XJTU-ICS Lab 4 – Optimization Lab

Ze Xia

xiaze2017@stu.xjtu.edu.cn

Optimization Techniques

- Good algorithm with low time & space complexity
  - very important, but out of scope of this course

- Dead code elimination
- Function inlining
- Reducing unnecessary memory accesses
- **Loop unrolling!**

Machine Dependent Optimization

ARMv8-A (e.g. Kunpeng 920)

- mem access: load/store only
- register: 31 GP, 32 FP

- `FMADD Dd, Dn, Dm, Da`

  ```
  Dd = Dn * Dm + Da
  ```

x86-64 (e.g. Haswell)

- mem access: any operand
- register: 16 GP, 16 FP

- MULSD + ADDSD: 2 issues!

Kunpeng 920

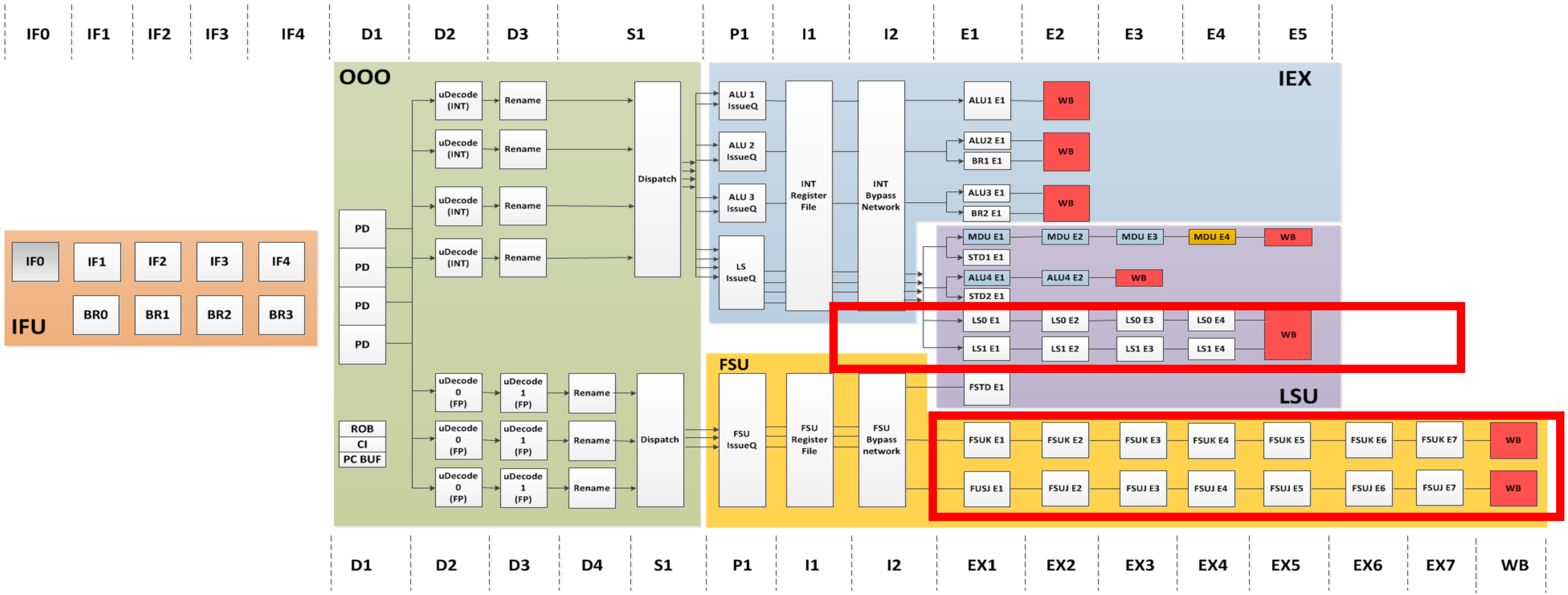
- Huawei's self-designed, ARMv8 based micro-architechture
- Up to 64 cores
- 4-issues per cycle

  Cortex-A72: 3
- 64 KB L1i/L1d cache

  Cortex-A72: 32 KB
- Exclusive L2 cache for each core, 9 cycles latency

  Cortex-A72: 4-Core shared L2 cache, 18 cycles latency

Kunpeng 920 Pipeline



Taishan coreV110 Pipeline Architecture

Timing to optimize

If the code you're going to optimize:

- is the system's bottleneck
- is using the right algorithm

Then you can start the happy machine-dependent optimizing!

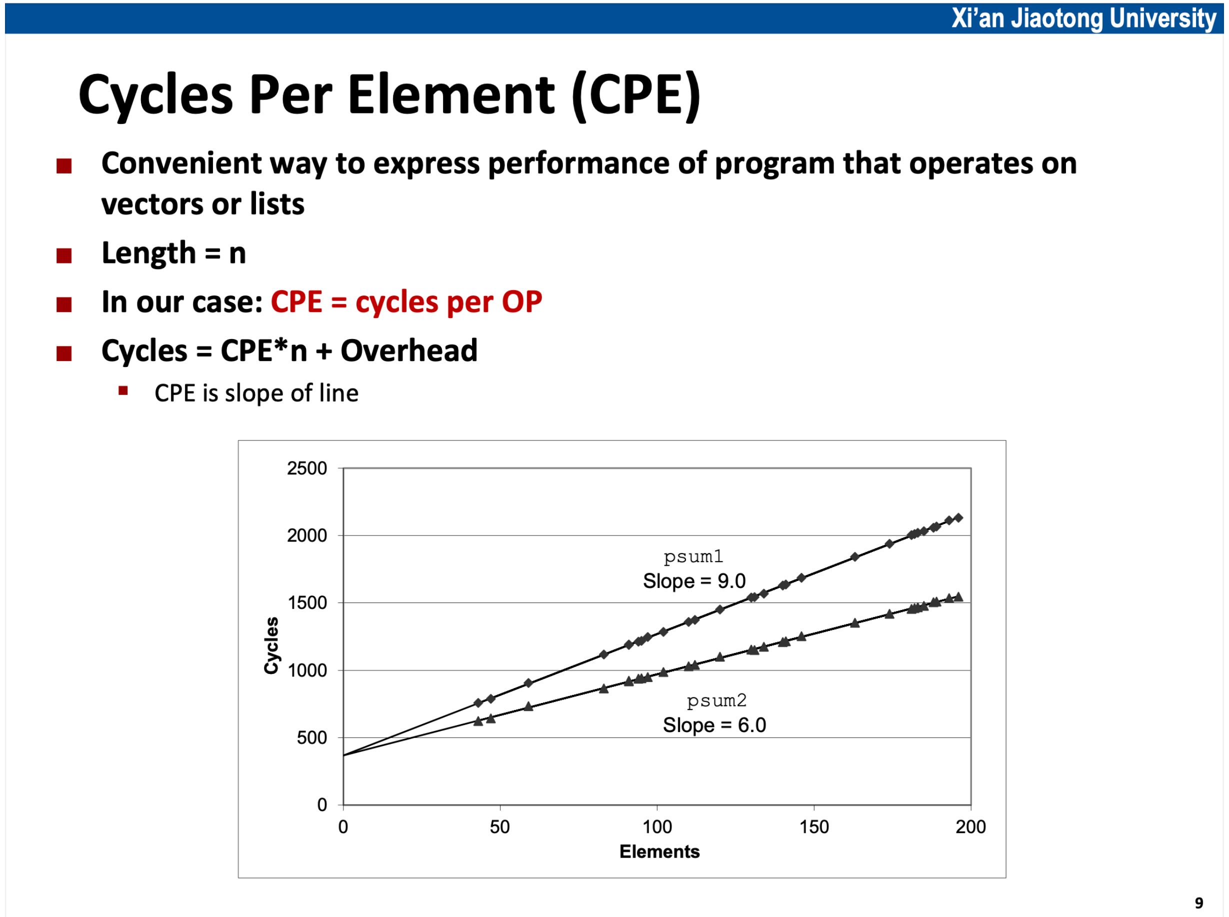
Otherwise: giving up readability, development speed, ... for nothing

Cycles Per Element

Bounded by:

- Latency bound
- Throughput bound

Check manual of the target machine!



Xi'an Jiaotong University

Cycles Per Element (CPE)

- **Convenient way to express performance of program that operates on vectors or lists**
- **Length = n**
- **In our case: CPE = cycles per OP**
- **Cycles = CPE*n + Overhead**
  - CPE is slope of line

2500
2000
1500
1000
500
0
Cycles
0
50
100
150
200
Elements
psum1
Slope = 9.0
psum2
Slope = 6.0

This lab: Measuring CPE

Your work:

- Measure execution time
- Optimize, then measure again

Our work:

- Compute CPE from your measurement
- Lock CPU frequency

Start of the story

$$\begin{aligned} poly(x) &= a_0 + a_1x + a_2x^2 + \ldots + a_nx^n \\ &= (((a_nx + a_{n-1})x + a_{n-2})x + \ldots + a_1)x + a_0 \end{aligned}$$

```
void poly(const double a[], double x, long degree, double *result) {
    long i;
    double r = a[degree];
    for (i = degree - 1; i >= 0; i--) {
        r = a[i] + r * x;
    }
    *result = r;
}
```

Task

Measure the CPE

- Reference CPE: **7.0**
- Why? (See assembly and the manual.)

Optimize, then measure again

- Reference CPE after optimization: **1.0**
- How to achieve this? Why not lower?

Time measuring

Choose an appropriate clock function:

- `clock()`
- `gettimeofday()`
- `clock_gettime()`

Watch the clock before and after, take the difference

Keep your cache hot

Cache miss may affect CPE measurement a lot

Who can mess up your cache?

- OS Scheduling
- Heavy library calls
- Testing script ☺

How to keep cache hot: run measured function once before measuring

Misc

- OS Scheduling sometimes happen, our testing script picks out outliers using RANSAC algorithm
- Run many times, take the average

Tips

- Read `main.c`, modify on need
- Visualize your measurement results
- Check the assembly and ARM manual, make sure you know what's happening
- Keep thinking and digging

Dig deeper...

- Longer polynomial coefficient list
- Computing polynomial value at multiple x-s simultaneously
- SIMD
- Larger/smaller x, causing floating-point overflow/underflow
- Other implementation of `poly()`
  (See CSAPP: Exercise 5.5 & 5.6)
- Do again on x86-64 machine

We can do more!

This powerful method enables us to measure:

- Latency?
- Throughput?
- # of Functional Units?

... of an instruction

FAQ

SSH

- `ssh <id>-ics@ics-arm.dfshan.net`
- First sign-in by command line, change the password, then vscode-ssh
- `"Current Password"` means the old password
- Under campus network