

Minimizing Power Consumption in CPUs

A survey of two papers

Paul Kohlbrenner
George Mason University
CS 803
Spring 2006

Introduction

Why are we concerned about power?

- Mobile and embedded devices are generally constrained by their need for power.
- Power is projected to become one of the limiting factors in future high performance CPUs.

Introduction

From: *The 2005 International Technology Roadmap for Semiconductors*, Grand Challenges section:

... extrapolation from the Overall Roadmap Technology Characteristics and the System Drivers chapter shows that high-performance MPU power consumption **significantly exceeds** the high-performance single-chip package power limits established in the Assembly and Packaging chapter,

Introduction

Table 6a Power Supply and Power Dissipation—Near-term Years

Year of Production	2005	2006	2007	2008	2009	2010	2011	2012	2013
DRAM 1/2 Pitch (nm) (contacted)	80	70	65	57	50	45	40	36	32
MPU/ASIC Metal 1 (M1) 1/2 Pitch (nm) (f)	90	78	68	59	52	45	40	36	32
MPU Physical Gate Length (nm)	32	28	25	23	20	18	16	14	13
Power Supply Voltage (V)									
V _{dd} (high-performance)	1.1	1.1	1.1	1.0	1.0	1.0	1.0	0.9	0.9
V _{dd} (Low Operating Power, high V _{dd} transistors)	0.9	0.9	0.8	0.8	0.8	0.7	0.7	0.7	0.6
Allowable Maximum Power [1]									
High-performance with heatsink (W)	167	180	189	198	198	198	198	198	198
Maximum Affordable Chip Size Target for High-performance MPU Maximum Power Calculation	310	310	310	310	310	310	310	310	310
Maximum High-performance MPU Maximum Power Density for Maximum Power Calculation	0.54	0.58	0.61	0.64	0.64	0.64	0.64	0.64	0.64
Cost-performance (W)	91	98	104	111	116	119	119	125	137
Maximum Affordable Chip Size Target for Cost-performance MPU Maximum Power Calculation	140	140	140	140	140	140	140	140	140
Maximum Cost-performance MPU Maximum Power Density for Maximum Power Calculation	0.65	0.70	0.74	0.79	0.83	0.85	0.85	0.89	0.98
Battery (W)—(low-cost hand-held)	2.8	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0

[1] Power will be limited more by system level cooling and test constraints than packaging

From: *The 2005 International Technology Roadmap for Semiconductors*, Power Supply and Power Dissipation Section

Introduction

What uses power in today's chips?

- Logic functions (ALU, etc.)
- Cache
- Address support logic
- Instruction decoding and control logic

Introduction

What is inside today's chips:

- CMOS (Complimentary Metal Oxide Semiconductor)
- First developed in the early 1960's but became the dominant technology in the mid-late 1980's.
- Key advantage over earlier technologies: very low *static power*.

Introduction

- Static power – Power consumed when the device is not switching its state. (Leakage)
- Dynamic power – Power consumed when the device is in the process of switching from one state to another. (Switching & Short Circuit)

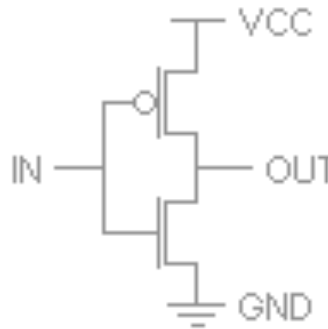


Illustration from Wikipedia

Introduction

Two papers are reviewed:

1. *Compiler Optimizations for Low Power Systems*, M. Kandemir, N. Vjaykrishnan, M.J. Irwin, Ch 10 of “Power Aware Computing”, 2002.
2. *The Energy Impact of Aggressive Loop Fusion*, YK. Zhu, G. Magklis, M. Scott, C. Ding, D. Albonesi, IEEE Conference on Parallel Architecture and Compilation Techniques (PACT04).

Compiler Optimizations - Introduction

Overall theme of the paper:

1. Optimizing for power and optimizing for speed are not necessarily the same thing.
2. Compilers can optimize for power at several levels.

Compiler Optimizations - Introduction

Three questions posed by paper:

1. Does fast = low power?
2. What is the impact of performance-orientated optimizations on power?
3. What are the relative gains from hardware vs. software optimizations?

Compiler Optimizations - Introduction

This paper discusses reductions in:

- **Core power:** Energy consumed by the ALU, instruction and address decoding logic, the registers, and other control logic.
- **Memory power:** Power consumed by the cache and its control logic.

Compiler Optimizations – Instruction Scheduling

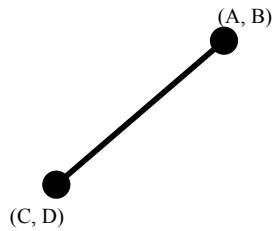
Consider Instruction Scheduling:

- Traditionally a performance optimization - try to schedule the instruction with the longest delay (forward looking).
- What if instead, we chose the instruction based on the energy cost of placing that instruction next (backward looking).

Compiler Optimizations – Scheduling Example

Problem: Find the longest line-segment in a list of segments each defined by two points:

$$\text{Equation: } S = (A - C)^2 + (B - D)^2$$



Code:

Instruction	Dest	Src 1	Src 2
1 Load	R1	A	
2 Load	R2	C	
3 Sub	R3	R1	R2
4 Mul	R4	R3	R3
5 Load	R5	B	
6 Load	R6	D	
7 Sub	R7	R5	R6
8 Mul	R8	R7	R7
9 Add	R9	R4	R8
10 Store	R9	S	

13 April 2006

Paul Kohlbrener, CS 803, Spring 2006

13

Compiler Optimizations – Scheduling Example

Compute time:

Add/Sub	1
Load/Store	3
Mul	4

Energy cost:

	Add	Sub	Mul	Load	Store
Add	1	2	3	3	3
Sub	2	1	3	3	3
Mul	3	3	1	3	3
Load	3	3	3	1	2
Store	3	3	3	2	1

Instruction	Dest	Src 1	Src 2	Start	Ready	Energy
1 Load	R1	A		0	3	0
2 Load	R2	C		1	4	1
3 Sub	R3	R1	R2	4	5	3
4 Mul	R4	R3	R3	5	9	3
5 Load	R5	B		6	9	3
6 Load	R6	D		7	10	1
7 Sub	R7	R5	R6	10	11	3
8 Mul	R8	R7	R7	11	15	3
9 Add	R9	R4	R8	15	16	3
10 Store	R9	S		16	19	3
					<u>19</u>	<u>23</u>

13 April 2006

Paul Kohlbrener, CS 803, Spring 2006

14

Compiler Optimizations – Scheduling Example

Schedule for speed:

Instruction	Dest	Src 1	Src 2	Start	Ready	Energy
1 Load	R1	A		0	3	0
2 Load	R2	C		1	4	1
3 Sub	R3	R1	R2	4	5	3
4 Mul	R4	R3	R3	5	9	3
5 Load	R5	B		6	9	3
6 Load	R6	D		7	10	1
7 Sub	R7	R5	R6	10	11	3
8 Mul	R8	R7	R7	11	15	3
9 Add	R9	R4	R8	15	16	3
10 Store	R9	S		16	19	3
					<u>19</u>	<u>23</u>

•Savings of 4 cycles (21%)
•Savings of 2 energy units (9%)

Instruction	Dest	Src 1	Src 2	Start	Ready	Energy
1 Load	R1	A		0	3	0
2 Load	R2	C		1	4	1
3 Load	R5	B		2	5	1
4 Load	R6	D		3	6	1
5 Sub	R3	R1	R2	4	5	3
6 Mul	R4	R3	R3	5	9	3
7 Sub	R7	R5	R6	6	7	3
8 Mul	R8	R7	R7	7	11	3
9 Add	R9	R4	R8	11	12	3
10 Store	R9	S		12	15	3
					<u>15</u>	<u>21</u>

13 April 2006

Paul Kohlbrener, CS 803, Spring 2006

15

Compiler Optimizations – Scheduling Example

Schedule for energy then speed:

Instruction	Dest	Src 1	Src 2	Start	Ready	Energy
1 Load	R1	A		0	3	0
2 Load	R2	C		1	4	1
3 Load	R5	B		2	5	1
4 Load	R6	D		3	6	1
5 Sub	R3	R1	R2	4	5	3
6 Mul	R4	R3	R3	5	9	3
7 Sub	R7	R5	R6	6	7	3
8 Mul	R8	R7	R7	7	11	3
9 Add	R9	R4	R8	11	12	3
10 Store	R9	S		12	15	3
					<u>15</u>	<u>21</u>

•Savings of 3 cycles (15%)
•Savings of 6 energy units (26%)

Instruction	Dest	Src 1	Src 2	Start	Ready	Energy
1 Load	R1	A		0	3	0
2 Load	R2	C		1	4	1
3 Load	R5	B		2	5	1
4 Load	R6	D		3	6	1
5 Sub	R3	R1	R2	4	5	3
6 Sub	R7	R5	R6	6	7	1
7 Mul	R4	R3	R3	7	11	3
8 Mul	R8	R7	R7	8	12	1
9 Add	R9	R4	R8	12	13	3
10 Store	R9	S		13	16	3
					<u>16</u>	<u>17</u>

13 April 2006

Paul Kohlbrener, CS 803, Spring 2006

16

Compiler Optimizations – Bus Energy

- Idea: Relabel registers used in instructions to minimize switching on the instruction bus.

	<u>OP</u>	<u>Dest</u>	<u>Src1</u>	<u>Src2</u>
Instruction 1: Add R1, R6, R10	1011	0001	0110	1010
Instruction 2: Add R3, R1, R8	1011	0011	0001	1000
			↑ ↑↑↑	↑

We can try to minimize the hamming distance (the number of bits that are different in two binary strings of the same length) between two registers.

Compiler Optimizations – Bus Energy

- Calculate the Hamming distance between each operand and the next instruction's operand:

R2=>R5 R5=>R7 R7=>R2

Instruction	Dest	Src 1	Src 2	Start	Ready	Energy	H(1)	H(2)	H(3)	H Tot
1 Load	R1	A		0	3	0	0			0
2 Load	R2	C		1	4	1	2			2
3 Load	R5	B		2	5	1	3			3
4 Load	R6	D		3	6	1	2			2
5 Sub	R3	R1	R2	4	5	3	2	1	1	4
6 Sub	R7	R5	R6	6	7	1	1	1	1	3
7 Mul	R4	R3	R3	7	11	3	2	2	2	6
8 Mul	R8	R7	R7	8	12	1	2	1	1	4
9 Add	R9	R4	R8	12	13	3	1	2	4	7
10 Store	R9	S		13	16	3	0	1	1	2
					<u>16</u>	<u>17</u>				<u>33</u>

Instruction	Dest	Src 1	Src 2	Start	Ready	Energy	H(1)	H(2)	H(3)	H Tot
1 Load	R1	A		0	3	0	0			0
2 Load	R5	C		1	4	1	1			1
3 Load	R7	B		2	5	1	1			1
4 Load	R6	D		3	6	1	1			1
5 Sub	R3	R1	R5	4	5	3	2	1	2	5
6 Sub	R2	R7	R6	6	7	1	1	1	2	5
7 Mul	R4	R3	R3	7	11	3	2	1	2	5
8 Mul	R8	R2	R2	8	12	1	2	1	1	4
9 Add	R9	R4	R8	12	13	3	1	2	2	5
10 Store	R8	S		13	16	3	0	1	1	2
					<u>16</u>	<u>17</u>				<u>29</u>

Compiler Optimizations – Memory Power

Consider three kinds of optimizations:

1. **Linear Loop Transformations:** Change the traversal order of the iteration space.
2. **Loop Tiling:** Split up loops that work on arrays that are too big to fit in the cache.
3. **Loop Unrolling:** Duplicate the body of a loop so as to reduce or eliminate the loop control structure.

Compiler Optimizations – Memory Power

- **Test program used: Matrix Multiply**

- **Platform caches simulated:**
 - 1, 2, 4, and 8 –way set associative
 - 1k, 2k, 4k, and 8k sizes

Compiler Optimizations – Memory Power

- Define three measures to look at correlation between cache hit-rate, memory energy savings, and total energy savings:

$$\text{Improvement}_M = \frac{\text{Miss rate of the original code}}{\text{Miss rate of the optimized code}}$$

$$\text{Improvement}_E = \frac{\text{Memory energy consumption of the original code}}{\text{Memory energy consumption of the optimized code}}$$

$$\text{Improvement}_T = \frac{\text{Total energy consumption of the original code}}{\text{Total energy consumption of the optimized code}}$$

13 April 2006

Paul Kohlbrenner, CS 803, Spring 2006

21

Compiler Optimizations – Memory Power

For tiling we get the following values:

	1K, 1-way	2K, 4-way	4K, 2-way	8K, 8-way
Improvement _M	6.21	63.31	20.63	19.5
Improvement _E	2.13	18.77	5.75	2.88
Improvement _T	1.96	9.27	3.08	1.47

Key Points:

1. Large improvements in cache miss rate do NOT translate into large reductions in energy. However, there is an improvement.
2. The improvement in memory energy is 2-15 times as large as the total energy improvement → Tiling increases core power.

13 April 2006

Paul Kohlbrenner, CS 803, Spring 2006

22

Compiler Optimizations – Memory Power

For loop unrolling we get the following values:

	1K, 1-way	2K, 4-way	4K, 2-way	8K, 8-way
Improvement t_M	1.65	2.82	1.67	1.52
Improvement t_E	2.07	3.53	2.07	1.83
Improvement t_T	2.03	3.37	1.97	1.68

Key Points:

1. Memory energy improvement is greater than the cache hit rate improvement.
2. Core power also improves but not by as much as the memory improvement.

Compiler Optimizations – Memory Power

For loop transforms we get the following values:

	1K, 1-way	2K, 4-way	4K, 2-way	8K, 8-way
Improvement t_M	4.02	10.23	3.3	1.18
Improvement t_E	3.42	8.51	2.74	0.99
Improvement t_T	3.17	6.84	2.32	0.94

Key Points:

1. Improvement in memory energy follows cache hit-rate improvement.
2. Core power increases with this optimization.

Compiler Optimizations – Memory Power

Memory optimization summary:

- Memory energy improvement follows cache hit-rate improvement.
- Overall energy improvement lags memory energy improvement by a factor of 1.05 to 1.80.

Compiler Optimizations – HW / SW

Two hardware optimizations considered:

1. Block Buffering: small buffer preserves the cache line for subsequent use. (Like a level 0 cache.)
2. Cache Subbanking: divide the data portion of the cache into parts and only access the part you need.

Compiler Optimizations – HW / SW

Results:

- The combination of the hardware optimizations and the software optimizations resulted in even greater energy savings.
- In the tiled case, using eight block buffers and four subbanks the overall memory system savings was 10%.

Compiler Optimizations – HW / SW

Another hardware optimization:

- Use different power modes for memory banks.
- We can use Loop Fission to try to take advantage of this.

Compiler Optimizations – HW / SW

- Loop Fission (AKA Loop Distribution) breaks up statements in nested loops into multiple nested loops.
- The hope is to improve instruction cache locality and instruction level parallelism.
- There will be an improvement in energy if we can use only one memory bank for a loop (the others will be in low power mode).

Compiler Optimizations – HW / SW

- The author's fissioning algorithm works by searching through the available partitioning of the inner "dominating loop" looking for the lowest energy solution.

Compiler Optimizations – HW / SW

Results:

- Dominating loops can have large (50%) improvements in energy use.
- Nests other than the inner “dominating nest” can have large energy increases due to the choice of layout strategy.
- However, for the benchmarks run, the non-dominating loop increase was only a 2.8% increase in energy.

Compiler Optimizations – Conclusions

Back to original questions:

1. Does fast = low power?
Usually there is an improvement in both.
2. What is the impact of performance-orientated optimizations on power?
Usually power decreases too, but to get maximum power savings you must optimize for power first.
3. What are the relative gains from hardware vs. software optimizations?
Primary savings come from software but an additional 10-15% savings can be achieved by combining hardware and software.

Next Paper

Loop Fusion - Introduction

Main points of this paper:

- Lowest energy demand occurs when the demand on the processor resources is balanced.
- Loop fusion can help balance demand and work with MCD and DVS processors to lower total energy.

Loop Fusion - Introduction

Multiple Clock Domain (MCD) and Dynamic Voltage Scaling (DVS) Processors:

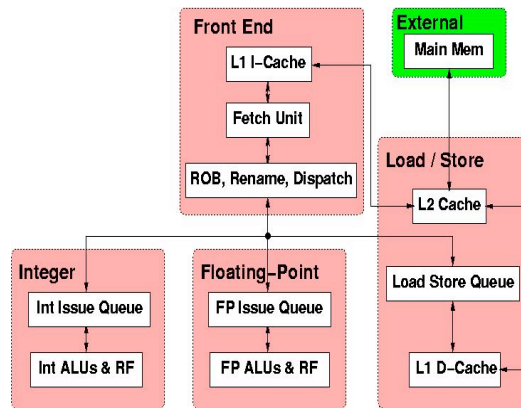


Diagram courtesy of paper's authors

13 April 2006

Paul Kohlbrener, CS 803, Spring 2006

35

Loop Fusion – Program Balance

- Assuming that time is held constant, a program's execution consumes minimal energy if it has constant balance.
- Constant balance means that the demand for the various processor resources is at the same level.
- Bounded balance means there is a critical component who's demand is constant and higher than other components.

13 April 2006

Paul Kohlbrener, CS 803, Spring 2006

36

Loop Fusion – Program Balance

- Essential point:

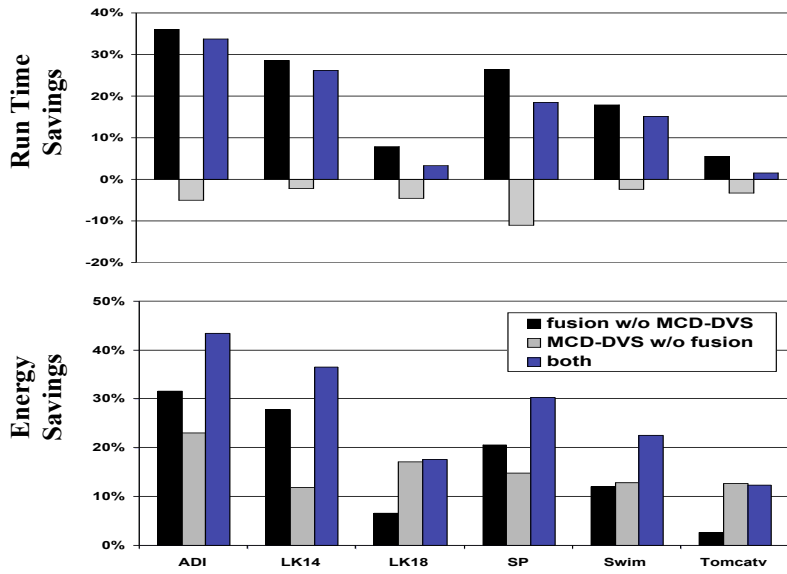
Energy is **not** linearly dependant on the operating voltage (and therefore frequency) of a component.

Loop Fusion – Experiments

Methodology:

- Use a source-to-source fusing compiler
- Aggressive loop fusion
 - Uses whole program inter-procedural analysis.
 - Fuses loops as much as dependence will allow.
- Analysis ignores cache problems.

Loop Fusion – Experiments



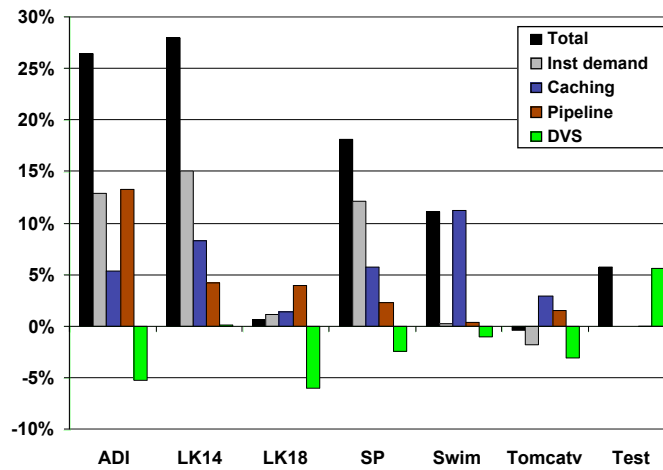
13 April 2006

Paul Kohlbrenner, CS 803, Spring 2006

39

Loop Fusion – Experiments

Breakdown of Fusion-induced Energy Savings on MCD/DVS processors



13 April 2006

Paul Kohlbrenner, CS 803, Spring 2006

40

Loop Fusion – Conclusions

- Loop fusion can smooth program demand which can save energy if execution time is held constant.
- Loop fusion reduces the ability of an MCD processor to save energy if the program is run as fast as possible.
- Loop fusion can work synergistically with MCD if program is run at a constant rate.

Paper Reconciliation

- So, how can both loop fusion and loop fission save energy?

- Start dancing....

Paper Reconciliation

Answer:

- Through the magic of choosing your own benchmarks.
 - In Loop fission case very small caches (8k max) with arrays that won't fit.
 - In Loop fusion case much larger caches (64k L1, 1M L2) and benchmarks that fit.

Questions