Methodology of extraction of reliable energy data on a basic block level

Dimitrios Stamatios Bouras - el17072@mail.ntua.gr

Laboratory: Microprocessors and Digital systems Lab

University: National Technical University of Athens

Department: Electrical and Computer Engineering

Supervising Professor : Dimitrios Soudris

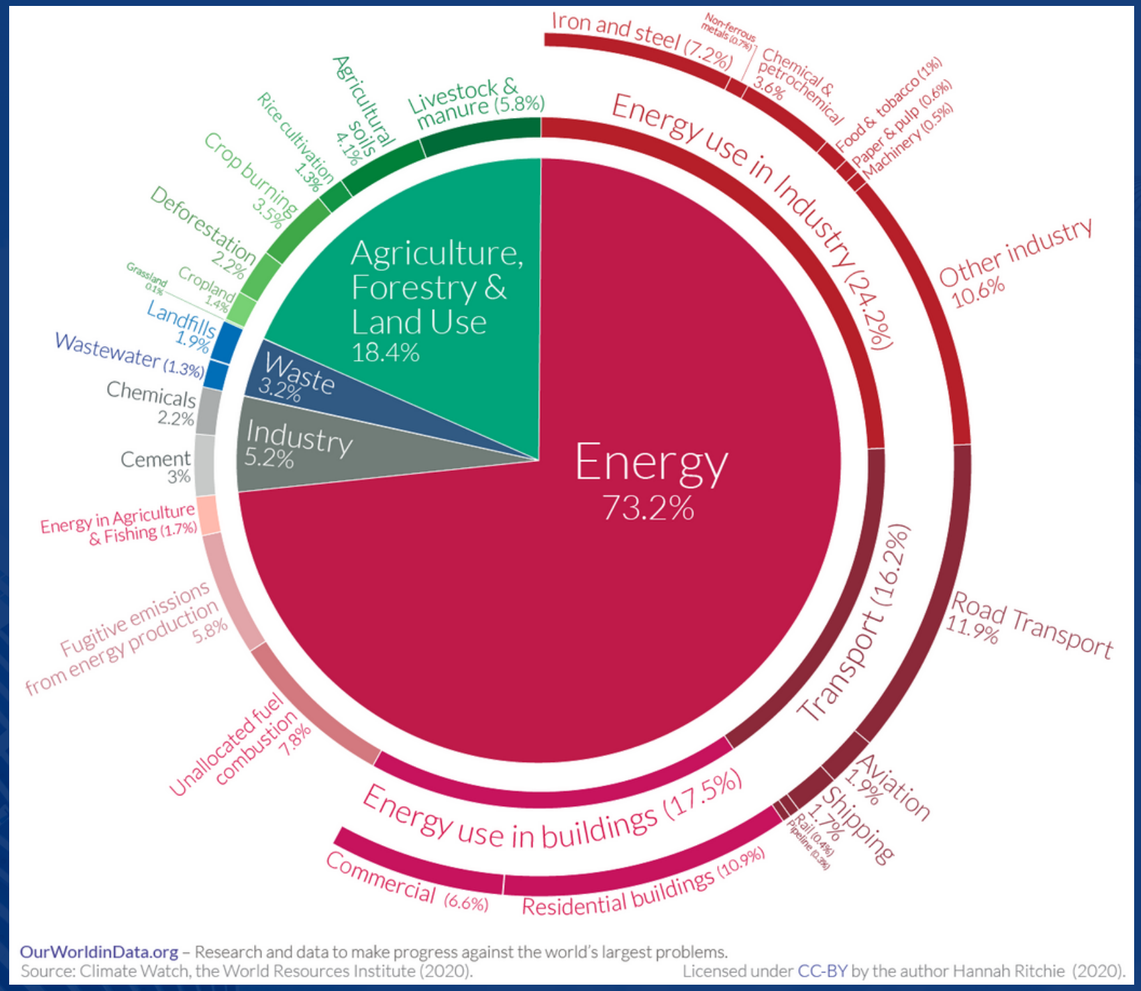
Supervising Phd candidate: Christos Lamprakos

Github repo: https://github.com/jimbou/dataset_tool.git

Energy efficient Software

- Reduced energy cost
- Reduced Heat dissipation
- Reduce energy footprint - GHG emmision
- Longer battery life

Greenhouse gas emissions per sector



ICT ~ 4%

Energy efficient Hardware

- Larger improvements with less change
- Improvements easily observed

but

- Energy cost per claculation decreasing but calculations' amount increasing
- Unable to satisfy increasing energy needs

To achieve → Energy efficient software

First → Software energy measuring tool

Why basic blocks?

- Higher Granularity
- Higher Accuracy
- More flexibility for developers
- Can't get any lower practically with energy reduction in mind

Basic block : a straight-line code sequence with one branch in and one branch out and no branches out.

Infastructure

Computer specs :

Specifications of the computer utilized for this thesis	
Operating System	Linux Ubuntu 20.04.1
Core version	5.15.0-58 -generic
Architecture	$x86_64$
Processor	Intel Core i7 – 6700
Processor frequency	3.4 GHz
Bridge	Xeon E3-1200 v5/E3-1500 v5/6th Gen
Bus	100 Series/C230 3.0 xHCI Controller
Communication	100 Series/C230 3.0 MEI Controller 1
RAM	32GiB System memory

Energy Measurements Intel RAPL

- set of registers providing energy and power consumption information
- uses a software power model based on performance counters
- Refresh frequency 1KHz
- Available measurements for core , uncore, RAM and GPU
- High accuracy

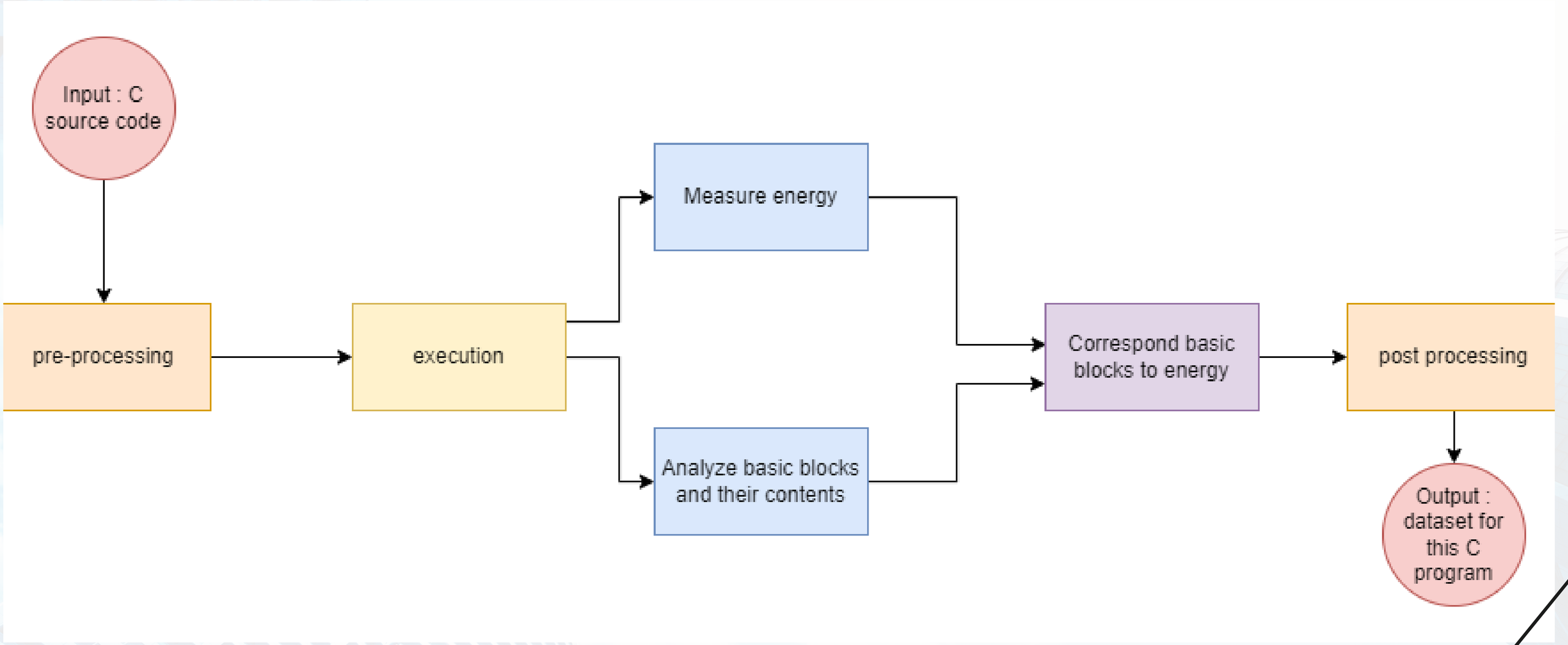
LLVM

- LLVM is a set of compiler and toolchain technologies
- Designed around a language-independent intermediate representation (IR)
- language-independent instruction set and type system
- LLVM passes perfom analysis, transformations ansd optimizations on the code

3 different approaches :

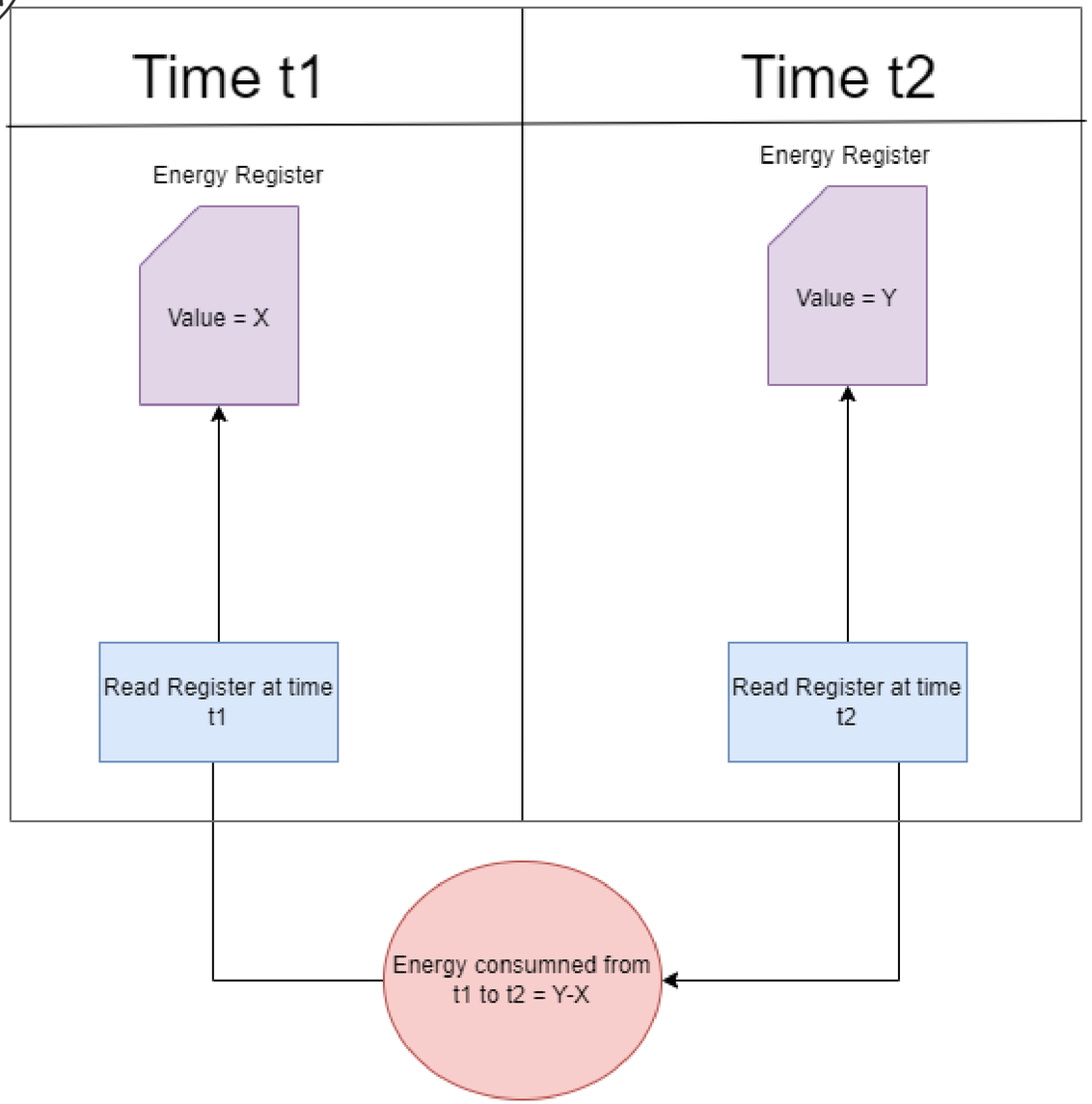
- Individual basic block measuring with RAPL calls
- Binary Lifting to LLVM IR
- Post analysis of execution trace

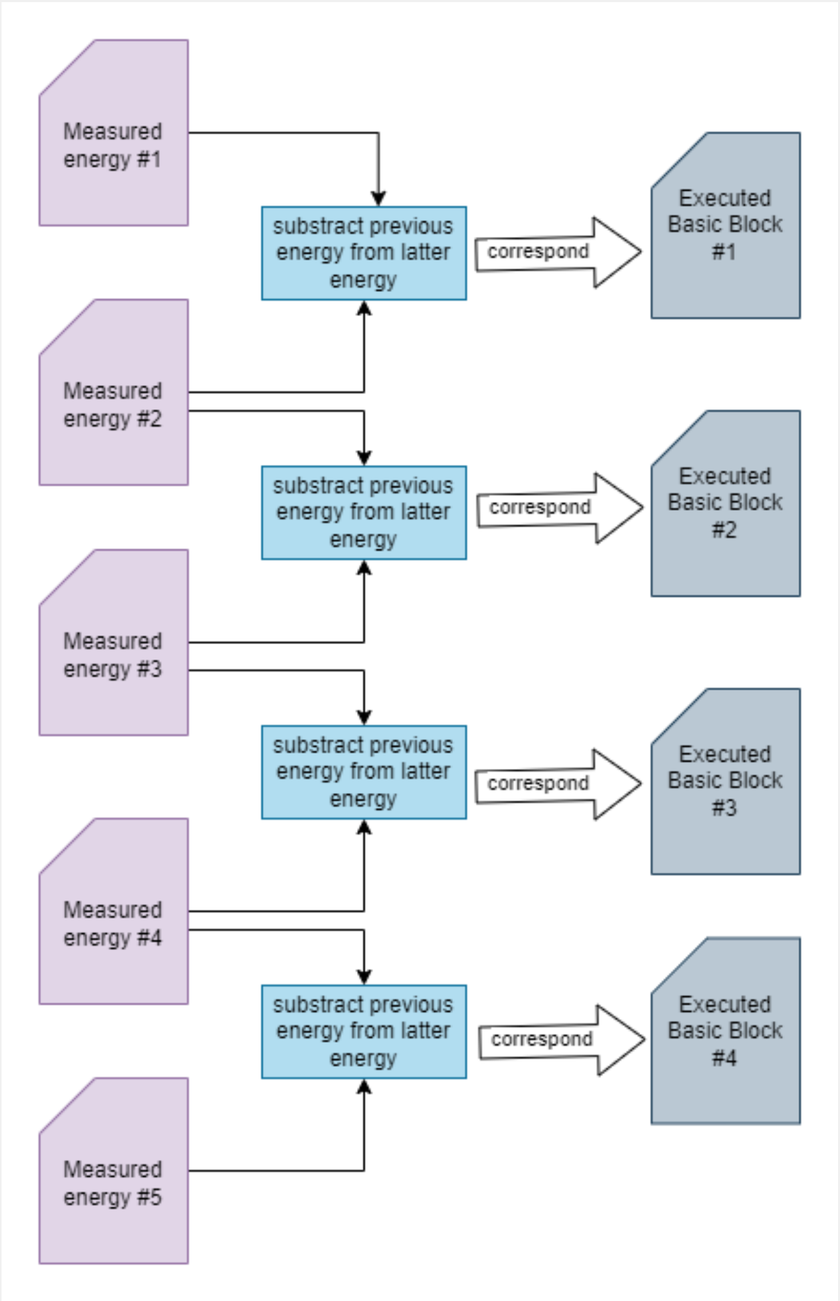
Pipeline


Input : C source code
pre-processing
execution
Measure energy
Analyze basic blocks and their contents
Correspond basic blocks to energy
post processing
Output : dataset for this C program

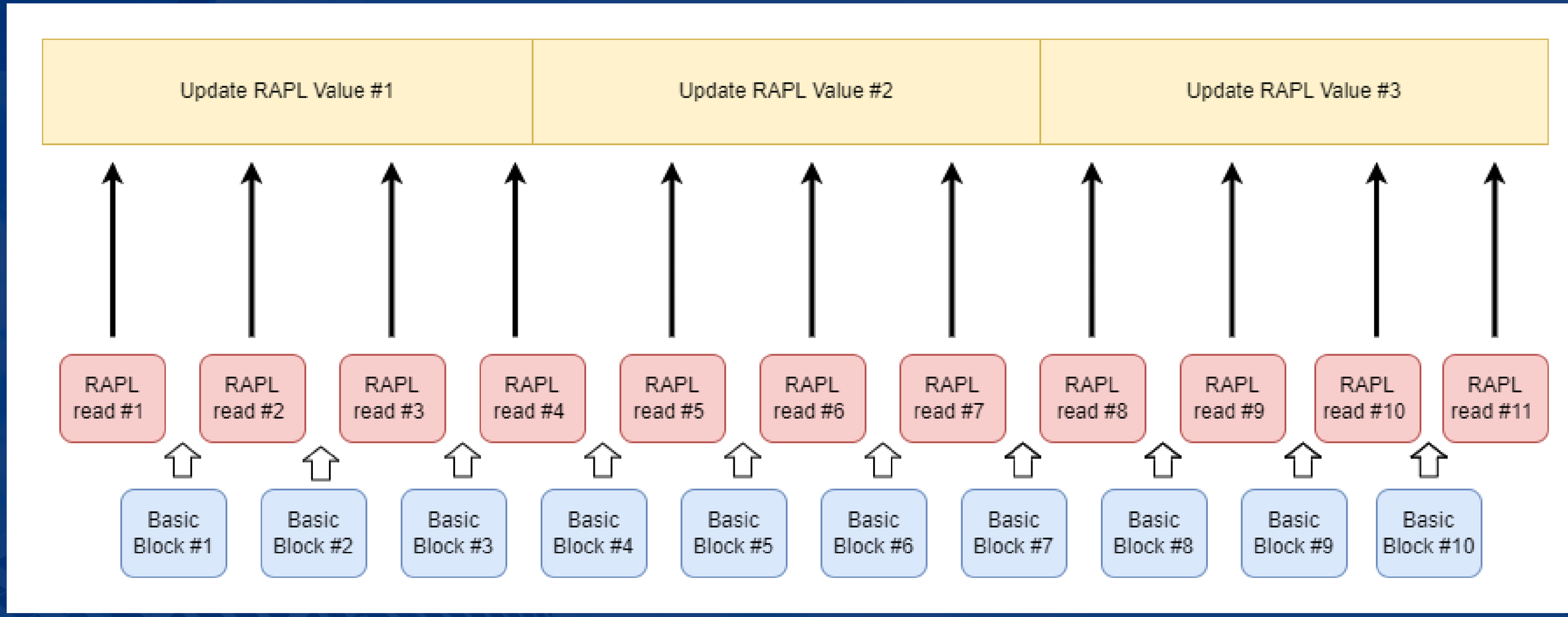
First approach : individual basic block measurement using RAPL calls

- Create an unique ID for each BB
- Add RAPL call function before every BB
- Measure energies through subtraction
- Correspond energy to BB using ID


Time t1
Time t2
Energy Register
Value = X
Energy Register
Value = Y
Read Register at time t1
Read Register at time t2
Energy consumned from t1 to t2 = Y-X


Measured energy #1
Measured energy #2
Measured energy #3
Measured energy #4
Measured energy #5
substract previous energy from latter energy
substract previous energy from latter energy
substract previous energy from latter energy
substract previous energy from latter energy
correspond
correspond
correspond
correspond
Executed Basic Block #1
Executed Basic Block #2
Executed Basic Block #3
Executed Basic Block #4

Problem 1: Ineffectiveness of RAPL read granularity



Solution 1: Split energy based on sum of execution time of assembly commands of each BB

Wiz= execution time of comand z of BB i

ti = execution time of BBi (in CPU cycles)

X = energy of multiple BBs

$$t_i = \sum_{z=1}^{M} w_{iz}$$

[1]

$$\sum_{i=1}^{N} t_i = T$$

[2]

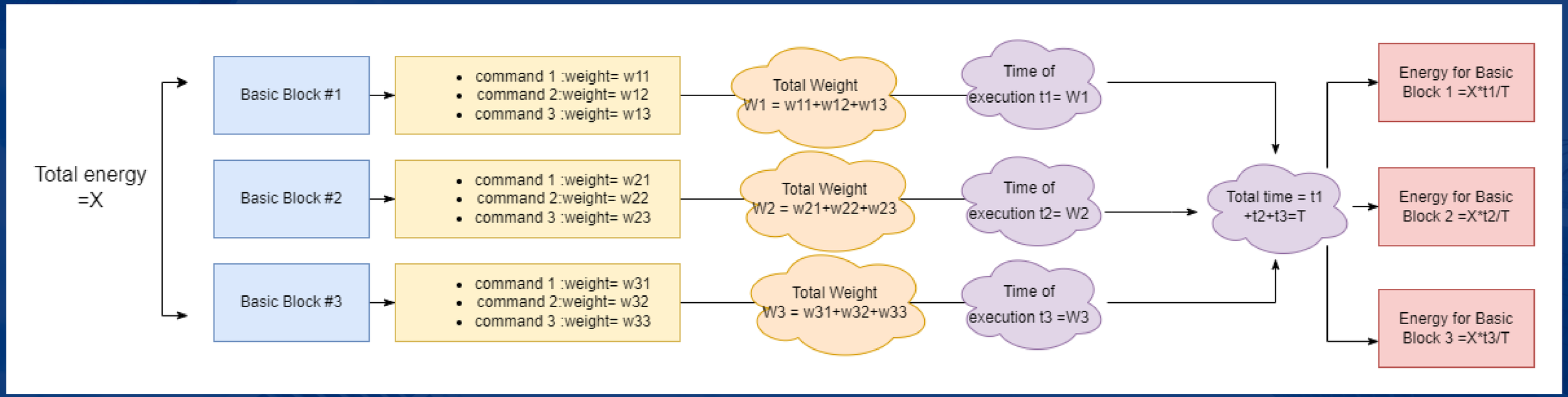
Final energy of BB i=

$$\frac{X * t_i}{T}$$

.[1] Tiwari, S. Malik, A. Wolfe και M.T. C. Lee. Instruction level power analysis and optimization of software. Proceedings of 9th International Conference on VLSI Design, σελίδες 326–328, 1996

[2] Lev Mukhanov, Dimitrios S. Nikolopoulos και Bronis R. De Supinski. ALEA: Fine-Grain Energy Profiling with Basic Block Sampling. 2015 International Conference on Parallel Architecture and Compilation (PACT), σελίδες 87–98, 2015

Solution 1: Split energy based on sum of execution time of assembly commands of each BB



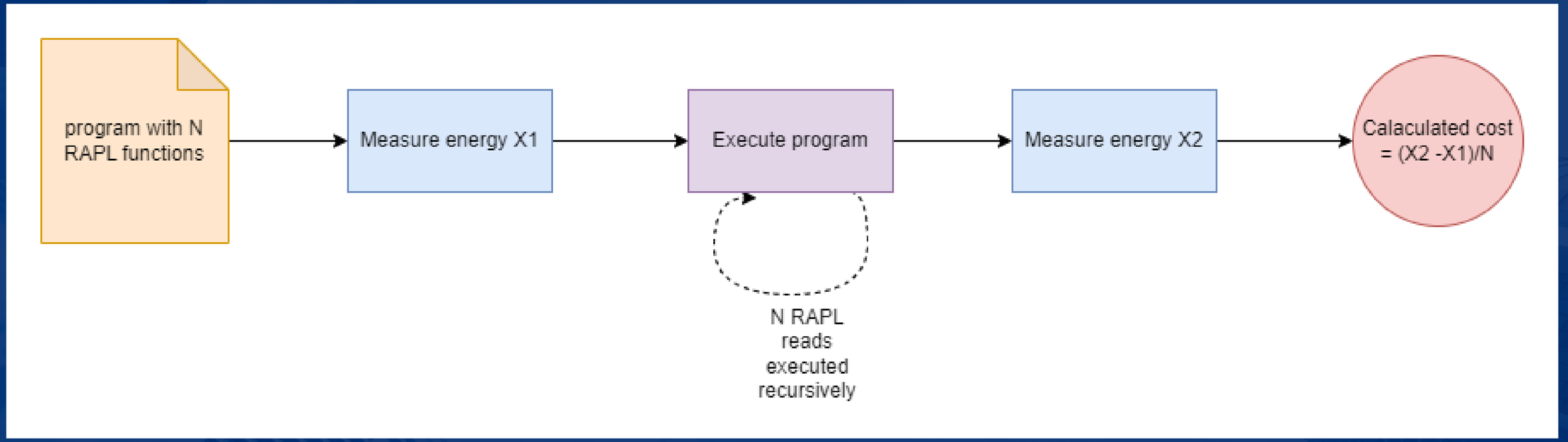
Problem 2: Cost of RAPL overshadowing cost of Basic Block

RAPL read function introduces energy overhead of up to 20X times the energy cost of BB

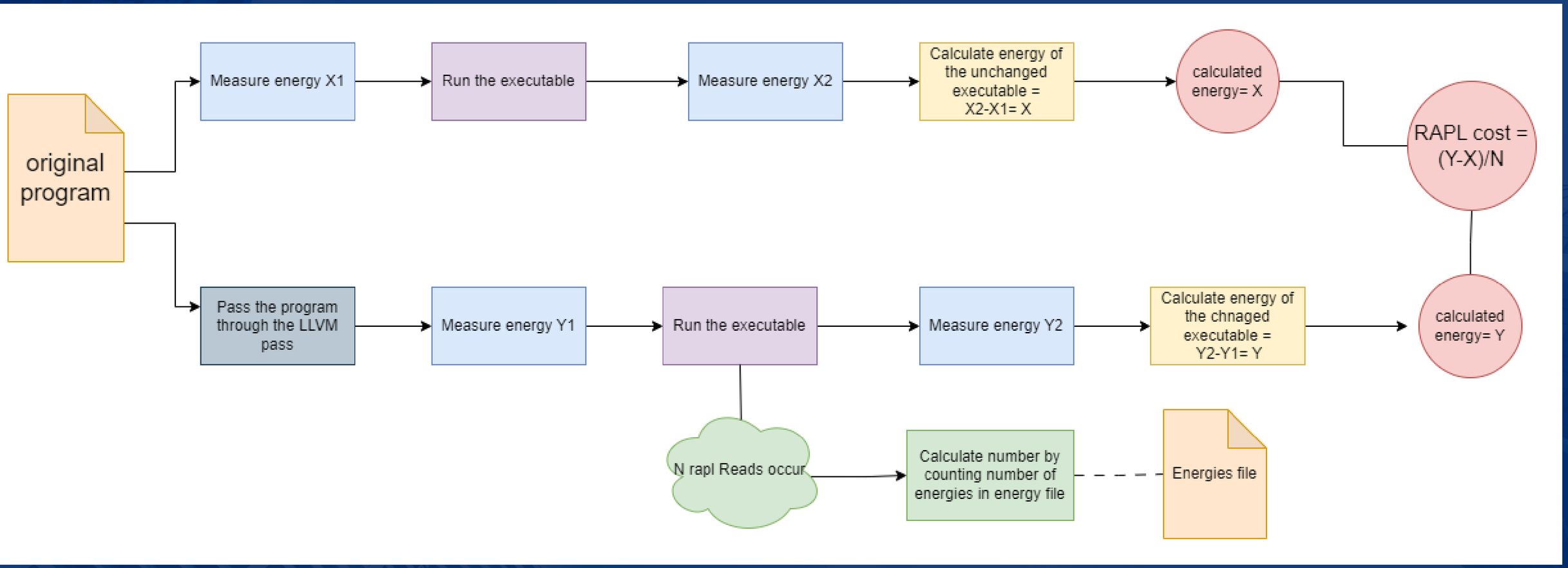
but

It is always the same , so we can calculate it and subtract it

Solution 2: Subtract fixed cost of RAPL read



Solution 2: Subtract fixed cost of RAPL read

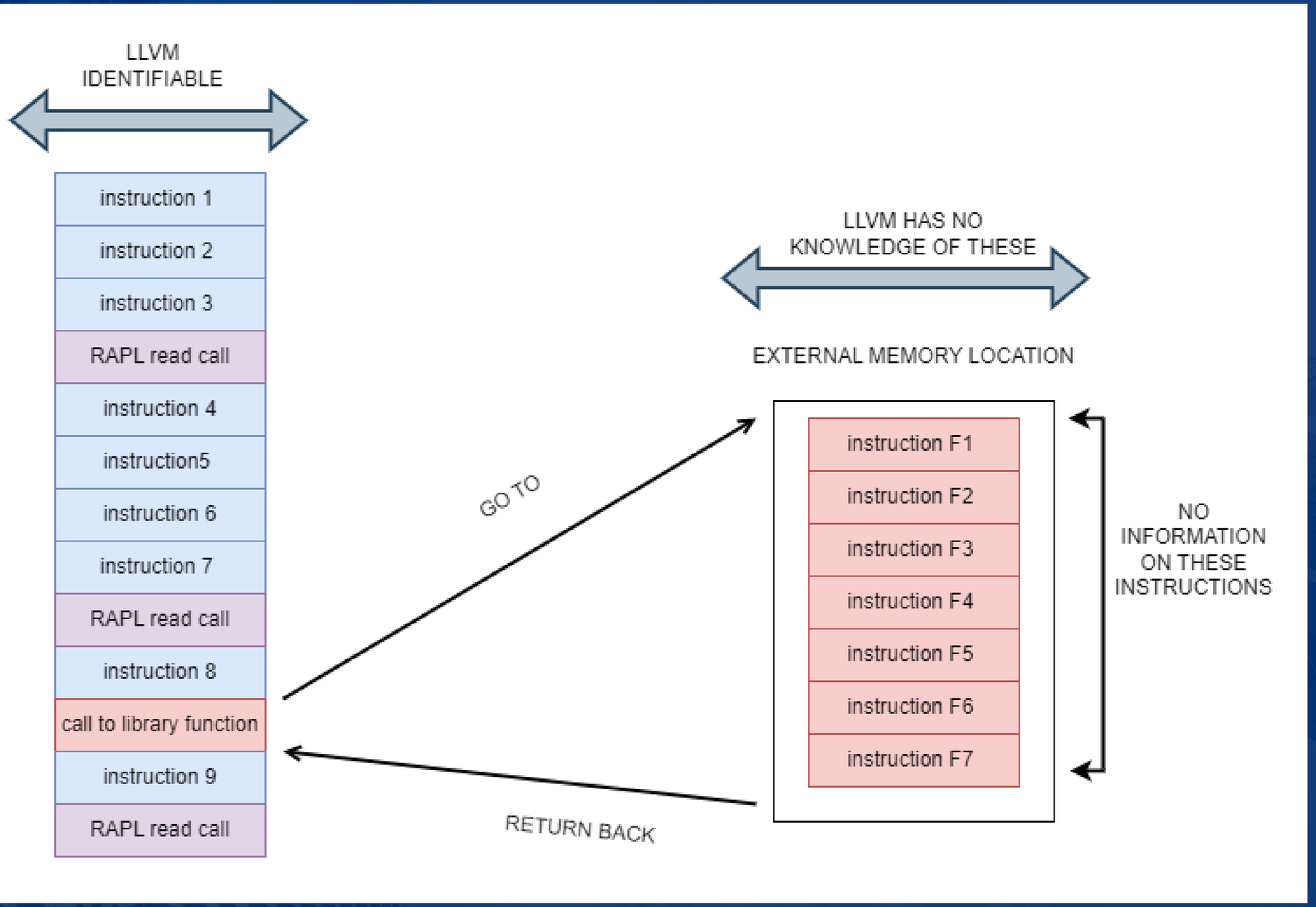


Problem 3: Lost library functions

Library functions are added during linking time and contain bulk of the code

so

LLVM pass cannot analyze their BBs or add RAPL calls


LLVM
IDENTIFIABLE
instruction 1
instruction 2
instruction 3
RAPL read call
instruction 4
instruction5
instruction 6
instruction 7
RAPL read call
instruction 8
call to library function
instruction 9
RAPL read call
LLVM HAS NO
KNOWLEDGE OF THESE
EXTERNAL MEMORY LOCATION
GO TO
RETURN BACK
instruction F1
instruction F2
instruction F3
instruction F4
instruction F5
instruction F6
instruction F7
NO
INFORMATION
ON THESE
INSTRUCTIONS

Solution 3: find the code of the library functions

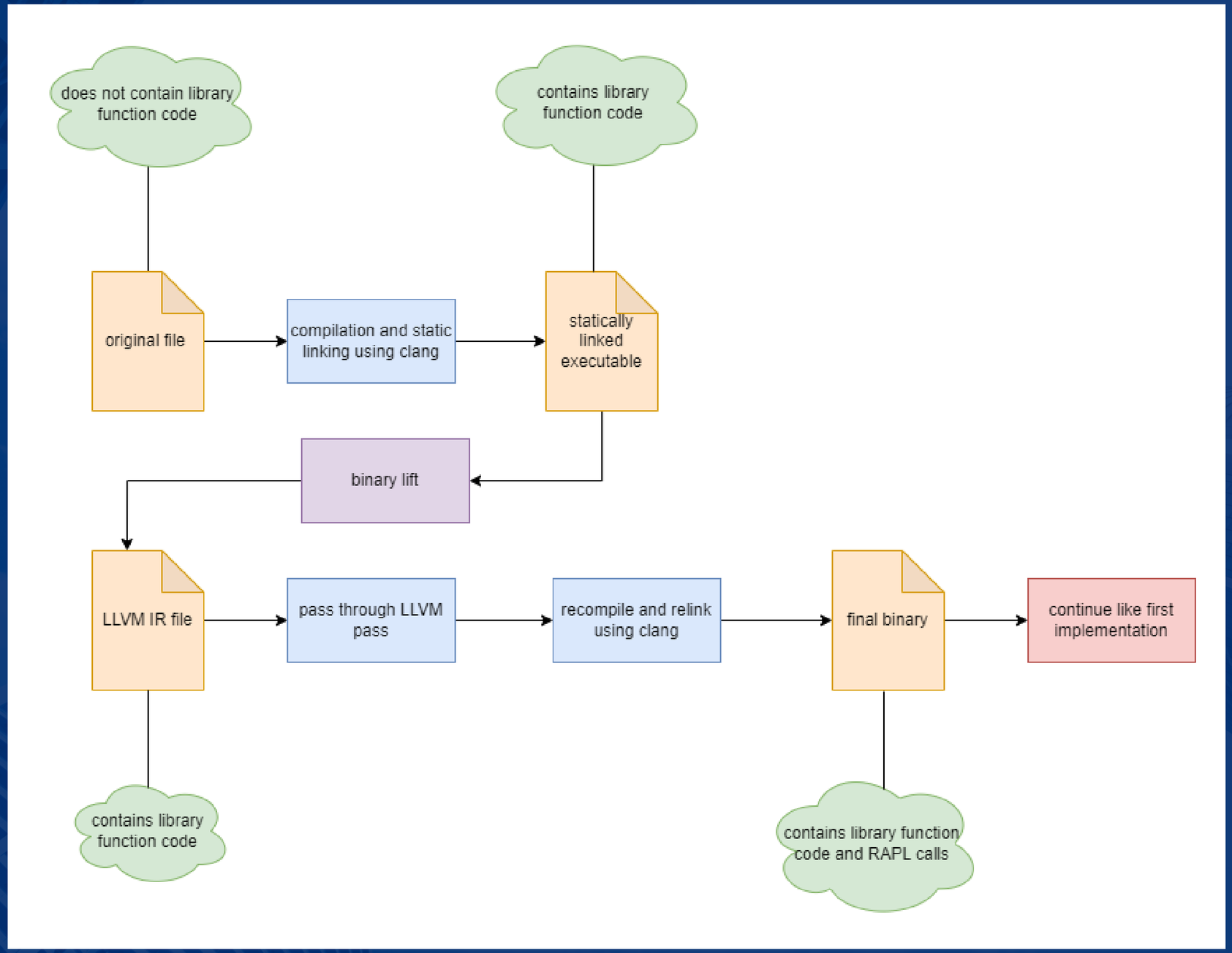
Could not find viable solution so we were lead to Approach 2

Second approach : binary lift of linked executable to LLVM IR

- LLVM pass can only be performed on LLVM IR
- Library function code is added during link time

so

- Transform linked executable back to IR
- Pass it trhough the LLVM pass


does not contain library function code
contains library function code
original file
compilation and static linking using clang
statically linked executable
binary lift
LLVM IR file
pass through LLVM pass
recompile and relink using clang
final binary
continue like first implementation
contains library function code
contains library function code and RAPL calls

Problem : lifted binary is not executable after it passes through the LLVM pass

Used 3 different binary lifting tools

Managed to lift executable and pass it through LLVM pass
Resulting executable always failed with segmentation fault

Must find other way to have access to library function code ...

Third approach : Execution tracing using Intel perf

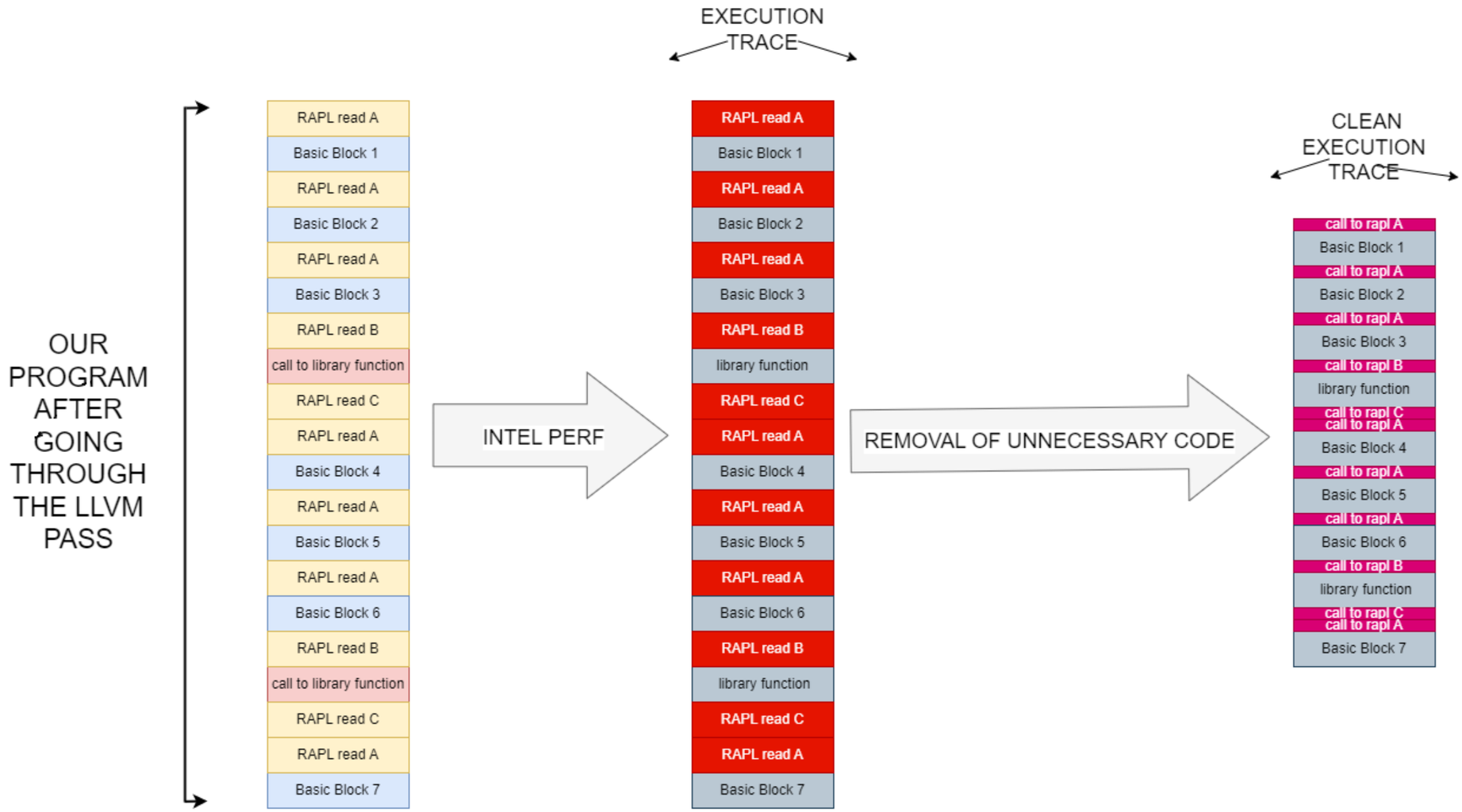
- Insert 3 types of RAPL calls
- Run executable and collect trace
- Remove RAPL call energy and code
- Analyze trace corresponding energy to BBS or functions
- Split functions into BBs
- Split function energy to its BBs
- aggregate duplicate BBs

Problem 1: Remove RAPL code and energy

Remove RAPL energy as seen before

and

Remove RAPL code from trace since it is always the same


OUR PROGRAM AFTER GOING THROUGH THE LLVM PASS
RAPL read A
Basic Block 1
RAPL read A
Basic Block 2
RAPL read A
Basic Block 3
RAPL read B
call to library function
RAPL read C
RAPL read A
Basic Block 4
RAPL read A
Basic Block 5
RAPL read A
Basic Block 6
RAPL read B
call to library function
RAPL read C
RAPL read A
Basic Block 7
INTEL PERF
EXECUTION TRACE
RAPL read A
Basic Block 1
RAPL read A
Basic Block 2
RAPL read A
Basic Block 3
RAPL read B
library function
RAPL read C
RAPL read A
Basic Block 4
RAPL read A
Basic Block 5
RAPL read A
Basic Block 6
RAPL read B
library function
RAPL read C
RAPL read A
Basic Block 7
REMOVAL OF UNNECESSARY CODE
CLEAN EXECUTION TRACE
call to rapl A
Basic Block 1
call to rapl A
Basic Block 2
call to rapl A
Basic Block 3
call to rapl B
library function
call to rapl C
call to rapl A
Basic Block 4
call to rapl A
Basic Block 5
call to rapl A
Basic Block 6
call to rapl B
library function
call to rapl C
call to rapl A
Basic Block 7

Problem 2: Split energy of functions into their BBS

Split functions into BBs using branch or call instructions as break points

Use method based on execution times of instructions to split energy into newly created BBs

Final Dataset

- 3828 unique Basic Blocks from >565K total
- 24 C benchmarks
- Average energy : 0.64 energy units
- Mean error : 2.63%

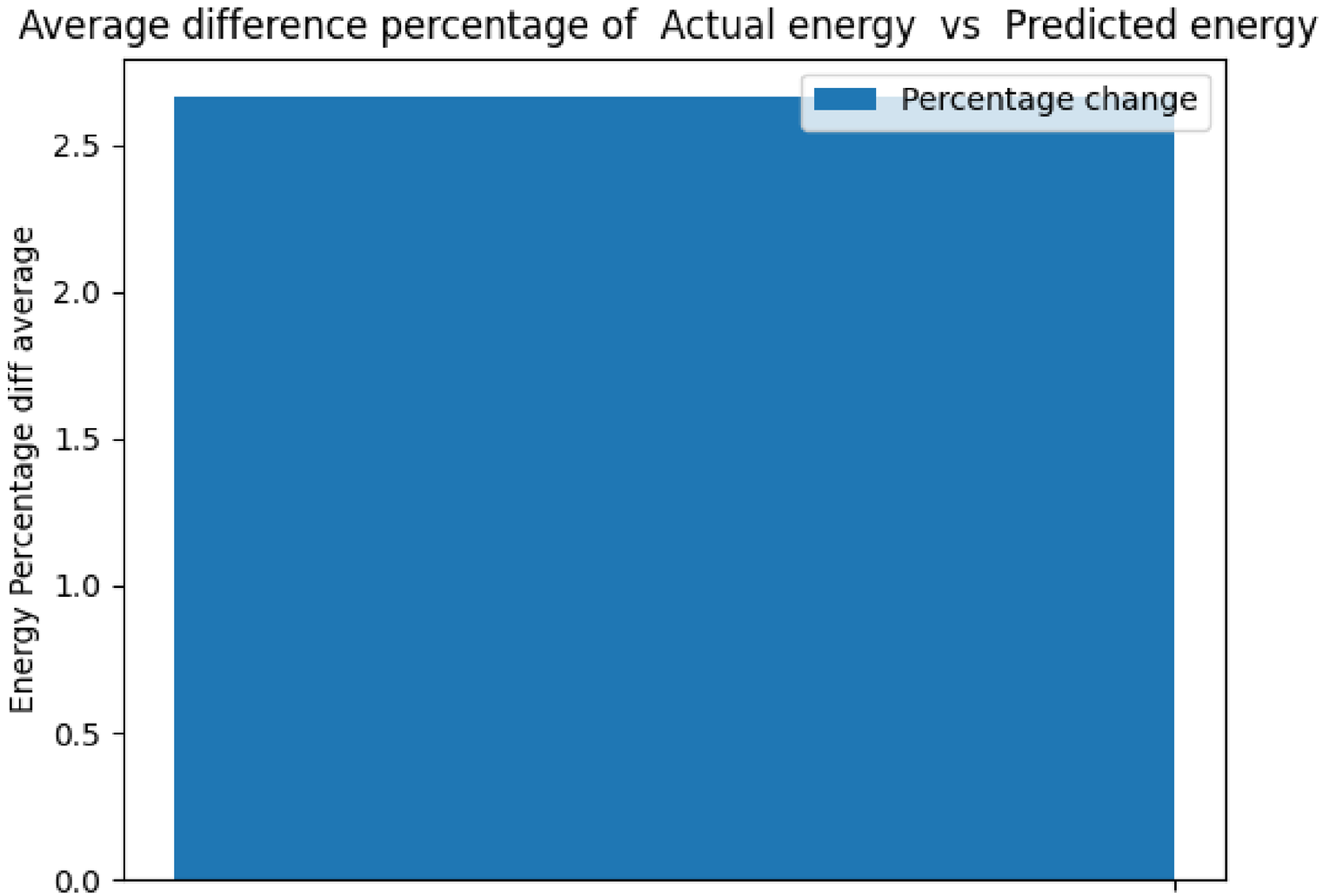
Evaluation Method

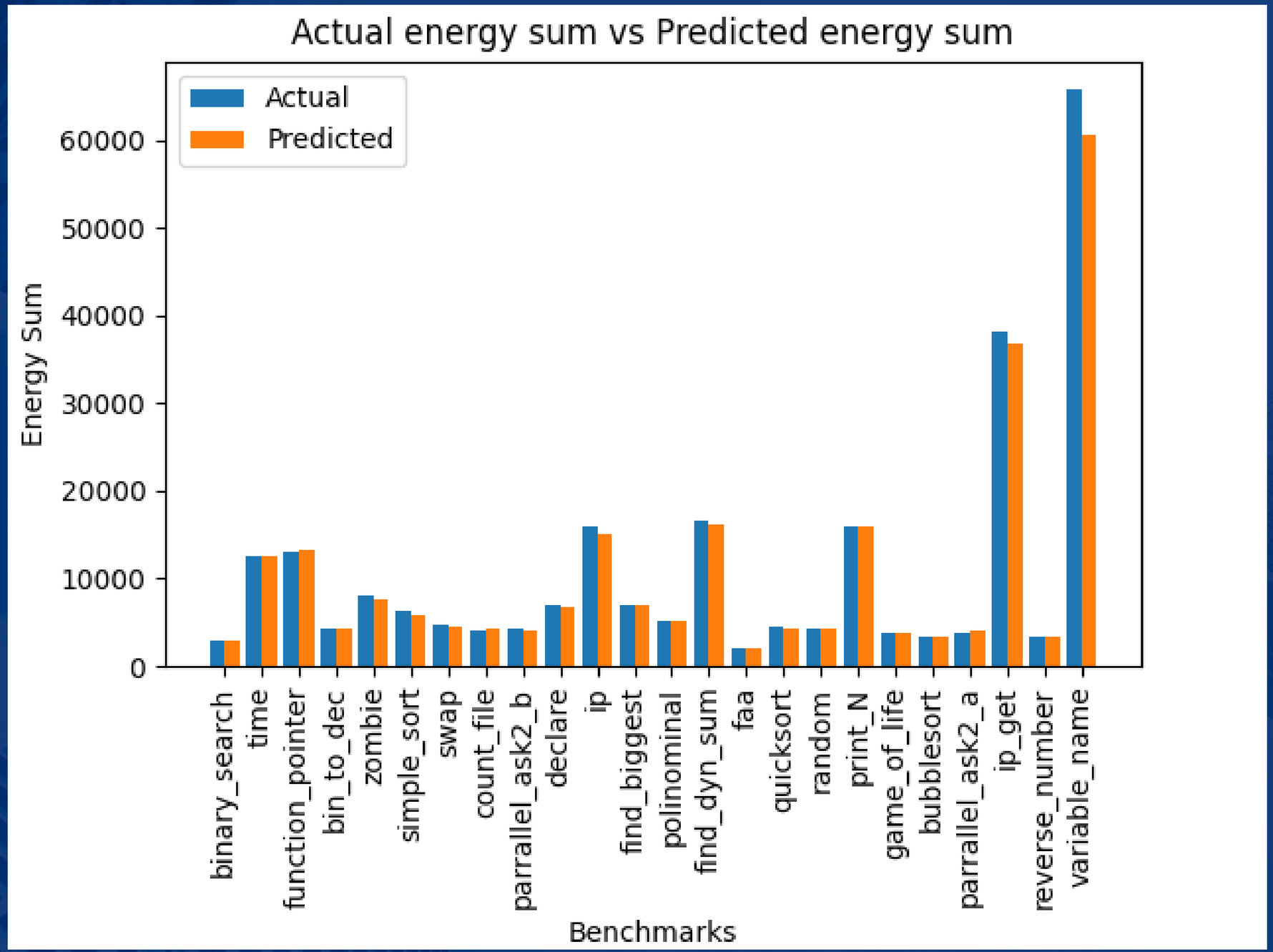
Compared actual energy of 24 benchmarks

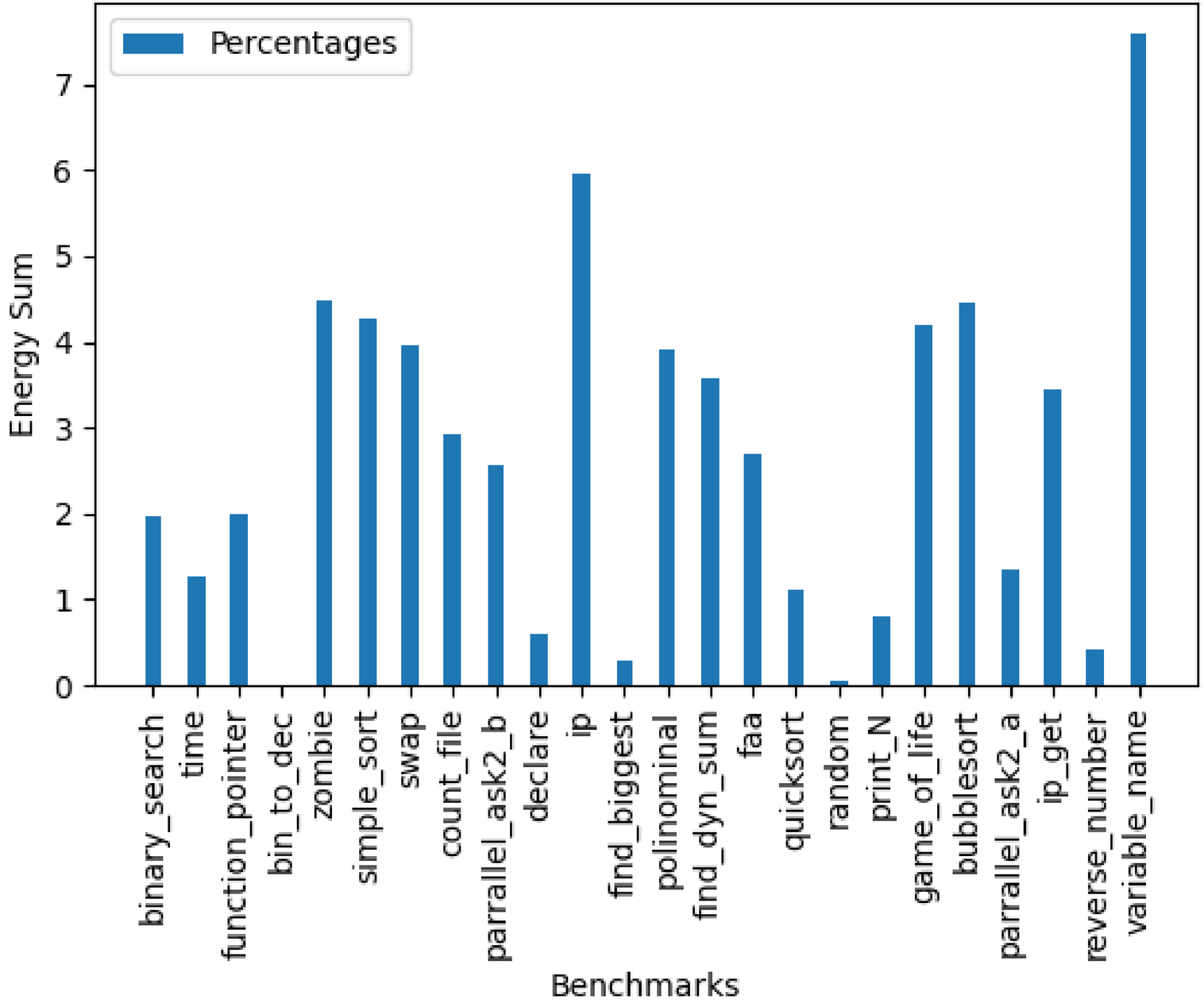
vs

Estimated energy based on our dataset

Results


Average difference percentage of Actual energy vs Predicted energy
Percentage change
Energy Percentage diff average
2.5
2.0
1.5
1.0
0.5
0.0


Actual energy sum vs Predicted energy sum
Actual
Predicted
Energy Sum
0
10000
20000
30000
40000
50000
60000
binary_search
time
function_pointer
bin_to_dec
zombie
simple_sort
swap
count_file
parrallel_ask2_b
declare
ip
find_biggest
polinominal
find_dyn_sum
faa
quicksort
random
print_N
game_of_life
bubblesort
parrallel_ask2_a
ip_get
reverse_number
variable_name
Benchmarks


Percentage difference of predicted vs Actual
Percentages
Energy Sum
0
1
2
3
4
5
6
7
binary_search
time
function_pointer
bin_to_dec
zombie
simple_sort
swap
count_file
parrallel_ask2_b
declare
ip
find_biggest
polinominal
find_dyn_sum
faa
quicksort
random
print_N
game_of_life
bubblesort
parrallel_ask2_a
ip_get
reverse_number
variable_name
Benchmarks

Comparison with similar research

- Comparable error to the state of the art ,ALEA
- Better accuracy than works with similar granularity
- Better granularity than works with similar accuracy
- Open source and reproducable

Current continuation : Energy dataset for memory and GPU

change location of RAPL read register → repeat process for all benchmarks → create new dataset for memory, gpu, or uncore

Current continuation : Energy predicting neural network

dataset

energy predictions for BBs

train

NN

input: high level code

Future work : Energy reducing code transformations

Software energy reduction techniques based on code transformation

find energy costy BBs

replace them with others

Conclusions

- Produced reliable energy dataset on a basic block level
- Custom re-production for each system recommended, easy using our tool
- dataset + NN -> basis for software reduction techniques
- Binary lifting intresting approach needs more research

Contact Us

el17072@ntua.mail.gr - Bouras

+30 6980247969

Thank You

Don't forget to check out the project codebase on github repo :

https://github.com/jimbou/dataset_tool.git