Inferring Energy Bounds via Static Program Analysis and Evolutionary Modeling of Basic Blocks

Umer Liqat[1,3] Zorana Banković [1]

Pedro López-García[1,2] Manuel Hermenegildo[1,3]

[1]IMDEA Software Institute
[2]Spanish Research Council (CSIC)
[3]Technical University of Madrid (UPM)

LOPSTR, Namur, Belgium, Oct 11, 2017

Motivation

- Energy consumption is a significant issue in systems ranging from:
  - small *Internet of Things (IoT)* devices, sensors, smart watches, smart phones and portable/implantable medical devices, to
  - large data centers and high-performance computing systems.
- A need for estimating the *energy consumed by program executions*.
  - Often dependent on run-time data sizes (string length, signal samples, recursions, etc.).
- Different types of energy estimations can be performed, depending on the application: probabilistic, average, safe bounds, ...
- For verification $\rightarrow$ *safe upper and lower bounds* are required.
- Given an energy budget E_b and safe upper- and lower-bound estimations, E_u and E_l respectively:
  1. $E_u \leq E_b \implies$ the given program can be safely executed within the existing energy budget.
  2. $E_l \leq E_b \leq E_u \implies$ it might be possible to execute the program, but we cannot claim it for certain.
  3. $E_b < E_l \implies$ it is not possible to execute the program (the system will run out of batteries before program execution is completed).

Motivation

- Energy consumption is a significant issue in systems ranging from:
  - small *Internet of Things (IoT)* devices, sensors, smart watches, smart phones and portable/implantable medical devices, to
  - large data centers and high-performance computing systems.
- A need for estimating the *energy consumed by program executions*.
  - Often dependent on run-time data sizes (string length, signal samples, recursions, etc.).
- Different types of energy estimations can be performed, depending on the application: probabilistic, average, safe bounds, ...
- For verification $\rightarrow$ *safe upper and lower bounds* are required.
- Given an energy budget E_b and safe upper- and lower-bound estimations, E_u and E_l respectively:
  1. $E_u \leq E_b \implies$ the given program can be safely executed within the existing energy budget.
  2. $E_l \leq E_b \leq E_u \implies$ it might be possible to execute the program, but we cannot claim it for certain.
  3. $E_b < E_l \implies$ it is not possible to execute the program (the system will run out of batteries before program execution is completed).

Motivation

- Energy consumption is a significant issue in systems ranging from:
  - small *Internet of Things (IoT)* devices, sensors, smart watches, smart phones and portable/implantable medical devices, to
  - large data centers and high-performance computing systems.
- A need for estimating the *energy consumed by program executions*.
  - Often dependent on run-time data sizes (string length, signal samples, recursions, etc.).
- Different types of energy estimations can be performed, depending on the application: probabilistic, average, safe bounds, ...
- For verification $\rightarrow$ *safe upper and lower bounds* are required.
- Given an energy budget E_b and safe upper- and lower-bound estimations, E_u and E_l respectively:
  1. $E_u \leq E_b \implies$ the given program can be safely executed within the existing energy budget.
  2. $E_l \leq E_b \leq E_u \implies$ it might be possible to execute the program, but we cannot claim it for certain.
  3. $E_b < E_l \implies$ it is not possible to execute the program (the system will run out of batteries before program execution is completed).

Using Upper and Lower Bounds for Energy Verification

- We face an interesting safety/accuracy trade-off.
- Challenge: finding a practical compromise.

Goal

Estimate tight *upper and lower bounds* on the energy consumption of a program *as functions on its input data sizes*

$\rightarrow$ that are practical for energy verification (and optimization).

Approach

A novel combination of static and dynamic (modeling) techniques.

- We face an interesting safety/accuracy trade-off.
- Challenge: finding a practical compromise.

Goal

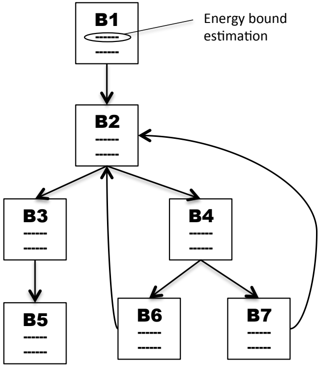
Estimate tight *upper and lower bounds* on the energy consumption of a program *as functions on its input data sizes*

$\rightarrow$ that are practical for energy verification (and optimization).

Approach

A novel combination of static and dynamic (modeling) techniques.

Modeling at the Instruction Level (Choice 1)

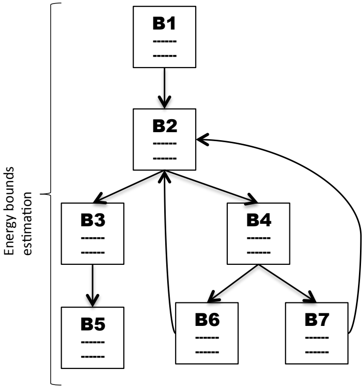


- Each instruction is profiled (using, e.g., an Evolutionary Algorithm – EA) to derive upper- and lower-bound energy estimates.
- These are combined using static analysis.

+ Very compositional.
+ Can infer functions of input data sizes.

– Bounds obtained are *very conservative*.
– Dependence among instructions is not modeled (or very complex).

Modeling the Whole Program (Choice 2)

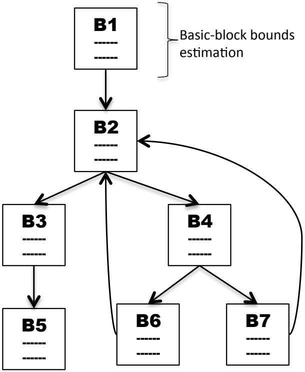


- The whole program is profiled using the EA to estimate upper/lower bounds (no static analysis performed).

+ Instruction dependence is captured.
+ Bounds can be very precise (if no data-dependent branching).

– The EA infers just one fixed cost for a given fixed input.
– The EA becomes imprecise and impractical due to data-dependent branching.

Our Proposal: Modeling at Basic Block Level

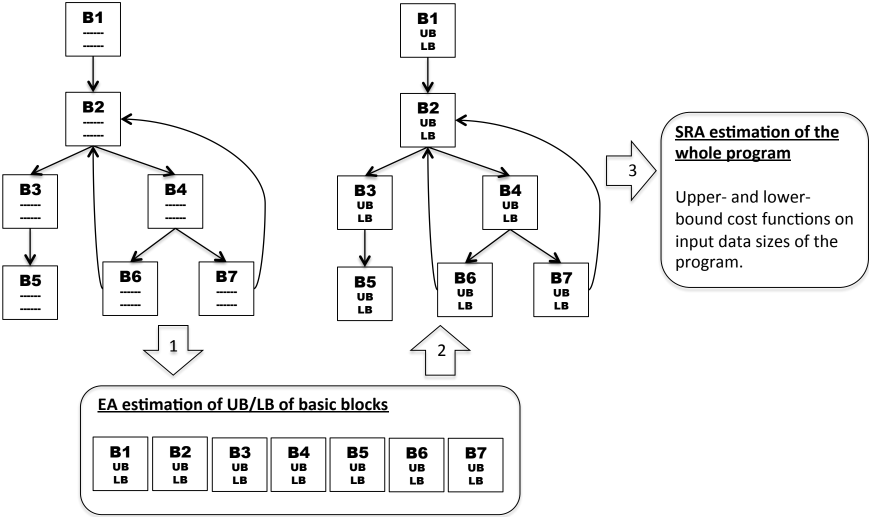


- Each basic block is profiled using the EA and upper/lower bounds estimated for each block.
- Bounds over basic blocks are composed (by static analysis) to infer the bounds over the whole program.

+ Inter-instruction dependence is captured within the blocks: more precise bounds.
+ The EA is precise and practical since no data-dependent branching within a block.
+ Infers functions of input data sizes.

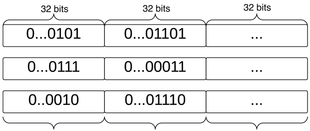
− Inter-block dependence may be over- or under-estimated.

Overview of our Approach



A custom EA is used to estimate the maximum/minimum energy consumption of each basic block.

- An individual is constructed from a set of input arguments to a basic block.
- The initial population includes randomly created individuals plus any known corner cases that may maximize/minimize the energy of basic blocks.
- Example: mutation operation



Dividing the Program into Basic Blocks

Listing 1.1: Factorial function.

```
int fact(int N)
{
  if (N <= 0)
    return 1;

  return N*fact(N - 1);
}
```

Listing 1.2: Basic blocks.

```
<fact>:
     01: entsp 0x2
     02: stw   r0, sp[0x1]
B1   03: ldw   r1, sp[0x1]
     04: ldc   r0, 0x0
     05: lss   r0, r0, r1
     06: bf    r0, <08>

     07: bu    <010>
     10: ldw   r0, sp[0x1]
     11: sub   r0, r0, 0x1
B2   12: bl    <fact>
     13: ldw   r1, sp[0x1]
     14: mul   r0, r1, r0
     15: retsp 0x2

B3   08: mkmsk r0, 0x1
     09: retsp 0x2
```

before call

after call

Listing 1.3: Modified basic blocks.

```
<fact>:
 01: entsp 0x2
 02: stw   r0, sp[0x1]
 03: ldw   r1, sp[0x1]            B1
 04: ldc   r0, 0x0
 05: lss   r0, r0, r1
 06: bf    r0, <08_NEW>
 08_NEW:

 07: bu    <010>
 10: ldw   r0, sp[0x1]            B2_1
 11: sub   r0, r0, 0x1

 12: bl <fact>

 13: ldw   r1, sp[0x1]
 14: mul   r0, r1, r0             B2_2
 15: retsp 0x2

 08: mkmsk r0, 0x1                B3
 09: retsp 0x2
```

Dividing the Program into Basic Blocks (Contd.)

- A basic block with k function call instructions is divided into $k+1$ basic blocks without the function call instructions.
- A set of special instructions (e.g., entsp, retsp, bl, etc.) are measured separately.
- The memory accesses in each block are transformed into accesses to a fixed address in the local memory of the harness function.

Basic Block Input Arguments to Individuals

- The set $gen(B)$ characterizes the set of variables

$$gen(b) = \bigcup_{k=1}^{n} \{v \mid v \in ref(k) \land \forall (j < k).v \notin def(j)\}$$

  where $ref(n)$ and $def(n)$ denote the variables referred to and defined/updated at node n in block b respectively.

- *Example:*

  $gen(B1)$={r0}, $gen(B2_1)$={sp[0x1]},
  $gen(B2_2)$={sp[0x1],r0}, $gen(B3)$={}.

Horn Clause Representation for Static Analysis

An ISA (factorial) program (left) and its Horn clause representation (right)

```
<fact>:
0x01: entsp 0x2
0x02: stw   r0, sp[0x1]
0x03: ldw   r1, sp[0x1]
0x04: ldc   r0, 0x0
0x05: lss   r0, r0, r1
0x06: bf    r0, <0x08>



0x07: bu    <0x10>
0x0a: ldw   r0, sp[0x1]
0x0b: sub   r0, r0, 0x1
0x0c: bl    <fact>
0x0d: ldw   r1, sp[0x1]
0x0e: mul   r0, r1, r0
0x0f: retsp 0x2


0x08: mkmsk r0, 0x1
0x09: retsp 0x2
```

```
fact(R0,R0_3):-
   entsp(0x2),
   stw(R0,Sp0x1),
   ldw(R1,Sp0x1),
   ldc(R0_1,0x0),
   lss(R0_2,R0_1,R1),
   fact_aux(R0_2,Sp0x1,R0_3,R1_1).


fact_aux(1,Sp0x1,R0_4,R1):-
   bu(0x10),
   ldw(R0_1,Sp0x1),
   sub(R0_2,R0_1,0x1),
   fact(R0_2,R0_3),
   ldw(R1,Sp0x1),
   mul(R0_4,R1,R0_3),
   retsp(0x2).

fact_aux(0,Sp0x1,R0,R1):-
   mkmsk(R0,0x1),
   retsp(0x2).
```

Energy Consumption of the Whole Program

- Static analysis combines the energy bounds estimations for each block in order to infer the energy bounds of the whole program.
- Let $B1_e^A$, $B2_e^A$ and $B3_e^A$ represent the energy bounds for the blocks $B1$, $B2$ and $B3$ respectively.
- The equation that characterizes the energy bounds of the whole program is:

$$fact_e^A(R0) \quad = \quad B1_e^A + fact\_aux_e^A(0 \leq R0, R0)$$

$$fact\_aux_e^A(B, R0) = \begin{cases} B2_e^A + fact_e^A(R0 - 1) & \text{if } B \text{ is } \texttt{true} \\ B3_e^A & \text{if } B \text{ is } \texttt{false} \end{cases}$$

  where A is the kind of approximation (upper/lower bound).

- Closed-form solution for upper- and lower-bounds:

$$fact_e^{ub}(R0) = 5.1R0 + 4.2 \text{ nJ}$$
$$fact_e^{lb}(R0) = 4.1R0 + 3.8 \text{ nJ}$$

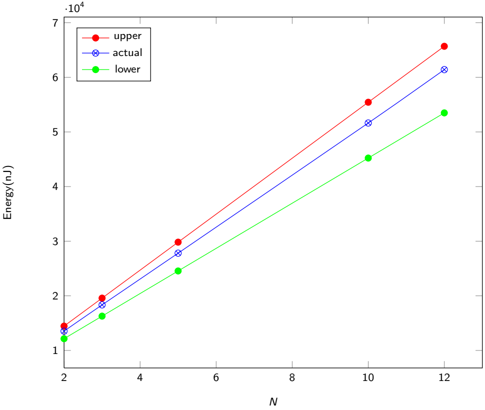
Experimental Evaluation (XMOS XS1 architecture)

Program	DDBr	Upper/Lower Bounds (nJ)$\times 10^3$	vs. HW
$fact(N)$	n	$ub = 5.1\ N + 4.2$ $lb = 4.1\ N + 3.8$	$+7\%$ -11.7%
$fibonacci(N)$	n	$ub = 5.2\ lucas(N) + 6\ fib(N) - 6.6$ $lb = 4.5\ lucas(N) + 5\ fib(N) - 4.2$	$+8.71\%$ -4.69%
$reverse(A)$	n	$ub = 3.7\ N + 13.3$ (N = length of array A) $lb = 3\ N + 12.5$	$+8\%$ -8.8%
$findMax(A)$	y	$ub = 5\ N + 6.9$ (N = length of array A) $lb = 3.3\ N + 5.6$	$+8.7\%$ -9.1%
$selectionSort(A)$	y	$ub = 30\ N^2 + 41.4\ N + 10$ (N = length of array A) $lb = 16.8\ N^2 + 28.5\ N + 8$	$+8.7\%$ -9.1%
$fir(N)$	y	$ub = 6\ N + 26.4$ $lb = 4.8\ N + 22.9$	$+8.9\%$ -9.7%
$biquad(N)$	y	$ub = 29.6\ N + 10$ $lb = 23.5\ N + 9$	$+9.8\%$ -11.9%

- EA times vary depending upon the initialization parameters.
  - On average within 150-200 min.
- Static analysis times are relatively small $\approx 4sec$.

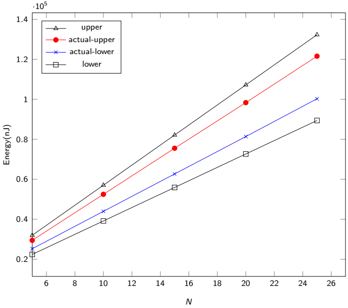
Experimental Results (Benchmark with no Data-Dependent Branching)

factorial(x): 7% over- and 11% under-approximation for random runs with different inputs.

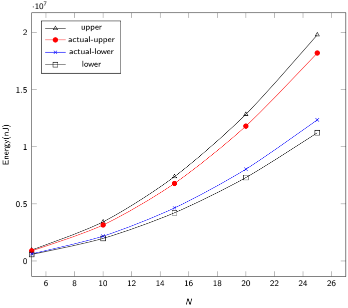


Experimental Results (Benchmark with Data-Dependent Branching)

Over- and under-approximation from actual upper- and lower-bounds (ascending vs. descending sorted array).



(a) *findMax*.



(b) *selectionsort*.

Static Analysis vs. Energy Modelling (*findMax*)

N	Cost App	Energy(nJ)$\times 10^3$ Est	Prof	Obs	D %	PrD %
Actual- worst and best case array data						
5	lb	22.3	22.3	25.2	-12.2	-12.2
	ub	31.9	31.9	29.4	8.1	8.1
15	lb	55.9	55.9	62.6	-11.3	-11.3
	ub	82.1	82.1	75.5	8.3	8.3
25	lb	89.4	89.4	100.2	-11.4	-11.4
	ub	132.2	132.2	121.5	8.4	8.4

- The static analysis part is accurate (exact).
- All the inaccuracy comes from the EA.

Summary

- We have proposed an approach for inferring parametric upper and lower bounds on the energy consumption of a program
  - a combination of dynamic (evolutionary algorithms) and static techniques.
- We have used an EA to estimate the energy bounds of basic blocks:
  - Instructions dependence is captured within blocks.
  - The blocks have no branches, which make the EA more practical.
- We have used static analysis to compose the energy bounds of basic blocks in order to infer upper/lower bounds of the whole program.
  - Inter-block dependence is over- or under-approximated.
- Experimental results: the bounds inferred are safe and quite accurate.
- A practical technique for its application to energy verification (and optimization).

  → A practical compromise for safety/accuracy trade-off.

Summary

- We have proposed an approach for inferring parametric upper and lower bounds on the energy consumption of a program
  - a combination of dynamic (evolutionary algorithms) and static techniques.
- We have used an EA to estimate the energy bounds of basic blocks:
  - Instructions dependence is captured within blocks.
  - The blocks have no branches, which make the EA more practical.
- We have used static analysis to compose the energy bounds of basic blocks in order to infer upper/lower bounds of the whole program.
  - Inter-block dependence is over- or under-approximated.
- Experimental results: the bounds inferred are safe and quite accurate.
- A practical technique for its application to energy verification (and optimization).

  → A practical compromise for safety/accuracy trade-off.

Summary

- We have proposed an approach for inferring parametric upper and lower bounds on the energy consumption of a program
  - a combination of dynamic (evolutionary algorithms) and static techniques.
- We have used an EA to estimate the energy bounds of basic blocks:
  - Instructions dependence is captured within blocks.
  - The blocks have no branches, which make the EA more practical.
- We have used static analysis to compose the energy bounds of basic blocks in order to infer upper/lower bounds of the whole program.
  - Inter-block dependence is over- or under-approximated.
- Experimental results: the bounds inferred are safe and quite accurate.
- A practical technique for its application to energy verification (and optimization).

  $\rightarrow$ A practical compromise for safety/accuracy trade-off.

Summary

- We have proposed an approach for inferring parametric upper and lower bounds on the energy consumption of a program
  - a combination of dynamic (evolutionary algorithms) and static techniques.
- We have used an EA to estimate the energy bounds of basic blocks:
  - Instructions dependence is captured within blocks.
  - The blocks have no branches, which make the EA more practical.
- We have used static analysis to compose the energy bounds of basic blocks in order to infer upper/lower bounds of the whole program.
  - Inter-block dependence is over- or under-approximated.
- Experimental results: the bounds inferred are safe and quite accurate.
- A practical technique for its application to energy verification (and optimization).

  $\rightarrow$ A practical compromise for safety/accuracy trade-off.