

A source-level estimation and optimization methodology for execution time and energy consumption of embedded software

Daniele P. Scarpazza
scarpaz@scarpaz.com
Dipartimento di Elettronica e Informazione
Politecnico di Milano

I. The need: Why this research was needed

1.1 Requirements:
designers need fast, dynamic, fine-detail, source-level techniques to estimate the energy consumed by their software;

1.2 Focus:
I focus on the core of single-issue CPUs (no memory hierarchy, no VLIW, ...)

1.3 State of the Art:
current techniques do not satisfy the above requirements;

1. fast

2. dynamic

3. source-level

4. fine-detail

2. dynamic

- modern applications are becoming more and more dynamic in nature;
- the behavior of multimedia en-/decoders depends more and more on the contents of the streams they process;

- workload variability is high and increasing;
- the gap between typical and worst case is very large;
- static techniques are worst-case techniques, and lead to expensive, oversized systems which are underutilized most of the time;

3. source-level

- many energy estimation flows operate at the assembly level, but designers do not code in assembly any more;
- designers use high-level languages instead, estimation flows should provide information at the same abstraction level;
- compilation is a (more and more) complex process; lot of skill and experience required to relate instruction-level estimates to the source-level causes;
- source-level optimizing transformations have been showed to lead to the highest gains; their steering need source-level analysis;

4. fine detail

- most of the time and energy is spent in small computational kernels;
- "small" is much smaller than a program and a function, potentially smaller than inner loops;
- many estimation techniques (even source-level ones) cannot "look inside functions"

1.3 Current techniques do not satisfy these requirements

- Static Timing Analysis (STA) techniques cannot deal with dynamism; (Puschner89, Chen01)
- Instruction-Set Simulation (ISS) is slow and at a low level; (Brooksoo, Sinaori, Qino)
- ISS + gprof provide estimates only at a function level; (Simunovic)
- Atomium/PowerEscape is source-level, but only for memory aspects (not our focus); (Bormans99, Arnouts)
- SoftExplorer is a static technique; (Senoz)
- Compilation-based approaches do not provide link to source level; (Lajologu)
- SIT is source level (good!) but still unable to resolve chosen clusters; (Ravasio)
- Black-box techniques do not provide any link with source code; (Mutreja04)

2. Theory: how this technique works

2.1 Divide and conquer:

$$C_i = n_i \cdot c_i$$

cost of executing the i-th node in the AST execution count single-execution cost

2.1. Divide and conquer

2.2. Determining single-execution costs

Attribute	Computation	Defined for which AST nodes
c total cost	synthesized	expressions and statements
ci inherent cost	synthesized	expressions and statements
cc conversion cost	synthesized	expressions and statements
cf flow control cost	inherited	expressions and statements
k constancy	synthesized	expressions
e constant value	synthesized	expressions
v valueness	inherited	expressions
r restricted result type	inherited	expressions
b register-boundedness	synthesized	expressions
f translation flavor	inherited	expressions and statements

2.3. Determining execution counts

optimal strategy to select probe insertion points

- only one probe per each generalized basic block (g.b.b.);
- a g.b.b. is a maximal set of nodes, all executed the same number of times (possibly larger than basic blocks); example:

transparent, probe-inserting source-to-source transformations:

```

expressions: e      ( __profile(137), e )
statements: s;      ( __profile(137); s; )
functions: int f(args)
                  ( __profile(151);
                    { ... }
                  )
    
```

3. Results: The technique is accurate and fast

3.1 ANSI-C compliant flow implementation available

3.2. New experiments – Setup:

Simulator: SimIt-ARM v2.0.3 with cache latency = 0 [Qino]

Platform: SA-1100 @ 206 MHz, 1.5 Vdd

Parameters: avg. currents for each instruction, from JouleTrack [Sinha01]

Compiler: gcc v2.95 -O2/-O3

Benchmarks: from MiBench [Guthaus01]

3.1 ANSI C-compliant tool flow available

3.3. Accuracy results

	SimIt		eStools		error	
	E (mJ)	T (ms)	E (mJ)	T (ms)	E	T
adpcm-s	46,1	166,3	41,9	156,4	-9,1%	-6,0%
adpcm-l	910,2	3289,9	722,1	2710,5	-20,7%	-17,6%
bitcount-s	65,7	242,8	55,0	204,0	-16,3%	-16,0%
bitcount-l	981,9	3628,6	977,1	3649,2	-0,5%	+0,6%
blowfish	1067,0	3742,7	748,3	3371,0	-29,9%	-9,9%
CRC32	38,3	132,2	35,4	129,6	-7,5%	-2,0%
FFT-s	207,9	764,6	207,1	770,3	-0,4%	+0,7%
FFT-l	3213,2	11851,5	3264,8	12142,5	+1,6%	+2,5%
IFFT-s	205,1	755,1	207,3	771,0	+1,1%	+2,1%
IFFT-l	3181,8	11744,7	3266,2	12147,8	+2,7%	+3,4%
jpeg	87,9	309,9	91,2	328,5	+3,8%	+6,0%
rijndael	63,8	221,3	71,4	257,3	+12,0%	+16,3%
sha-s	22,1	78,9	21,9	78,6	-0,9%	-0,4%
sha-l	229,4	820,0	224,7	818,3	-2,1%	-0,2%

Quality of result:

- $\rho(E, \hat{E}) = 0,9960$, $|\overline{E} - \hat{E}| = 7,49\%$
- $\rho(T, \hat{T}) = 0,9987$, $|\overline{T} - \hat{T}| = 5,65\%$

4. Uses and developments

4.1 Automated source optimization

Results:

energy reduction: -5,1 - -22,0%

execution time reduction: -7,8 - -22,3%

4.2 Support for VWR

- Very wide register (VWR) architectures achieve extreme low power via:
 - a wide data-path (e.g. 256 bit) and very wide registers (e.g. 2048 bit) with SIMD instructions;
 - a software controlled scratchpad in place of a L1 cache;
 - a loop buffer (32 instructions);

4.3 Support for VLIW

- trace-based: model exactly the per-trace compilation results of VLIW compilers;
- incremental rebuild: rebuild only the intermediate products actually needed by changes made in the source code, architecture, input data;
- keep the current efficiency; obtain trace-based profiles by current node-based profiles;

4.4 Extension to C++

- The extension to C++ is feasible with acceptable effort;
- Tasks required:
 - lexer adaptation (x8 new keywords, negligible effort);
 - parser syntax adaptation (213 >> 560 syntax rules);
 - new type system and scoping rules (significant effort);
 - parser needs some semantic-level disambiguation techniques;
 - overloading / templates / late binding (current instrumentation technique is sufficient to determine which function / method has been actually called);
 - extension of theoretical abstract translation model (significant effort);
- Required effort: 1 "me-year"

Selected Publications

- Book chapters:
 - "Estimation of the execution time and energy consumption at source code", in F. Cathoor, J. I. Gomez, S. Himpe, Z. Ma, P. Marchal, D. P. Scarpazza, C. Wong, P. Yang, "Systematic methodology for real-time cost-effective mapping of dynamic concurrent task-based systems on heterogeneous platforms", Springer Verlag [accepted];
- Journal papers:
 - with Carlo Brandolese, "A source-level software analysis methodology able to resolve clusters of operations and finer details", Journal on Low-power Electronics (JOLPE) [accepted];
 - with Carlo Brandolese, "Energy estimation for Embedded Software", IEEE Transactions on Computers; [accepted];
- Conference papers:
 - with C. Brandolese, "A fast, dynamic, source-level and fine-detail technique to estimate the energy consumed by embedded software on single-issue processor cores", CODES+ISSS'06, Seoul, Korea [submitted];
 - with P. Raghavan, D. Novo, C. Brandolese, F. Cathoor, D. Verkest, "Software Simultaneous Multi-Threading, a technique to exploit Task-level Parallelism to improve Instruction and Data-level Parallelism", PATMOS'06, Montpellier, France [submitted];