Intermediate Presentation

Design of Digital Platforms

Steven Janssens
Pieter Maene
Kristof Mariën

Content

- C
  - Performance Analysis
  - Performance Strategy
- Assembly
  - Performance Analysis
  - Performance Strategy
- Improvements in performance

C - Performance Analysis

- Many calls of ADD function
- Use of function parameters

- Calculation for lower and higher part of char
- Use of (small) arrays is slow

- More variables cost more memory

C – Performance Strategy

- Implement ADD function directly in fips
- Use of global variables

- Change two chars to short (→ more efficient)
- Replacement of some variables (→ less memory)
- Replace array t[3] by two short variables

  (→ reuse of variables)

- Variable to #define (n_prime)

Char → Short & Reuse

```
tmp = t[0] + a[j]*b[i-j];
s = tmp;
c = tmp >> 8;
tmp = t[1] + c;
t[1] = tmp;
t[2] = t[2] + (tmp >> 8);
```

```
tmp = (unsigned char)
(tmp)  + a[j]*b[i-j];
t = t + (tmp >> 8);
```

```
tmp =  (unsigned char)
(tmp) + m[j]*n[i-j];
t = t + (tmp >> 8);
```

Assembly – Performance Analysis

- Uses a lot of registers

- Inefficient compilation
  - mov r5, #0x00
    mov a, r5

Assembly – Performance Strategy

- Rewrite all code to assembly
- Save result of i-j
- Load n[0] and b[0] once
- More efficient translation
  - mov r5, #0x00
    mov a, r5

    → clr a

Improvements in Performance

- Time before optimization
  - 3.339131 seconds
- Time after c optimization
  - 1.438686 seconds
- Time after assembly optimization
  - 0.974628 seconds

Remarks

- Hard coded values
  - SIZE
  - n_prime

- Rewriting small parts to inline assembly

  → not efficient