Software-level Power-Aware Computing

Lecture 1

Embedded System Metrics

- Some metrics:
  - *performance*: MIPS, reads/sec etc.
  - *power*: Watts
  - *cost*: Dollars
    - Nonrecurring engineering cost, manufacturing cost
  - *size*: bytes, # components, physical space occupied
  - Flexibility, Time-to-prototype, time-to-market
  - Maintainability, correctness, safety
- MIPS, Watts and cost are related
  - technology driven
  - to get more MIPS for fewer Watts
    - look at the sources of power consumption
    - use power management and voltage scaling

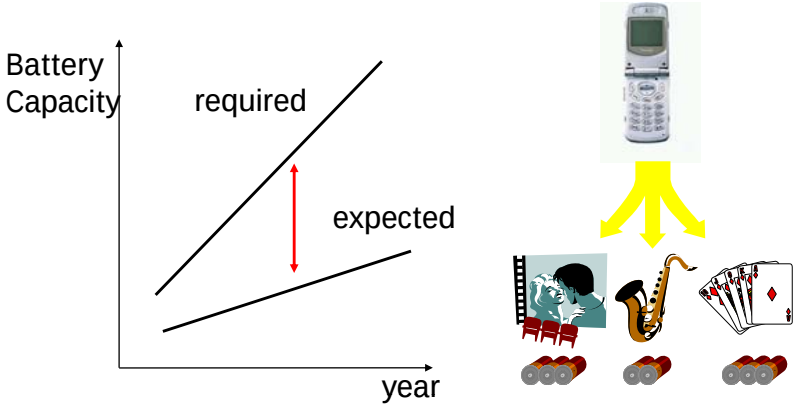
Source: MS HPL

Lecture Organizations

- Lecture 1:
  - Introduction to Low-power systems
  - Low-power binary encoding
  - Power-aware compiler techniques
- Lectures 2 & 3
  - Dynamic voltage scaling (DVS) techniques
    - OS-level DVS: Inter-Task DVS
    - Compiler-level DVS: Intra-Task DVS
    - Application-level DVS
  - Dynamic power management
- Lecture 4
  - Software power estimation & optimization
  - Low-power techniques for multiprocessor systems
  - Leakage reduction techniques

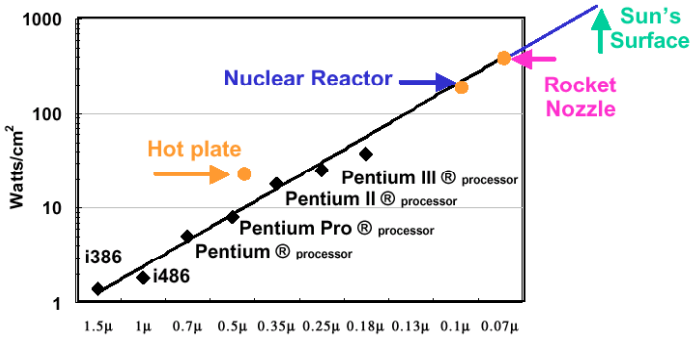
Why Low Power?

- **Limited Battery Capacity**



Why Low Power?

- **Heat Dissipation**

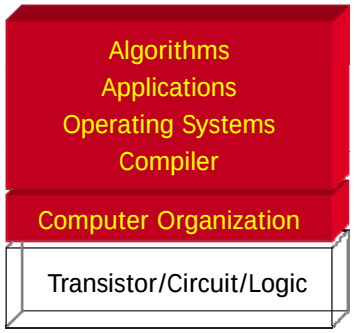


Power density getting worse

From F. Pollack

Low Power S/W Research

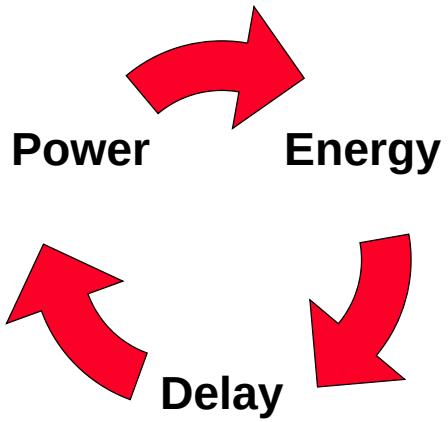
- Goal: ***Power-Aware Computing***



Why S/W Techniques for Low Power?

- S/W techniques require no H/W modifications
- Many low-power H/W techniques require concurrent engineering between H/W and S/W.
  - Examples: DPM, DVS, …
  - Efficiency of S/W Techniques are Critical for overall high energy efficiency

Basic Trade-Off



Power Consumption in CMOS

$$P_{CMOS} = P_{static} + P_{dynamic}$$

- Dynamic Power Consumption
  - Charging and discharging capacitors
- Short circuit currents
  - Short circuit path between supply rails during switching
- Leakage current
  - Leaking diodes and transistors

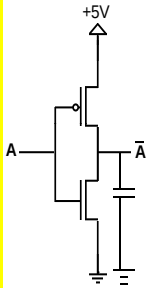
Dynamic Power Consumption

$$P_{dynamic} = K \times C_{out} \times V_{dd}^2 \times f$$

K: activity factor
C_{out}: total chip capacitance
V_{dd}: supply voltage
f: clock frequency

Reduce
1) switching activity
2) supply voltage

CMOS Inverter Example



A: high -> low

C_{out} x V_{dd}^2 drained from V_{dd} through I_p

A: low -> high

output capacitance discharged through I_N

Power Consumption Directly Depends on Switching Activity

Circuit Delay, Delta

- Delta ~ 1/f ~ $V_{dd} / (V_{dd} - V_t)^r$
  - V_t: threshold voltage
  - *r* : saturation velocity index
  - For a small V_t, f ~ $V_{dd}^{(r-1)}$

Energy Consumption, E

- E = P x T
- If power P is decreased BUT time T is increased, energy E may increase as well.

Power Consumption in CMOS

$P_{CMOS} = P_{static} + P_{dynamic}$

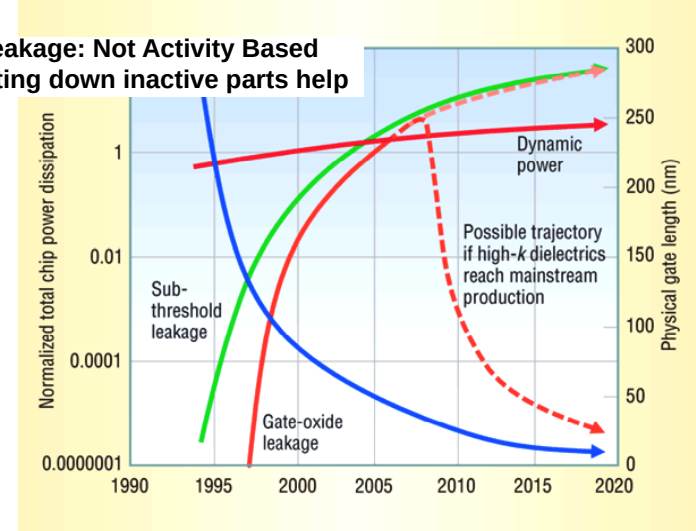
- Dynamic Power Consumption
  - Charging and discharging capacitors
- Short circuit currents
  - Short circuit path between supply rails during switching
- Leakage current
  - Leaking diodes and transistors

$P_{static} = V I_{leak}$

$= V (I_{sub} + I_{ox})$

Leakage Current

Leakage: Not Activity Based
Shutting down inactive parts help



(source: Kim et al., IEEE Computer, Dec, 2003)

Subthreshold Leakage, I_{sub}

- $I_{sub} \sim e^{(-Vt/Va)} (1 - e^{(-V/Va)})$

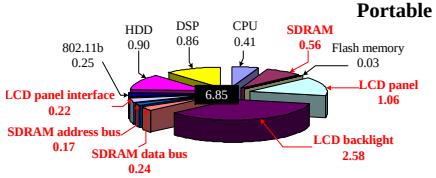
  where V_a is the thermal voltage

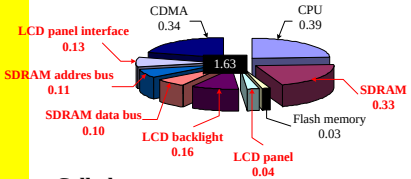
  I_{sub} ↑ -> V_a ↑ -> I_{sub} ↑ -> … Thermal Runaway!!

- How to reduce I_{sub}
  - Turn off the supply voltage
    (-) loss of state
  - Increase the threshold voltage
    (-) loss of performance

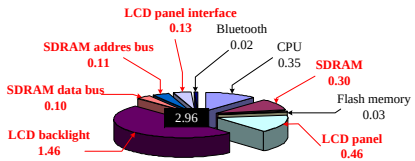
System-level Power Breakdowns [Shim 06]

Portable media player





Cell phone



Personal digital assistant

Power consumption for running a streaming video application (W)

Roadmap

- ✓ Introduction to Low-power systems
- Low-power binary encoding
- Power-aware compiler techniques

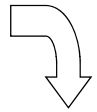
Low Power Binary Encoding

- Switching activity reduction
  - Switching activity can account for over 90% of power dissipation of CMOS circuit. [Chandrakasan *et al*, '92]
- Goal of Low power binary encoding
  - Modify the binary encoding/representation so that the switching activity is reduced.
  - Target Areas:
    - Op-code field
    - Register field
    - Bus

Register Relabeling

- Goal
  - Assign register numbers to minimize the switching activities in register field

ADD	r1 (0001)	r2 (0010)	r3 (0011)
SUB	r14 (1110)	r13 (1101)	r12 (1100)
MUL	r3 (0011)	r12 (1100)	r3 (0011)



r1 -> r12
r2 -> r2
r3 -> r0
r12 -> r1
r13 -> r3
r14 -> r8

ADD	r12 (1100)	r2 (0010)	r0 (0000)
SUB	r8 (1000)	r3 (0011)	r1 (0001)
MUL	r0 (0000)	r1 (0001)	r0 (0000)

Switching Activity
Above : 20bit
Right : 6bit

Register Relabeling

- General Approach
  - Collect the trace of register field usage information
  - Construct the Register Field Transition Graph (RFTG)
    - Nodes: registers
    - Edges: transitions
    - Edge weights: relative frequency of corresponding edges
  - Find new register number assignment that minimize the total bit changes.

Problem Formulation

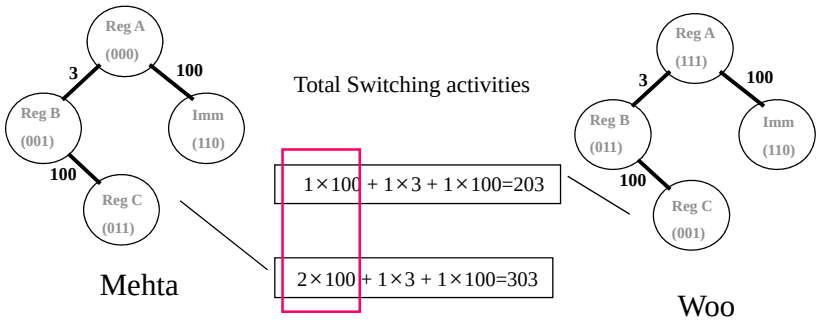
- A register field transition graph (RFTG)
  - $G = (V,E,w)$: $V = V_{reg} \cup V_{imm}$
- A relabeling function
  - find $f: V_{reg} \rightarrow V_{reg}$, to minimize the following cost metric

$$P(G,f) = \sum_{\substack{e=(v_1,v_2) \\ v_1,v_2 \in V_{reg}}} w(e) \times h(f(v_1), f(v_2)) + \sum_{\substack{e=(v_1,v_2) \\ v_1 \in V_{reg} \\ v_2 \in V_{imm}}} w(e) \times h(f(v_1), v_2) + \sum_{\substack{e=(v_1,v_2) \\ v_1 \in V_{imm} \\ v_2 \in V_{reg}}} w(e) \times h(v_1, f(v_2))$$

where, $w(e)$ is the weight of edge e

Register Relabeling

- **Alternatives:**
  - **Mehta's method: Immediate field not considered**
  - **Woo's method: Immediate field considered**



Register Relabeling Heuristics

- **Relabeling is a NP-hard problem**
  - **$_N C_2$ possible choices for each pair of exchanging candidates**
- **Slack-based heuristic [Woo, '01]**
  - **Define slack value for each node (encoding)**

$$slack(v_i, v_j) = \left| h(v_i, v_j) - 1 \right| \cdot w(e)$$

  - **Exchange the encoding between most promising candidates until no more reduction is obtained from exchanging encoding**
- **Greedy method [Woo, '01]**
  - **Exchange randomly but undo the exchange if no gain is obtained from it**

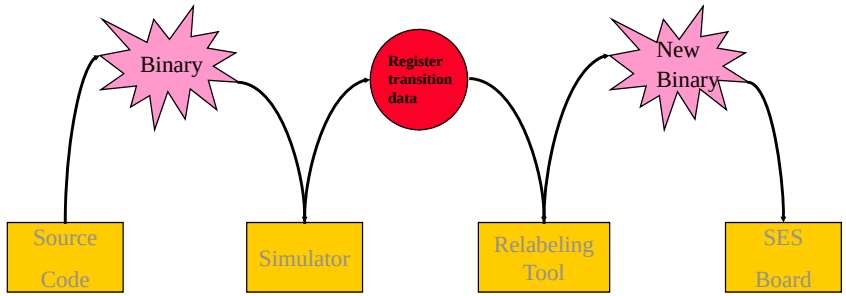
Experiment (Switching Activity)

- **Simulation environment**
  - **SimpleScalar simulator is used**
  - **Benchmark**
    - **SPEC95 int and SPEC95 fp**
    - **UTDSP benchmark**
    - **MPEG2 decoder with video only streams**
- **Result**
  - **% of switching activity**

Program	No relabeling	Mehta relabelling	New relabelling
SPEC95 geometric mean	1.0	0.96	0.91
applu	1.0	0.96	0.90
compress	1.0	0.96	0.89
gcc	1.0	0.98	0.93
UTDSP geometric mean	1.0	0.93	0.91
adpcm	1.0	0.93	0.92
histogram	1.0	0.91	0.87
turbo3d	1.0	0.96	0.93
MPEG2 decoder	1.0	0.91	0.86
Total average	1.0	0.94	0.89

Experiment (Energy Reduction)

- **Environment**
  - **Target architecture : ARM7TDMI**
  - **Measurement Tool : SES (SNU Energy Scanner) board**



Experiment (Energy Reduction)

- Effect of register relabeling at Instruction level

D/I	Inst/Data	Energy
1	e0222098	1011.946950
1	e1d380f8	1064.029688
1	e1dc00f8	906.189522
1	e0222098	1077.069527
1	e1d300fa	1062.568274
1	e1dc80fa	944.328240
1	e0222098	1010.444222
1	e1d300fc	1065.396874
1	e1dc80fc	899.009519
1	e0222098	1083.144593
1	e1d300fe	1062.473957
1	e1dc80fe	950.174787
1	e0222098	1088.944918
1	e1d381f0	1052.254592
1	e1dc01f0	928.342904
1	e0222098	1075.622301
1	e2833012	1068.433202
1	e1520001	945.908777

Total energy : 480.80mJ

Relabeling

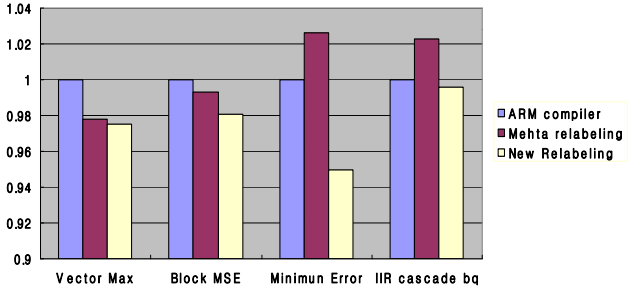
D/I	Inst/Data	Energy
1	e0288092	1100.673882
1	e1d920f8	997.394383
1	e1dc00f8	941.629293
1	e0288092	1055.144933
1	e1d900fa	982.343107
1	e1dc20fa	932.772563
1	e0288092	1080.059544
1	e1d900fc	1010.521764
1	e1dc20fc	922.440499
1	e0288092	994.389910
1	e1d900fe	1002.936386
1	e1dc20fe	902.043706
1	e0288092	1016.250972
1	e1d921f0	1006.228806
1	e1dc01f0	903.564458
1	e0288092	992.725140
1	e2899012	977.829571
1	e1580004	1155.485459

Total energy : 456.53mJ

Experiment (Energy Reduction)

- Result
  - Benchmark : TI C6000 Benchmark
    - Up to 5% energy reduction

Programs	ARM compiler	Mehta relabeling	New Relabeling
Vector Max	1	0.978	0.975
Block MSE	1	0.993	0.981
Minimun Error	1	1.026	0.950
IIR cascade bq	1	1.023	0.996



Related Work

- Register Relabeling
  - Kandemir [Kandemir et al, '00]
  - Similar to Mehta
  - Give more time efficient heuristics
- Low power opcode encoding [Kim *et al*, '99]

Conclusion

- Register relabeling with immediate values.
- Energy reduction without H/W modification
- Energy reduction with simple modification of binary codes
- Energy reduction up to 5% in CPU

References

- Chandrakasan, T. Shyng, and R. W. Brodersen, “Low power CMOS Digital Design”, IEEE Journal of Solid State Circuits, 1992
- S. Kim, and J. Kim, “Opcode Encoding for Low Power Instruction Fetch”, IEE Electronic Letters, 1999
- H. Mehta, R. M. Owens, M. Irwin, R. Chen, and D. Ghosh, “Techniques for Low Energy Software”, Proc of ISLPED, 1997
- Ching-Long Su, Chi-Ying Tsui, and Alvin M. Despain, “Low Power Architecture Design and Compilation Techniques for High-Performance Processors”, Proc of COMPCON, 1994
- M. Kandemir, N. Vijaykrishnan, M. J. Irwin, W. Ye, and I. Demirkiran, “Register relabeling : A post-compilation technique for energy reduction”, Proc of the Workshop on Compilers and Operating Systems for Low Power, 2000
- S. Woo, J. Yoon, and J. Kim, “Low-Power Instruction Encoding Techniques”, Proc of SoC Design Conference, 2001

Roadmap

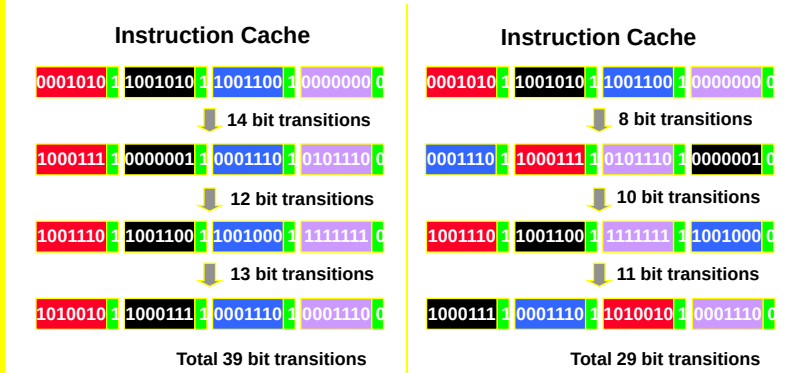
- ✓ Introduction to Low-power systems
- ✓ Low-power binary encoding
- Power-aware compiler techniques

Power-aware compiler techniques (for VLIW processors)

- Many mobile devices are designed using VLIW processors for high performance, which usually consume more power than single-issue processors.
- Operation rearrangement in VLIW instruction fetches
  - A post-past optimization technique
  - Reduce switching activities by rearranging operations in each VLIW instruction.
- Battery-aware balanced modulo scheduling
  - Effective battery utilization depends on current fluctuation
    - Less fluctuation leads to longer battery lifetime
  - Reduce power fluctuation

Operation Rearrangement in VLIW Instruction Fetches

Basic idea



(a) Before operation rearrangement (b) After operation rearrangement

The total # of bit changes are reduced by 25%

VLIW instruction encoding: uncompressed

```
IADD  /*IntU*/
|| FADD  /*FpU*/
|| LOAD  /*MemU*/
|| STORE /*MEMU*/

ISUB  /*IntU*/
|| IMUL  /*IntU*/

IADD  /*IntU*/
|| BEG   /*BrU*/
```

IntU	IntU	FpU	FpU	MemU	MemU	CmpU	BrU
IADD	NOP	FADD	NOP	LOAD	STORE	NOP	NOP
ISUB	IMUL	NOP	NOP	NOP	NOP	NOP	NOP
IADD	NOP	NOP	NOP	NOP	NOP	NOP	BEG

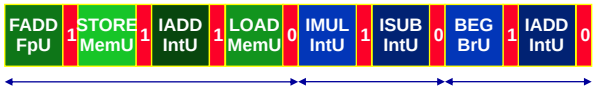
Alternative encoding

NOP	IADD	NOP	FADD	STORE	LOAD	NOP	NOP
IMUL	ISUB	NOP	NOP	NOP	NOP	NOP	NOP
IADD	NOP	NOP	NOP	NOP	NOP	BEG	NOP

VLIW instruction encoding: compressed

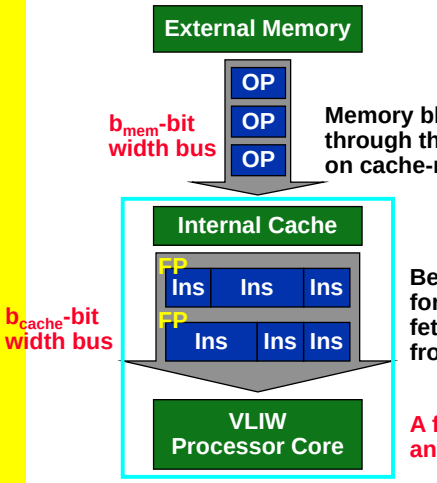
```
IADD  /*IntU*/
|| FADD  /*FpU*/
|| LOAD  /*MemU*/
|| STORE /*MEMU*/

ISUB  /*IntU*/
|| IMUL  /*IntU*/

IADD  /*IntU*/
|| BEG   /*BrU*/
```

Alternative encoding



Which encoding is the best for low-power consumption?

Machine model



Memory block is fetched from the main memory through the b_{mem}-bit width instruction bus on cache-miss.

Because of the compressed encoding format, several VLIW instructions are fetched together in a single fetch from the instruction cache.

A fetch packet consists of N operations, and $b_{mem} = b_{cache}/N$

Problem formulation

Problem

how to reorder given VLIW instructions to reduce the number of bit transitions between successive instruction fetches.

Solutions

Local Operation Rearrangement (LOR) :
each basic block is independently considered.
Global Operation Rearrangement (GOR) :
all the basic blocks are simultaneously considered.

LOR problem

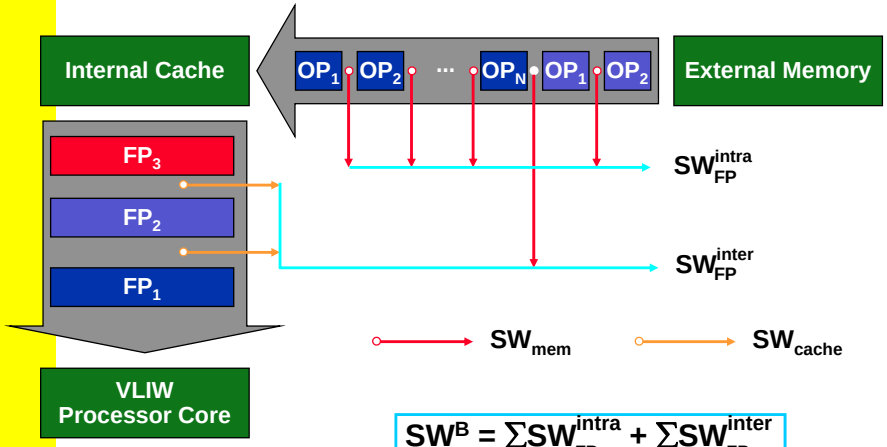
$$SW^B = SW^B_{cache} + \alpha \cdot SW^B_{mem}$$

α is the load capacitance ratio of the external instruction bus to the internal instruction bus.

SW^B_{cache} is the number of bit changes at the internal instruction bus.

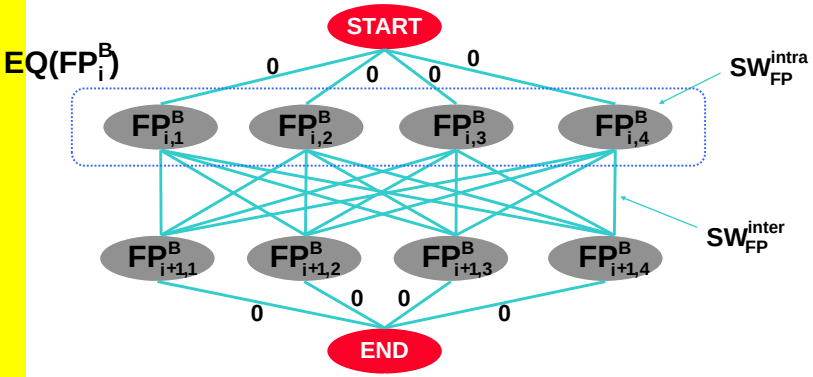
SW^B_{mem} is the number of bit changes at the external instruction bus.

LOR problem



$$SW^B = \sum SW^{intra}_{FP} + \sum SW^{inter}_{FP}$$

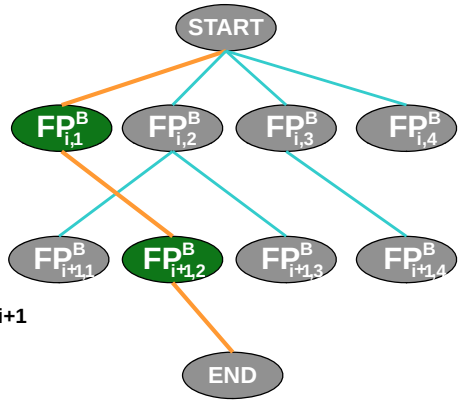
Solution for LOR



$EQ(FP^B_i)$: The set of equivalent fetch packets of FP^B_i.

Solution for LOR

- We find the shortest path from START to END, which is the solution of operation rearrangement to minimize the SW^B
- A node v_{i+1} in graph finds the node v_i through which the shortest path from START to the node v_{i+1} should pass.



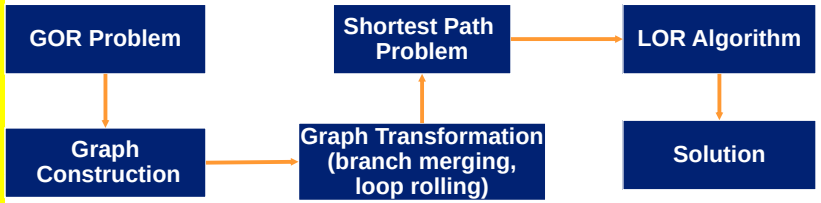
GOR problem

- All the basic blocks in a program are simultaneously considered
  - how many times each basic block is executed.
  - how often each basic block experiences cache misses.
  - how basic blocks are related each other.

$$SW^S = \sum \sum SW^{inter}_{BB}(bb_i,bb_j) + \sum SW^{intra}_{BB}(bb_i)$$

- SW^{inter}_{BB} and SW^{intra}_{BB} is represented by SW^{inter}_{FP}, SW^{intra}_{FP}, weight of each basic block, and cache miss rate.

Solution for GOR



This method may require an excessive amount of memory and cycles.

⇒ *We need a heuristic solution.*

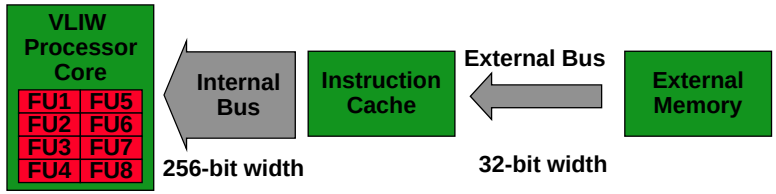
Heuristic for GOR

- All the basic blocks are not equally treated.
  - Basic blocks with larger effects on the total switching activity are more thoroughly reordered than ones with smaller effects.
- Not all the equivalent basic blocks in $EQ(bb_i)$ are tried to find an optimal solution.
  - Only N_{cand} equivalent basic blocks are created and included in graph.

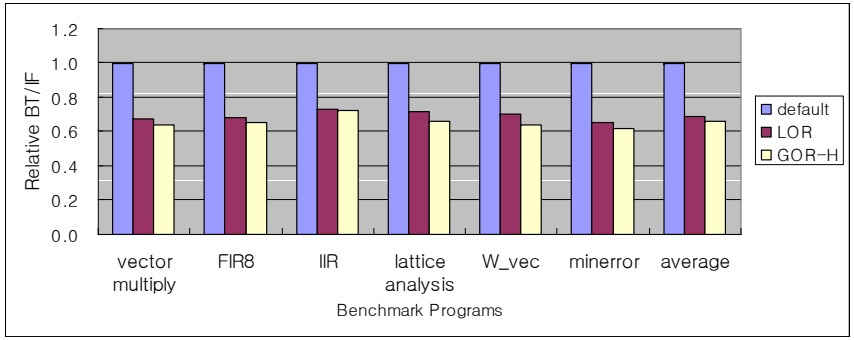
Experiment

TMS320C6201

- Fixed-point DSP
- VLIW processor that can specify eight 32-bit operations in a single 256-bit instruction.
- Use a compressed encoding



Experimental results



For our benchmark programs, the bit transitions was reduced by 34% on an average.

Conclusion

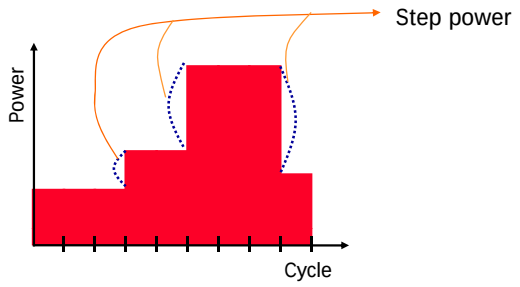
- Described a post-pass optimal operation rearrangement method for low-power VLIW instruction fetch.
  - The switching activity was reduced by 34% on an average.
- Future works
  - The phase-ordering problem between the operation rearrangement and other compiler optimization steps.
  - Operation rearrangement problem in super-scalar processors.

Battery-Aware Modulo Scheduling for VLIW Processors

Power Fluctuation

- In VLIW processors, power fluctuation significantly depends on the parallel schedule generated by compilers
- Closely related to battery-lifetime
  - As current fluctuation becomes larger, battery lifetime becomes shorter
- Battery-aware balanced modulo scheduling
  - Traditional power-unaware modulo scheduling algorithm is modified so that the power fluctuation is reduced
  - No performance loss nor additional energy consumption

Power Fluctuation



- step power
  - differences in the instantaneous power between consecutive cycles
  - Inductive noise $L \bullet di/dt$ (voltage glitch induced at power/ground buses) $\Rightarrow$ timing & logic errors

Step-power Aware Compilation

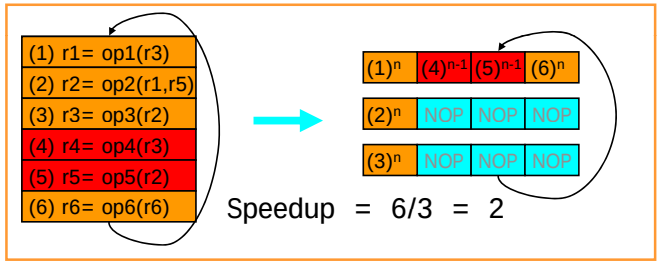
- Programs spend most of the execution time in loops
  - Optimizing compilers (for VLIWs) perform software pipelining to shorten the execution time of loops
- The traditional power-unaware software pipelining can be modified so that the power fluctuation is reduced
- Quite effective in reducing the power fluctuation
  - The compiler can fully control the usage of all the FUs in a VLIW processor

VLIW machine model & power model

- MIPS-like integer pipeline, UltraSPARC-like FPU pipeline
- 8-issue VLIW model
  - 1 integer ALU , 2 load/store unit , 1 integer MPY/DIV
  - 2 FP ALU , 2 FP MPY/DIV
- 16-issue VLIW model : # of each FU is doubled
- Use instruction-level power model
  - ignore inter-instruction effect
- The proposed algorithm can be easily extended to work with more accurate power model
  - does not depend on a particular power model

Software pipelining

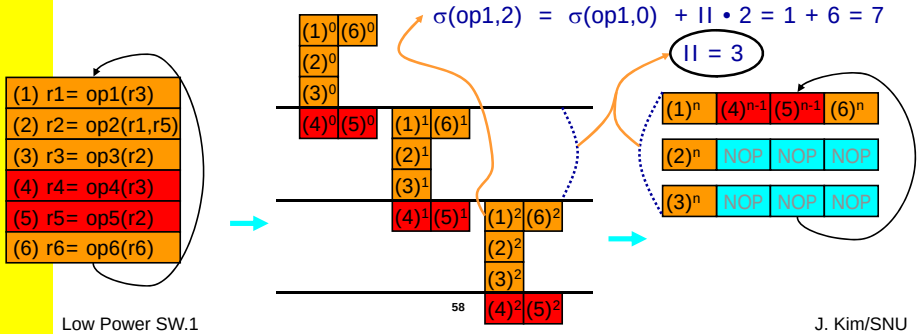
- Aggressive fine-grained loop scheduling technique
  - For VLIW processors (e.g., Intel IA-64, TI C6x, …)
- Essentially, equivalent to retiming technique used in VLSI synthesis
- Overlaps the execution of multiple iterations in a pipelined fashion



- Modulo scheduling is one of the scheduling algorithms for implementing software pipelining

Traditional modulo scheduling formulation

- II : the length of an iteration of parallelized loop body
- σ(op,i) : execution cycle when the instance of operation op in iteration i is begin to execute
- Periodicity constraint : σ(op,i) = σ(op,0) + II • i
- Goal : find the minimum II and a corresponding schedule σ(op,0) for each v subject to dependence constraint and resource constraint

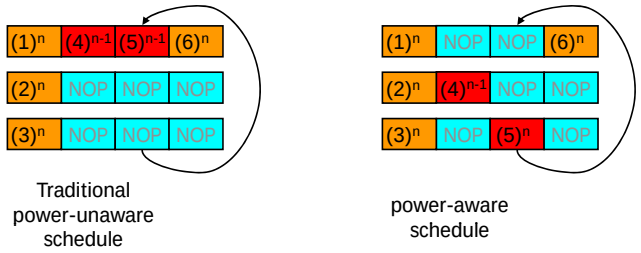


Power-aware modulo scheduling

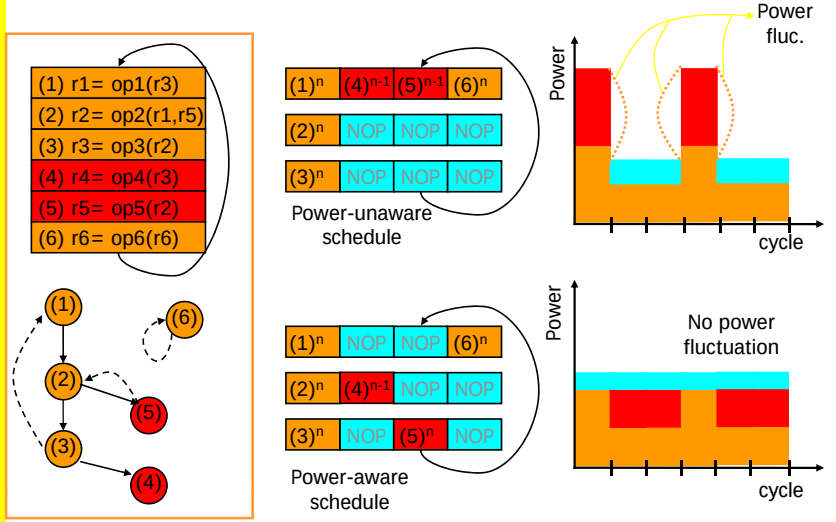
- Our goal

Given the II (found by traditional MS algorithm), find the schedule such that the power consumption distribution is as **flat** as possible

- No performance loss; no additional energy consumption



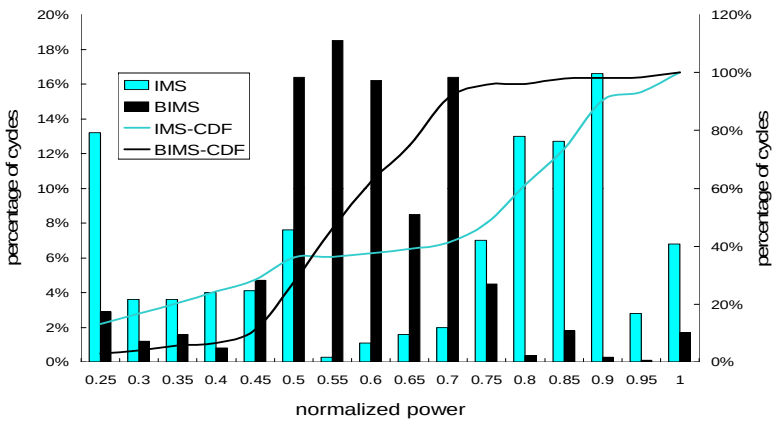
Cycle-by-cycle power dissipation



Experiment setting

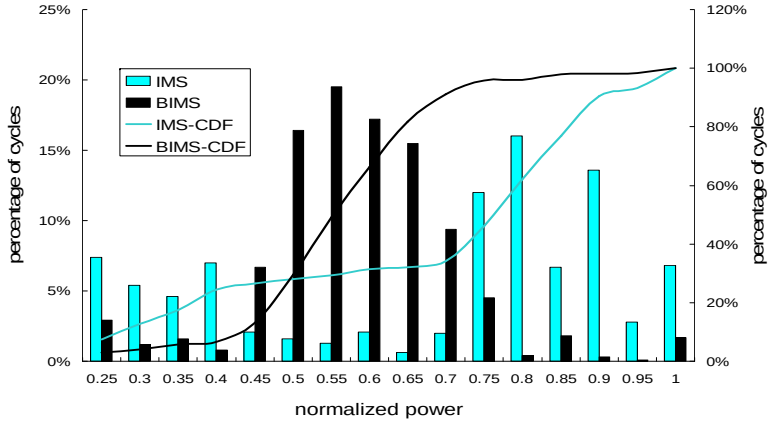
- Base algorithm
  - Iterative Modulo Scheduling (IMS) [Rau, MICRO'94]
  - Outperforms most of other MS algorithms
- Our power-aware algorithm : Balanced IMS (BIMS)
- Battery lifetime model [Pedram, DAC'99]
- SPEC95 FP benchmark programs
- SPARC-based VLIW testbed [Moon, MICRO'97]
  - 8 & 16-issue VLIW

Power distribution for 8-issue VLIW



Battery lifetime: 28% increased

Power distribution for 16-issue VLIW



Battery lifetime: 31% increased

Conclusion

- Quite effective in reducing the power fluctuation
  - The compiler can fully control the usage of all the FU in a VLIW processor
- Battery lifetime increases significantly
  - 29% for 8-issue VLIW
  - 31% for 16-issue VLIW

References

- **D. Shin and J. Kim, “Operation Rearrangement for Low Power VLIW Instruction Fetch”, Proc of DATE, 2001**
- **H.-S. Yun and J. Kim, “Power-Aware Modulo Scheduling for High-Performance VLIW Processors”, Proc of ISLPED, 2001**