Software Power Estimation and Optimization

Vivek Tiwari
Intel Corporation

(Research done at Princeton University 1994-95)


EE290a
UC Berkeley
April 8th, 1999
Power reduction through software

- **Software determines CPU power consumption**
  - Why not modify s/w to reduce power!

- Also, growing role of software in electronic systems
- **Embedded systems:** functionality partitioned between
  - **Software:** application-specific s/w on dedicated processor
  - **Hardware:** application-specific logic
    - Examples: car electronics, cameras, cellular phones etc.

- Main thrust so far has been on optimizing hardware

- **Software can determine overall power consumption**
Energy and Power

• Physical Definitions

\[ P_{\text{avg}} = I_{\text{avg}} \times V_{cc} \]
\[ E = P_{\text{avg}} \times T \]
\[ T = N \times t \]
\[ E = I_{\text{avg}} \times V_{cc} \times t \]

- \( P_{\text{avg}} \): Average power
- \( I_{\text{avg}} \): Average current
- \( V_{cc} \): Supply voltage
- \( E \): Energy consumption
- \( T \): Time taken
- \( N \): Number of cycles
- \( t \): Cycle time

Example:

<table>
<thead>
<tr>
<th>MOV DX, [BX]</th>
<th>MOV AX, CX</th>
<th>MOV AX, DX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power = 1.15 W</td>
<td>Energy = 8.6 \times 10^{-8} J</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MOV DX, [BX]</th>
<th>MOV AX, CX</th>
<th>MOV AX, DX</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOP</td>
<td>NOP</td>
<td>NOP</td>
</tr>
<tr>
<td>Power = 0.99 W</td>
<td>Energy = 22.3 \times 10^{-8} J</td>
<td>- 14% lesser power</td>
</tr>
<tr>
<td>NOP</td>
<td>NOP</td>
<td>NOP</td>
</tr>
<tr>
<td>- 158% more energy</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• Energy consumption determines battery life
How?

- Need to know current drawn by CPU
- Simulation based methods
  - simulate program execution on low level models of CPU
    - Need low level info.
    - Impossible or impractical
- Physical measurement
  - Expensive data acquisition systems
  - Simple, cheap technology
    - Digital ammeter
    - Put programs in loops
    - Get stable visual reading

Current Measurement Setup

- Power Supply
- CPU
- Rest of the system
- Integration Period of Ammeter
- Clk
Instruction level power analysis

- Can get resolution for instruction level models
  - Measure current for specially created instruction sequences
  - Provides all information needed for instruction level analysis
  - Fundamental information to quantify s/w power at higher levels

- Applied to three commercial micro-processors
  - Intel 486DX2
  - Fujitsu SPARC lite
  - Fujitsu DSP
    - ISSS 1995; IEEE Transactions on VLSI Systems, ‘96
Base Energy Costs

- First set of parameters in the models:
  - *Base energy costs of instructions*

- Measured current for loop of several instances of a given instruction
  - *Avoid stalls and cache misses: modeled separately*

- Represent power cost for basic processing needed for the instruction
## Base Energy Costs (contd.)

**486DX2**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Current (mA)</th>
<th>Cycles</th>
<th>Energy $(8.25 \times 10^{-8} \text{ J})$</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nop</code></td>
<td>276</td>
<td>1</td>
<td>2.27</td>
</tr>
<tr>
<td><code>mov dx, [bx]</code></td>
<td>428</td>
<td>1</td>
<td>3.53</td>
</tr>
<tr>
<td><code>mov [bx], dx</code></td>
<td>522</td>
<td>1</td>
<td>4.30</td>
</tr>
<tr>
<td><code>add dx, bx</code></td>
<td>314</td>
<td>1</td>
<td>2.59</td>
</tr>
<tr>
<td><code>jmp</code></td>
<td>373</td>
<td>3</td>
<td>9.23</td>
</tr>
</tbody>
</table>

**SPARClite**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Current (mA)</th>
<th>Cycles</th>
<th>Energy $(8.25 \times 10^{-8} \text{ J})$</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ld [%i0], %i0</code></td>
<td>198</td>
<td>1</td>
<td>3.26</td>
</tr>
<tr>
<td><code>st %i0, [%i0]</code></td>
<td>213</td>
<td>1</td>
<td>3.51</td>
</tr>
<tr>
<td><code>add %i0, %o0, %i0</code></td>
<td>346</td>
<td>2</td>
<td>11.40</td>
</tr>
<tr>
<td><code>mul %g0, %r29, %r27</code></td>
<td>199</td>
<td>1</td>
<td>3.28</td>
</tr>
<tr>
<td></td>
<td>198</td>
<td>1</td>
<td>3.26</td>
</tr>
</tbody>
</table>

- Sample base energy costs for 486DX2 and SPARClite
Base Energy Costs (contd.)

- Instruction pipelines are handled by default
- Costs may vary with operand and address values
  - Use averages
    - Variation < 5% for 486DX2 and SPARClite
    - Greater for DSP, e.g. 15.8-22.9 mA for LDI
- Instructions can be grouped into classes

<table>
<thead>
<tr>
<th>Fujitsu DSP Instruction Class</th>
<th>LDI</th>
<th>LAB</th>
<th>MOV1</th>
<th>MOV2</th>
<th>ASL</th>
<th>MAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current range (mA)</td>
<td>15.8-22.9</td>
<td>34.6-38.5</td>
<td>18.8-20.7</td>
<td>17.6-19.2</td>
<td>15.8-17.2</td>
<td>17-17.4</td>
</tr>
<tr>
<td>Average energy (8.25 X 10^-8 J)</td>
<td>0.160</td>
<td>0.301</td>
<td>0.163</td>
<td>0.151</td>
<td>0.136</td>
<td>0.142</td>
</tr>
</tbody>
</table>

Instruction Classes for the DSP
Inter-Instruction Effects

- Second set of parameters in the models
  - *Inter-instruction effects*

- Effect of circuit state
  - Base costs in-adequate for mixed instruction sequences
    - E.g. 486DX2
      - XOR BX, 1
      - ADD RX, DX
    - \( I_{\text{base-cost-estimate}} = \frac{(319.2 + 313.6)}{2} = 316.4 \)
    - \( I_{\text{observed}} = I_{\text{obs}} - I_{\text{est}} = 6.8 \)

  - Difference defined as *circuit state overhead*
  - Limited for 486DX2, SPARClite, 0-30MA most programs are 300-400mA
    - Impact masked by large “common” cost
  - Significant for DSP, 0-26mA, most programs are 20-60mA
    - DSP is smaller, simpler processor, with no caches
Inter-Instruction Effects (contd.)

- Other *inter-instruction effects*
  - Pipeline stalls, write buffer stalls, cache misses
    - Construct programs where effects occur repeatedly
    - Assign energy cost for a single instance

- Above effects are modeled as energy overheads
  - Multiply single instance cost by number of occurrences
  - Use as a compensating term, added to base cost
Software power estimation

- Program energy cost =
  \[ \sum_i (\text{Base}_i \times N_i) + \sum_{i,j} (\text{Ovhd}_{i,j} \times N_i) + \sum_k \text{Energy}_k \]
  
  - \(N_i\): Number of times instruction \(i\) is executed
  - \(\text{Base}_i\): Base energy cost of \(i\)
  - \(\text{Ovhd}_{i,j}\): Circuit state overhead when \(i\), \(j\) are adjacent
  - \(\text{Energy}_k\): Energy overhead of stalls, cache misses

- Program power cost = Energy cost / execution time

- Circuit state overhead
  - Use a constant value 486DX2, SPARClite
  - Table for DSP due to greater variation
Estimation example: 486DX2

<table>
<thead>
<tr>
<th>Program</th>
<th>Base Cost (mA)</th>
<th>Cycles</th>
<th>Block</th>
<th>Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>main:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mov bp, sp</td>
<td>285.0</td>
<td>1</td>
<td>B1</td>
<td>1</td>
</tr>
<tr>
<td>sub sp, 4</td>
<td>309.0</td>
<td>1</td>
<td>B2</td>
<td>4</td>
</tr>
<tr>
<td>mov dx, 0</td>
<td>309.8</td>
<td>1</td>
<td>B3</td>
<td>1</td>
</tr>
<tr>
<td>mov word ptr -4[bp], 0</td>
<td>404.8</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L2:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mov si, word ptr -4[bp]</td>
<td>433.4</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add si, si</td>
<td>309.0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add si, si</td>
<td>309.0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mov bx, dx</td>
<td>285.0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mov cx, word ptr _a[si]</td>
<td>433.4</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add bx, cx</td>
<td>309.0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mov si, word ptr _b[si]</td>
<td>433.4</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add bx, si</td>
<td>309.0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mov dx, bx</td>
<td>285.0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mov di, word ptr -4[bp]</td>
<td>433.4</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>inc di</td>
<td>297.0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mov word ptr -4[bp], di</td>
<td>560.1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmp di, 4</td>
<td>313.1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jl L2</td>
<td>405.7(356.9)</td>
<td>3(1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L1:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mov word ptr _sum, dx</td>
<td>521.7</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mov sp, bp</td>
<td>285.0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jmp main</td>
<td>403.8</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Base Cost_{\text{PROGRAM}} = \sum \text{Base Cost}_{\text{BLOCK } i} \times \text{Instances}_{\text{BLOCK } i}

Estimated base current =

Base Cost_{\text{PROGRAM}} / 72 = 369.0 mA

Final estimated current = 369.0 + 15.0 = 384.0 mA

Measured Current = 385.0 mA

- Similar experiments in 486DX2 and SPARClite accurate to within 3%
Software energy estimation flow

1. Assembly/Machine Code
2. Base Cost Table
3. Determination of Basic Blocks
4. Stall Analysis
5. Basic Block Cost Estimate
6. Execution Profiling
7. Global Cost Estimate
8. Cache Penalty Est. (Cache Simulation)
9. Final Cost Estimate
Software power/energy optimization

- Ignored due to lack of practical analysis techniques
  - Deficiency overcome

- Fundamental information to guide:
  - Higher level decisions
    - H/W -S/W partitioning, choice of algorithm
  - Development of automated tools
    - Compilers, code schedulers

Software power/energy optimization comes for free!

- No increase in system cost or complexity
- Performance improves or remains the same

- General as well as specialized techniques
Reduction in memory operations

- **Memory operands have high energy costs**
  - 486DX2: Register operands - 280 mA - 320 mA
  - Reads (cache hits) > 420 mA, writes even more expensive

- **Paradigm for energy efficient s/w:** *reduce memory ops*

- **During code generation:** *utilize registers effectively*
General observations

- *Instruction reordering to reduce switching*
- No significant impact for 486DX2, SPARClite
  - Low variation in circuit state overhead
- Valid for the Fujitsu DSP [Lee et al., 1995]
  - Automated technique based on *list scheduling*
  - Schedule instructions based on overhead cost table and dependencies
  - Up to 14% energy reduction for some actual DSP applications
  - Performance not affected
Energy cost driven code generation

- Change the traditional cost metrics
  - Experimented with gcc [Fraser, SIGPLAN Notices, 1991]
  - Tree mapping based code generation driven by number of cycles

```c
int i;
char a, b[10];
a = b[i] + "0";
```

(a) Program Segment

```
stm: MOVE (MEM (loc), reg) = 1(4);
reg: PLUS (con, reg) = 2 (3);
reg: PLUS (reg, reg) = 4 (4);
reg: PLUS (MEM (loc), reg) = 4 (4);
reg: MEM (loc) = 5 (4);
reg: con = 6 (2);
loc: reg = 7;
loc: Name = 8;
loc: PLUS (NAME, reg) = 9;
con: CONST = 10;
```

(b) A grammar for the patterns

- Changed costs to energy costs for 486DX2

(c) The IR tree representation
Energy and performance

- Have a code generator for minimizing energy
- Observation: generated code similar to before
  - Difference in current can not offset difference in cycles
- Faster instruction sequence also has lower energy
- Guideline to software design: reduce running time
- Directly utilize existing research on performance optz.
- Additional motivation for aggressive optimizations
486DX2 optimization illustration

- heapsort example
- Original code generated by lcc
- Room for further optim
  - Manual application of above ideas
  - 9% current reduction
  - 24% running time reduction
  - 40.6% energy reduction
  - 33% for circle

<table>
<thead>
<tr>
<th>Program</th>
<th>sort</th>
<th>circle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>Original</td>
<td>Final</td>
</tr>
<tr>
<td>Current (mA)</td>
<td>525.7</td>
<td>486.6</td>
</tr>
<tr>
<td>Ex. Time (ms)</td>
<td>11.02</td>
<td>7.07</td>
</tr>
<tr>
<td>Energy (10^-6 J)</td>
<td>19.12</td>
<td>11.35</td>
</tr>
<tr>
<td>Savings</td>
<td>40.6%</td>
<td></td>
</tr>
</tbody>
</table>
Processor specific optimizations

- Identify other sources of measurable power variations
  - Exploit them through specific s/w optimizations
- Dual memory loads (DSP)
  - Two on-chip memory banks
    - Dual load vs. two single loads
    - Almost 50% reduction in energy
- Instruction Packing (DSP)
  - Dual instructions: 1 cycle
  - Almost 50% lesser energy seen
- Simulated annealing based memory allocation
- Greedy packing technique (ASAP)
- Other commercial DSPs also have these functions
Further optimizations

- Swapping multiplication operands (DSP)
  - Operands (A and B) are treated asymmetrically
  - Put operand with lower weight in B
  - Examples with up to 30% current reduction
  - Table constructed to decide operand placement
    - Reduction in current with out reduction in cycles

- Software controlled power down (SPARClite)
  - Up to 22% benefit, some control overhead
    - Justifies use of hardware controlled power down

- Use of higher end of memory (SPARClite)
  - Every “0” in memory address costs 3.3 mA more
Results for Fujitsu DSP

- Programs: Std. Benchmarks + internal Fujitsu benchmarks
  - un_p: Original
    - Unpacked, no dual loads
  - m: Memory bank assignment
    - Simulated annealing
  - p: Instruction packing
    - Greedy ASAP
  - o: Instruction reordering
    - List scheduling
  - s: Multiplier operand swapping
    - Table lookup
- Up to 30% energy reduction
- Up to 17% even with just reordering and swapping
Conclusions

- The CPU power problem
  - Power is now one of the biggest concerns in CPU design

- Reducing power in high-end CPUs is hardest of all
  - Not everything is directly applicable to high performance designs
  - The need for low power innovation is also the highest here

- Looked at what has been successful so far
  - Voltage and technology scaling are biggest allies
  - *But need to design for power too*

- Architecture community cannot ignore this anymore
  - Power may limit architectural innovation

- Outlined areas for future exploration