Techniques at Different Compilation Level
• Example: Low-Power Design for Real-Time Systems
Needs for Low Power Design

- **Portable systems**
  - notebook, cell phone and palm-tops
  - long battery lifetime
  - system cost and weight limit

- **Thermal Consideration**
  - packaging: ceramic vs plastic
  - cooling requirement

- **Environmental Concerns**
  - "Green Products"

- **Reliability**
  - Electromigration
  - IR drops on supply line
Low Power Optimization Hierarchy

- High-level optimization saves much more power and is faster.
- High-level optimization achieve lower power design at the expense of accuracy
- High-level optimization becomes more and more popular

## Compilation-level Power Optimization

<table>
<thead>
<tr>
<th>Power Optimization in Compilation</th>
<th>Source Code</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer Replacement</td>
<td>Conditional Execution</td>
<td></td>
</tr>
<tr>
<td>Expression replacement</td>
<td>Switch Statement/TLU</td>
<td></td>
</tr>
<tr>
<td>Switch Statement/TLU</td>
<td>Function Design</td>
<td></td>
</tr>
<tr>
<td>CSE</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Instruction Scheduling</td>
<td>Switching Reduction</td>
<td></td>
</tr>
<tr>
<td>Loop optimizations</td>
<td>Register relabeling</td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Heuristics from CMOS power model \( P = CV^2 fN \):
- less transition
- less computation and addressing activity … … .
Instructional-level Optimization I

• **Instruction Level Power Model** *

\[ E_p = \sum_i (B_i \times N_i) + \sum_{i,j} (B_{i,j} \times N_{i,j}) + \sum_k E_k \]

- \( E_p \) - energy cost of a program \( p \)
- \( B_i \) - base cost of an instruction \( i \); \( N_i \) - the number of instructions
- \( O_{i,j} \) - inter-instruction cost; \( N_{i,j} \) - the number of consecutive two instructions
- \( E_k \) - run-time overhead for each cache miss or pipeline stall

• **Experiment calibrates the instruction costs.**

• **Instruction level power model enables the evaluation of software power consumption:**
  - less computation and addressing activity
  - instruction order
  - cache miss and pipeline stall

Loop Optimizations

• Loop Unrolling*

**Operation:** Unroll the loop
**Reason:** Decrease the number of control operation & overhead instruction.


• Software Pipelining*

**Operation:** Software Pipelining
**Reason:** Decrease the number of stalls by fetching instruction from instruction sequence.

Source Code-level Optimization I

• **Variable Replacement with Integer**
  
  **Operation**: Replace variables with integers
  
  **Reason**: Integer is more energy efficient (18.3% than char*).
  
  **Condition**: Chars are used

• **Copy Pointer (Address)**
  
  **Operation**: Store first reference into a variable
  
  **Reason**: Save multiple memory lookup (33.9%).
  
  **Condition**: Pointer chain is used

**Example:**

```c
void InitPos1(Object *p) {
    p->pos->x = 0;
    p->pos->y = 0;
    p->pos->z = 0; }
```

```c
void InitPos1(Object *p) {
    Point3 *pos = p->pos;
    pos->x = 0;
    pos->y = 0;
    pos->z = 0; }
```

Source Code-level Optimization II

• Switch Statement vs. Table Lookup
  Operation: Replace Switch Statement with Table Lookup
  Reason: Table Lookup is more energy efficient (52.9%*).
  Example:
  Return “EQ\ONE\OCS\0”+3*cond;

• Function Design
  Operation: Pass function call in registers rather than on stack
  Reason: Function call overhead is high (4 cycles*, 72.3%).
  Condition: small number of augments in the function
  Example:
  Int func1(int a, int b)
  { if (a>b)
    return(func2(a-b));
    else
    return(func2(b-a));
  }

Low Power Design for Real-time Systems* I

- Low power design for real-time system
  - power restriction: energy efficient program
  - real-time restriction: functional + temporal

- Models for optimization
  - Power optimization: instruction-level power model
  - Real-time execution:
    - real-time system model
      consist of a set of periodic real-time programs
      define elemental unit (EU) as the smallest execution unit
      describe each program as an EU graph(node: EU  edge: preced. & commun. relationship)
    - relative timing model
      relative timing model vs. absolute timing model
      Define start time and deadline (more flexibility for optimization) Fix execution time interval

Low Power Design for Real-time Systems II

• Real-time Scheduling Problem with Power Optimization
  – A sequence of compiled EUs
  – Relative timing constraint imposed on EUs
  – Energy consumption values for all the instructions and their pairs

• The Problem is NP-Complete: Heuristic Solution

• Proposed Scheduling Algorithm
  – to generate an initial calendar
  – to test if a change to the execution time of an EU is acceptable
  – to adjust a calendar dynamically when the execution time is changed
Low Power Design for Real-time Systems III

• Base Power Reduction
  – simple instruction vs. complex instruction
  – to use the same function but less power consumption instruction
  – problem inputs: data dependence, instructions (simple and complex), and system resource utilization table

Example:

<table>
<thead>
<tr>
<th>Clock 4</th>
<th>I_{10}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock 3</td>
<td>\text{I}_6 \text{I}_7 \text{I}_8 \text{I}_9</td>
</tr>
<tr>
<td>Clock 2</td>
<td>\text{I}_3 \text{I}_5 \text{I}_4</td>
</tr>
<tr>
<td>Clock 1</td>
<td>\text{I}_1 \text{I}_2</td>
</tr>
</tbody>
</table>

\[ f_1, f_2, f_3, f_4, f_5, f_6 \]

C = f_1 + f_4
D = f_2 + f_3 + f_6
E = f_3 + f_5

P(D_1) + P(I_8) < P(E_2) + P(I_6) + P(I_9)
Low Power Design for Real-time Systems IV

- **Inter-instruction Power Reduction**
  - Reordering a sequence of instructions in an EU without changing semantics and functionality of the EU.
  - Define the relationship graph (RG) to describe the data dependence, precedence, and resource synchronization relationship.
  - Given the RG of an EU, all the possible combination of instruction sequence can be generated by using topological sorting.
  - It is a NP-complete problem.
  - Proposed algorithm:
    - examine instructions one by one according to the initial condition
    - the minimum cost is kept in each iteration
    - if the changing of the position of instruction reduces the total power, a new sequence with less cost is found
    - the algorithm terminate when all instructions are examined
Table 1. The comparison of the averaged energy cost when NumInstSet = 64.

Table 5. The reduction rates of our algorithm for different sizes of instruction sets.

Experiment Result
Discussion

- Optimizations are based on instruction level model. Thus accurate cost evaluation is important.
  
  Current model (Tiwari) vs Square current model

- Can and how can we apply source code optimization techniques to real-time system?
Conclusion

- Power optimization at compilation level can save significant power than low level optimizations.
- Power optimization can be done at instruction level and source code level by various optimization techniques.
- Quantitative power optimization is based on base cost model (instruction cost model).
- Problem is hard and most methods are heuristic.