Software Power Estimation and Optimization

Vivek Tiwari

Intel Corporation

(Research done at Princeton University 1994-95)

For further reference: "Instruction Level Power Analysis and Optimization of Software", *Journal of VLSI Signal Processing Systems*, Vol. 13, No. 2, August 1996

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

$P_{avg} = I_{avg} \times V_{cc}$
$E = P_{avg} \times T$
$T = N \times t$
$E = I_{avg} \times V_{cc} \times t$

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

Example:

```
MOV DX, [BX]
MOV AX, CX
MOV AX, DX
```

Power = 1.15 W
Energy = 8.6×10^{-8} J

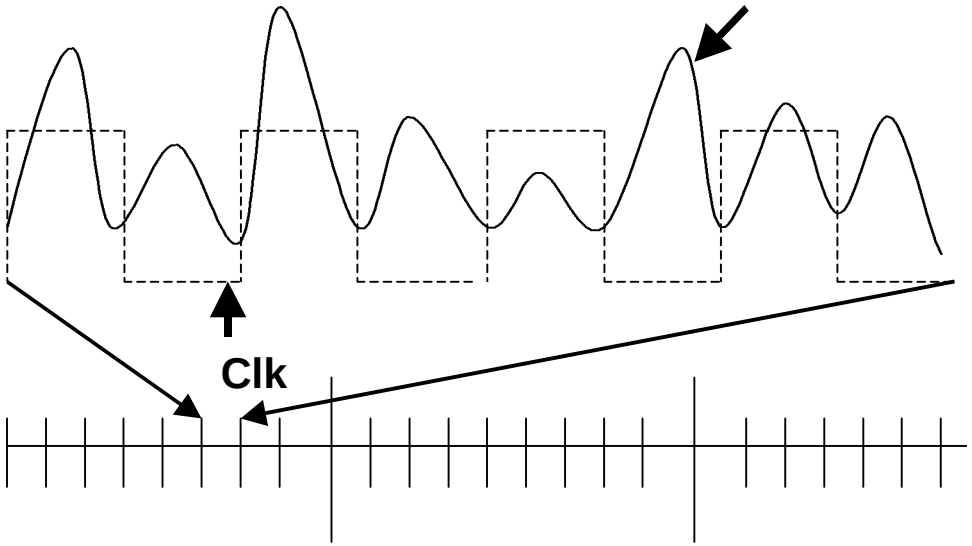
```
NOP
MOV DX, [BX]
NOP
NOP
MOV AX, CX
NOP
NOP
ADD AX, DX
NOP
```

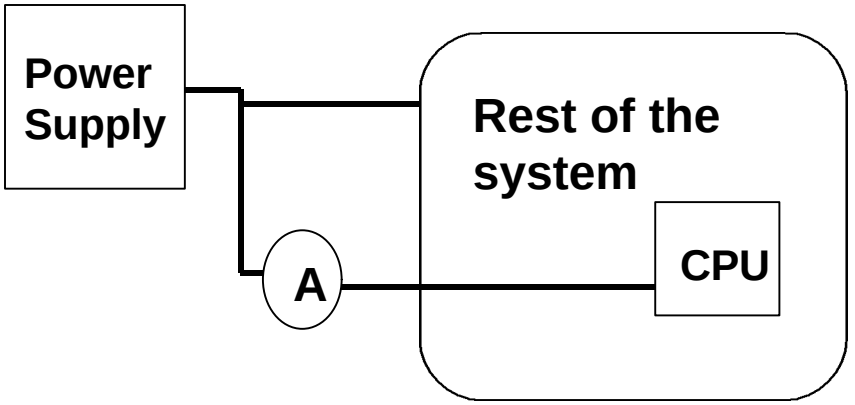
Power = 0.99 W
Energy = 22.3×10^{-8} J
- 14% lesser power
- 158% more energy

- 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

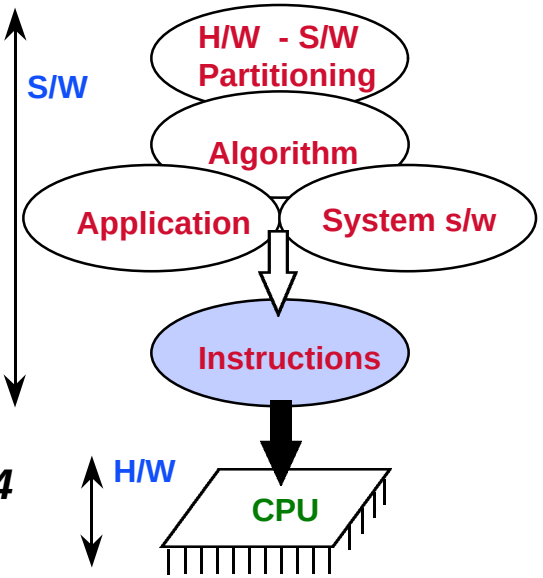
Instruction level power analysis

- 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

    ICCAD 1994; IEEE Transactions on VLSI Systems, Dec. '94
  - Fujitsu SPARClite

    Asia-Pacific DAC, Oct. 1995; VLSI Design Journal, '96
  - 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.)

- Sample base energy costs for 486DX2 and SPARClite

486DX2 →

Instruction	Current (mA)	Cycles	Energy (8.25×10^{-8} J)
nop	276	1	2.27
mov dx, [bx]	428	1	3.53
mov [bx], dx	522	1	4.30
add dx, bx	314	1	2.59
jmp	373	3	9.23

SPARClite →

Instruction	Current (mA)	Cycles	Energy (8.25×10^{-8} J)
nop	198	1	3.26
ld [%l0], %i0	213	1	3.51
st %i0, [%l0]	346	2	11.40
add %i0, %o0, %l0	199	1	3.28
mul %g0, %r29, %r27	198	1	3.26

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

Fujitsu DSP Instruction Class	LDI	LAB	MOV1	MOV2	ASL	MAC
Current range (mA)	15.8-22.9	34.6-38.5	18.8-20.7	17.6-19.2	15.8-17.2	17-17.4
Average energy (8.25×10^{-8} J)	0.160	0.301	0.163	0.151	0.136	0.142

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

    $I_{base\text{-}cost\text{-}estimate} = (319.2+313.6)/2 = 316.4$
    $I_{observed} = I_{obs} - I_{est} = \underline{6.8}$

  - ⇨ 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 =**
  - ⇨ $\Sigma_i (Base_i \times N_i) + \Sigma_{i,j} (Ovhd_{i,j} \times N_i) + \Sigma_k Energy_k$
    - N_i : Number of times instruction i is executed
    - $Base_i$: Base energy cost of i
    - $Ovhd_{i,j}$: Circuit state overhead when i, j are adjacent
    - $Energy_k$: 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

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

Block	Instances
B1	1
B2	4
B3	1
jl L2 (taken)	3
(not taken)	1

$\text{Base Cost}_{\text{PROGRAM}}$ =

Σ $\text{Base Cost}_{\text{BLOCK I}}$ * $\text{Instances}_{\text{BLOCK I}}$

Estimated base current =

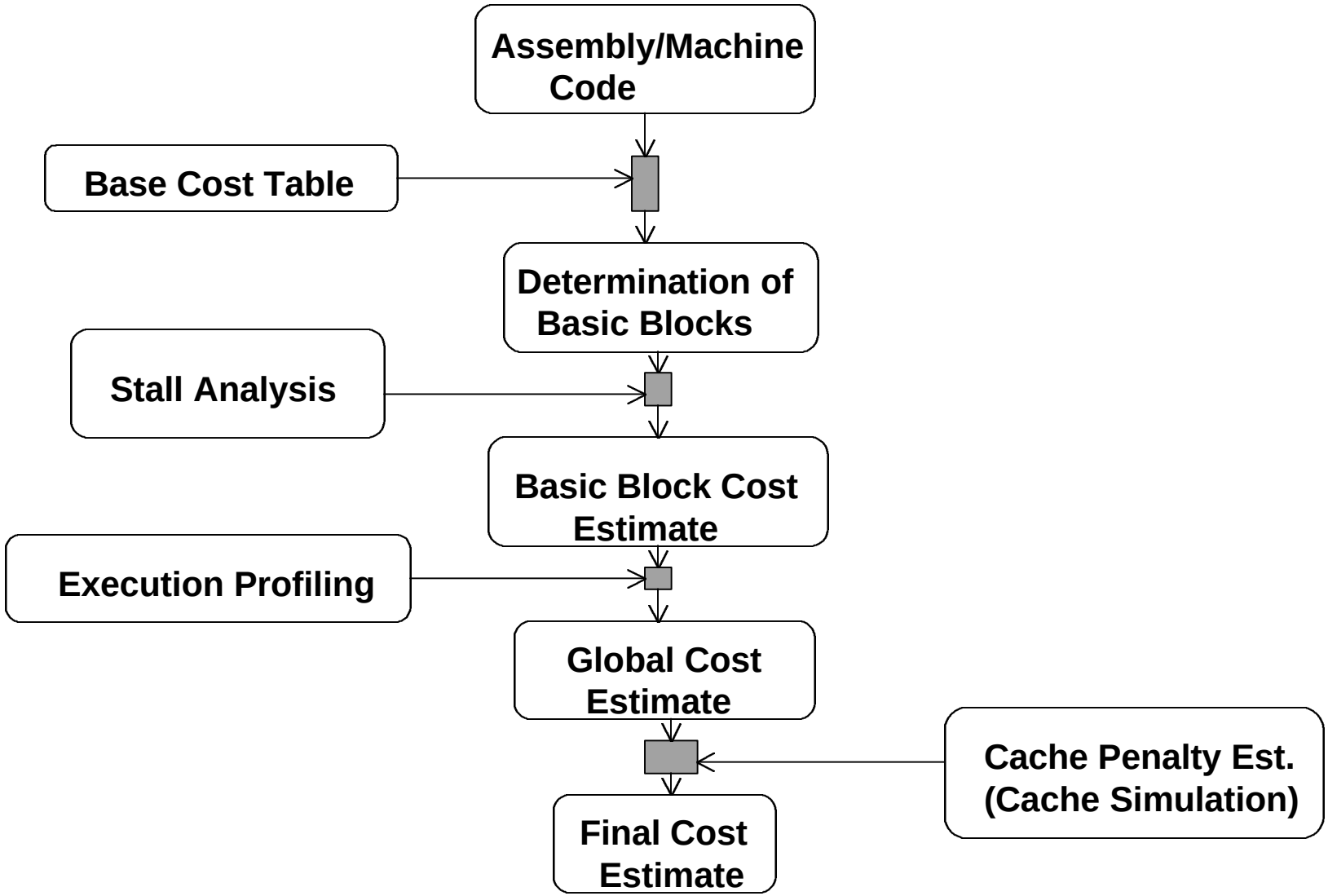
$\text{Base Cost}_{\text{PROGRAM}}$ / 72 = 369.0 mA

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



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

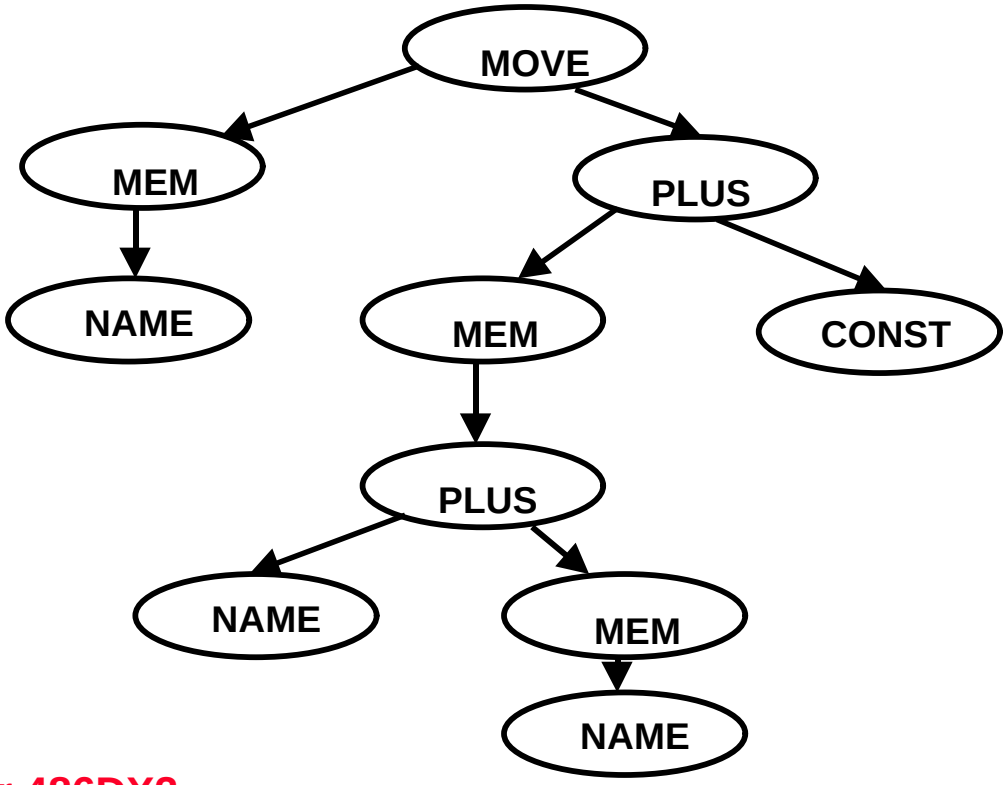
- **Change the traditional cost metrics**
  - **Experimented with lcc [Fraser, *SIGPLAN Notices*, 1991]**
  - **Tree mapping based code generation driven by number of cycles**

```
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



(c) The IR tree representation

- **Changed costs to energy costs for 486DX2**

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

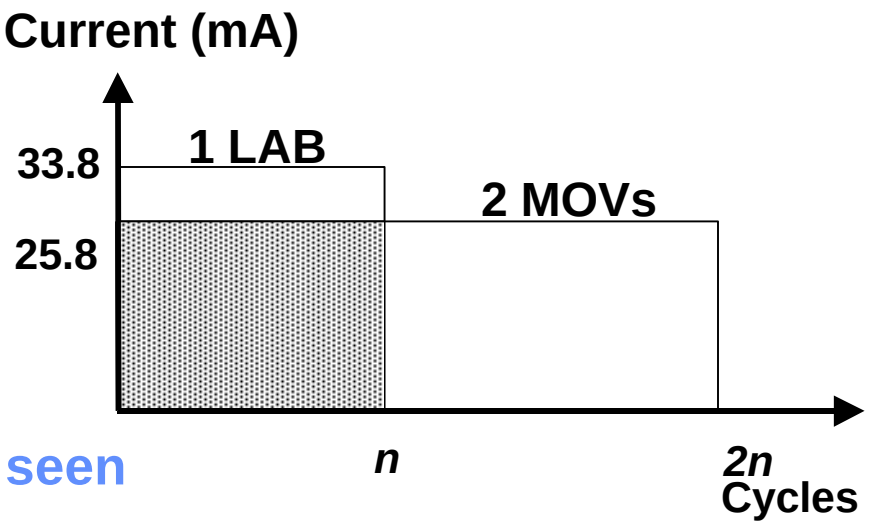
486DX2 optimization illustration

- *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*

Program	sort		circle	
Version	Original	Final	Original	Final
Current (mA)	525.7	486.6	530.2	514.8
Ex. Time (ms)	11.02	7.07	7.18	4.93
Energy (10^{-6} J)	19.12	11.35	12.56	8.37
Savings		40.6%		33.4%

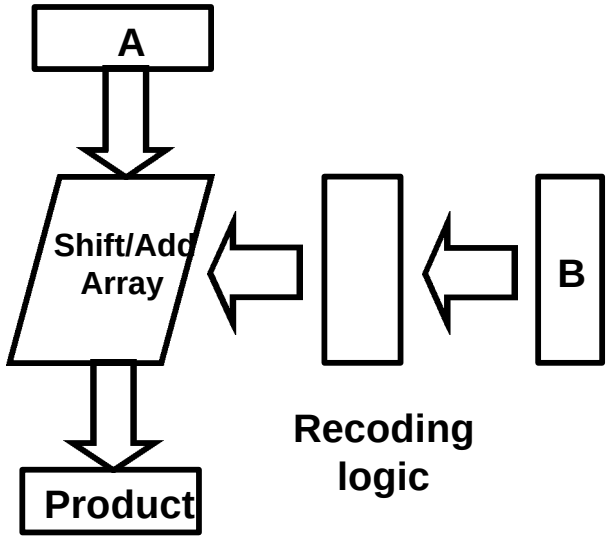
Processor specific optimizations

- 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

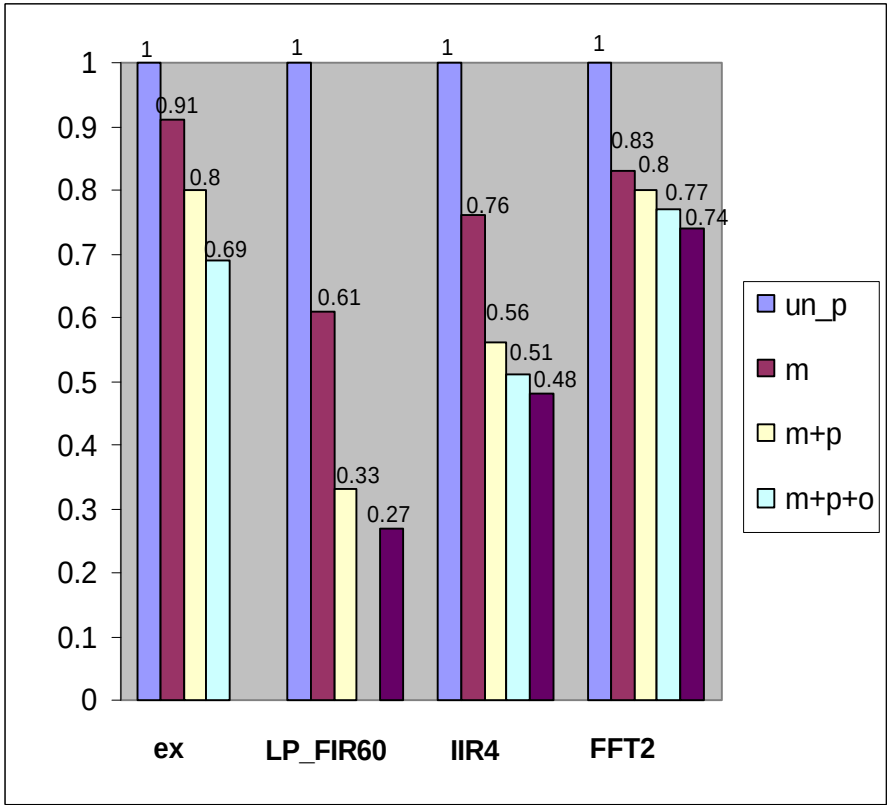


- Software controlled power down (SPARClite)
  - ⇨ Up to 22% benefit, some control overhead
    - Justifies use of hardware controlled power down
- Use of higher end of memory (SPARClite)
  - ⇨ Every “0” in memory address costs 3.3 mA more

LPD
Low Power
Design Technology

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