Static Energy Analysis of Low-Level Programs

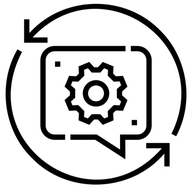
Patrick May (Advisor: Drew Guarnera)

THE COLLEGE OF WOOSTER

Program Analysis

- Observe existing computer programs to improve them
- Make faster, smaller, more accurate, more efficient, etc.

Static Analysis:
Performed “statically”, at compile time or earlier

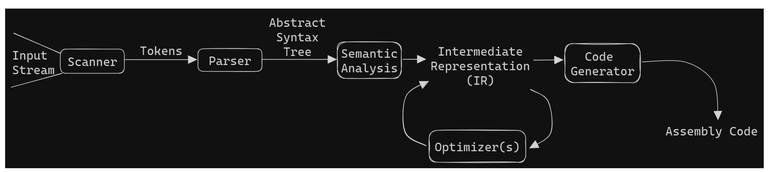


Dynamic Analysis:
During runtime, sometimes “profiling”

Can we use static analysis to predict the energy cost to run a program?

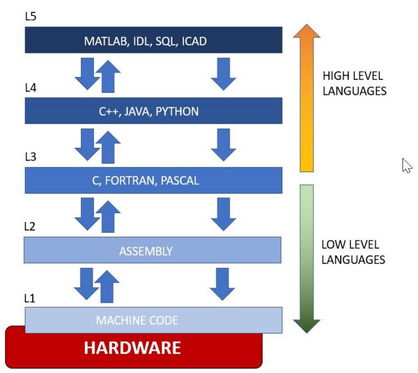
Software Considerations

There are many steps between writing a program and the “code” run on the computer. These steps should be acknowledged in the static analysis process.



Compilation

- Programs are *compiled* (or interpreted) to be run
- Compilers *translate* high level programming languages into a different form, typically *assembly language*
- Abstracts hard and tedious tasks from developer, but also lessens understanding of how a computer works



Hardware Considerations

Programs run on computers that can be drastically different architecturally. Modern processors perform techniques that can affect the end power cost, thus should be acknowledged.

Reduced Instruction Set Computer (RISC)

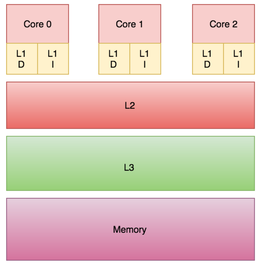
- Simpler instructions
- 1 cycle per instruction
- Chosen platform for project

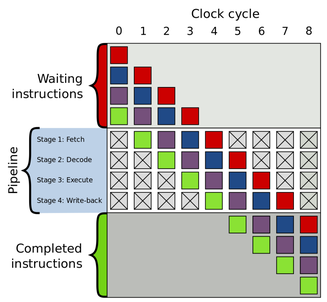
Complex Instruction Set Computer (CISC)

- Larger instructions
- ≥1 cycle(s) per instruction

Cache Hierarchies

- Programs interact with memory
- Caches allow for speedups
- Misses slow down program, increasing power consumption



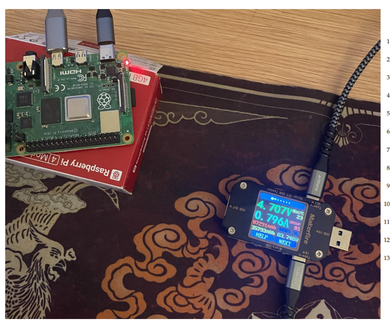


Instruction Pipelines

- Parallelize instruction exec.
- Shorter for RISC CPUs
- Instructions reliant on prior data cause *stalls*, increasing power consumption

Instruction Benchmarking

The static program model requires real world data to get a parameterized real energy prediction. A Raspberry Pi 4 Model B was used to test largely unrolled assembly loops



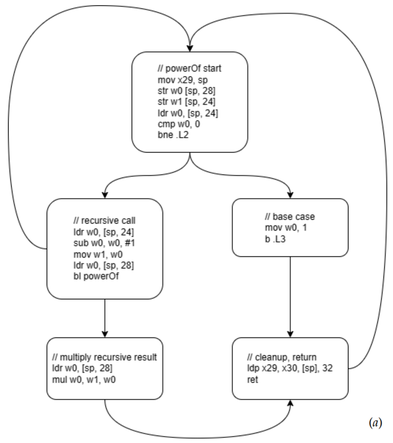
```
    .arch armv8-a   ; various pseudo-ops
    .text
    .align 8
    .global main
    .type main, %function
main:       ; function entrypoint
.LOOP_START:
    .cfi_startproc
    <TESTOP>    <TESTOPERANDS> ; repeated ~2000
    b.al    .LOOP_START ; branch always to loop start
    .cfi_endproc
.LOOP_END:
    ; various end-of-file pseudoops
```

Static Cost Analysis

A mathematical framework and process to translate a program and automatically infer an upper bound cost

1. Source Code

```
int powerOf(int base, int power) {
    if (power == 0) {
        return 1;
    }
    return base * powerOf(base, power - 1);
}
```



3. Control Flow Graph

2. Low Level Representation (ARM Assembly)

```
powerOf:
.LFB0:
    .cfi_startproc
    stp x29, x30, [sp, -32]!   //,,,
    .cfi_def_cfa_offset 32
    .cfi_offset 29, -32
    .cfi_offset 30, -24
    mov x29, sp //,
    str w0, [sp, 28]    // base, base
    str w1, [sp, 24]    // power, power
// power.c:3:     if (power == 0) {
    ldr w0, [sp, 24]    // tmp94, power
    cmp w0, 0   // tmp94,
    bne .L2     //,
// power.c:4:         return 1;
    mov w0, 1   // _3,
    b   .L3     //
.L2:
// power.c:6:     return base * powerOf(base, power - 1);
    ldr w0, [sp, 24]    // tmp95, power
    sub w0, w0, #1  // _1, tmp95,
    mov w1, w0  // _1
    ldr w0, [sp, 28]    // , base
    bl  powerOf     //
    mov w1, w0  // _2,
// power.c:6:     return base * powerOf(base, power - 1);
    ldr w0, [sp, 28]    // tmp96, base
    mul w0, w1, w0  // _3, _2, tmp96
.L3:
// power.c:7: }
    ldp x29, x30, [sp], 32  //,,,
    .cfi_restore 30
    .cfi_restore 29
    .cfi_def_cfa_offset 0
    ret
    .cfi_endproc
```

4. Cost Relation Form

$$
\begin{aligned}
&(a) \quad C_{powerOf}(A,B) = k_1 + C_{powerOf}(A,B) && \{B \geq 0\}\\
&(b) \quad C_{powerOf}(A,B) = k_1 + C_{base}(A,B) && \{B \geq 0\}\\
&(c) \quad C_{recur}(A,B) = k_2 + C_{powerOf}(A,B-1) + k_3 + C_{cleanup}(A,B) && \{B \geq 1\}\\
&(d) \quad C_{cleanup}(A,B) = k_4 && \{B \geq 0\}\\
&(e) \quad C_{base}(A,B) = k_5 + C_{cleanup}(A,B) && \{B = 0\}\\
&\qquad k_1 := \langle MOV \rangle + 2 \times \langle STR \rangle + \langle LDR \rangle + \langle CMP \rangle + \langle BNE \rangle\\
&\qquad k_2 := \langle LDR \rangle + \langle SUB \rangle + \langle MOV \rangle + \langle LDR \rangle\\
&\qquad k_3 := \langle LDR \rangle + \langle MUL \rangle\\
&\qquad k_4 := \langle LDP \rangle + \langle RET \rangle\\
&\qquad k_5 := \langle MOV \rangle + \langle B \rangle
\end{aligned}
$$

5. Solved in PUBS

- Practical Upper Bound Solver
- Prolog Logic System

Alternative Analysis Methods

- Dynamic block execution analysis
- Different fundamental cost expression selection

Future Work

- Alternative classification of basic costs
- Automation of cost relation extraction
- Testing for more forms of instruction execution
- Experimentation on alternative hardware platforms
- Exploration of alternative analytical frameworks