2023 CCF CHINASOFT

中国软件大会

智能化软件创新推动数字经济与社会发展

NASAC 第22届全国软件与应用学术会议
The 22nd National Software and Application Conference

FMAC 第8届全国形式化方法与应用会议
The 8th National Conference on Formal Method and Application

中国·上海 2023年12月1-3日
SHANGHAI·CHINA December 1-3, 2023

2023 CCF CHINASOFT 中国软件大会

软件系统性能优化的教学与实践

报告人：郭健美

报告日期：2023.12.1

汇报提纲

- **教学I：软件系统优化**
- 教学II：计算机系统
- 实践I：跨平台的硬件性能计数器自适应分组复用 [TACO’23]
- 实践II：不同编译器优化能力的分析与识别 [CC’23]

为何开设“软件系统优化”课？

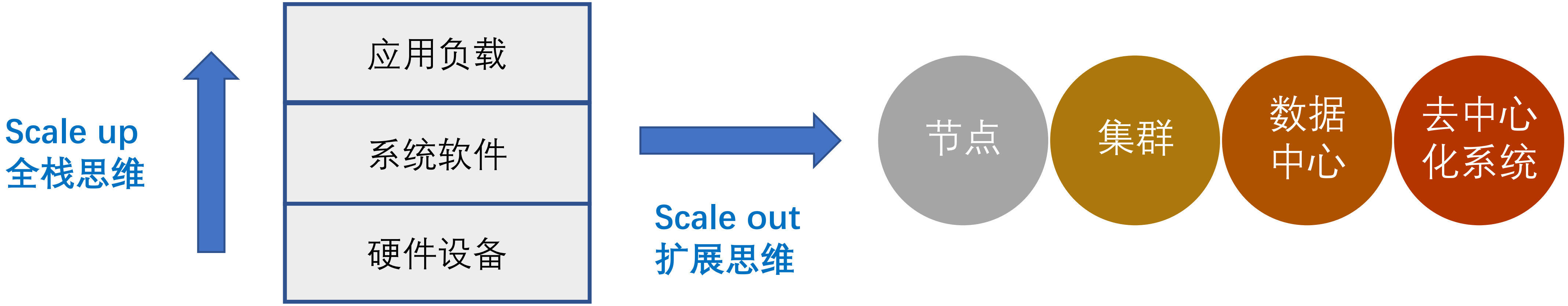
- 性能是**衡量软件系统质量和竞争力**的一个重要方面，是软件系统设计、开发和应用过程中必须关注的一个基本属性。如何在给定的硬件资源配置下提升软件系统的性能，是数字化系统的设计和实现必须思考和解决的问题，同时也是优化利用软硬件资源的有效途径。
- 软件系统优化的原理、技术和实践是一位卓越的软件系统工程师、架构师或研究人员**必备的素养**。发起软件系统优化方面的课程设置和教学是解决我国计算机系统方面“卡脖子”问题人才培养的有效措施。

Table 1. Speedups from performance engineering a program that multiplies two 4096-by-4096 matrices. Each version represents a successive refinement of the original Python code. “Running time” is the running time of the version. “GFLOPS” is the billions of 64-bit floating-point operations per second that the version executes. “Absolute speedup” is time relative to Python, and “relative speedup,” which we show with an additional digit of precision, is time relative to the preceding line. “Fraction of peak” is GFLOPS relative to the computer’s peak 835 GFLOPS. See Methods for more details.

Version	Implementation	Running time (s)	GFLOPS	Absolute speedup	Relative speedup	Fraction of peak (%)
1	Python	25,552.48	0.005	1	—	0.00
2	Java	2,372.68	0.058	11	10.8	0.01
3	C	542.67	0.253	47	4.4	0.03
4	Parallel loops	69.80	1.969	366	7.8	0.24
5	Parallel divide and conquer	3.80	36.180	6,727	18.4	4.33
6	plus vectorization	1.10	124.914	23,224	3.5	14.96
7	plus AVX intrinsics	0.41	337.812	62,806	2.7	40.45

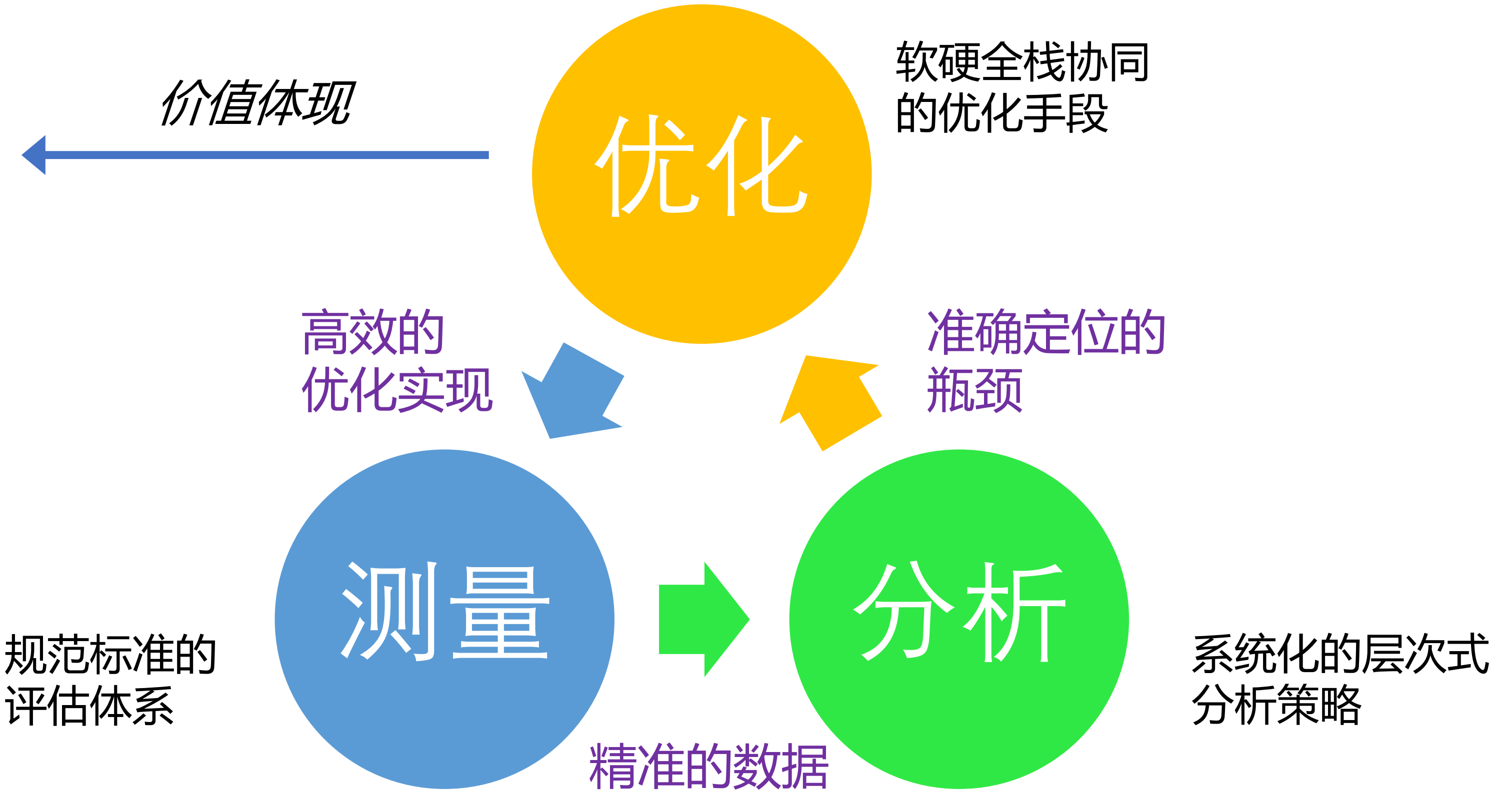
[C. E. Leiserson et al., There’s plenty of room at the Top: What will drive computer performance after Moore’s law? Science 368, eaam9744 (2020)]

培养“系统观”：从单点到全局的系统思维



数据驱动的系统优化方法论

- 性能提升
- 能耗降低
- 存储减少
- 资源利用率提升
- 运营成本降低
- 架构和系统创新



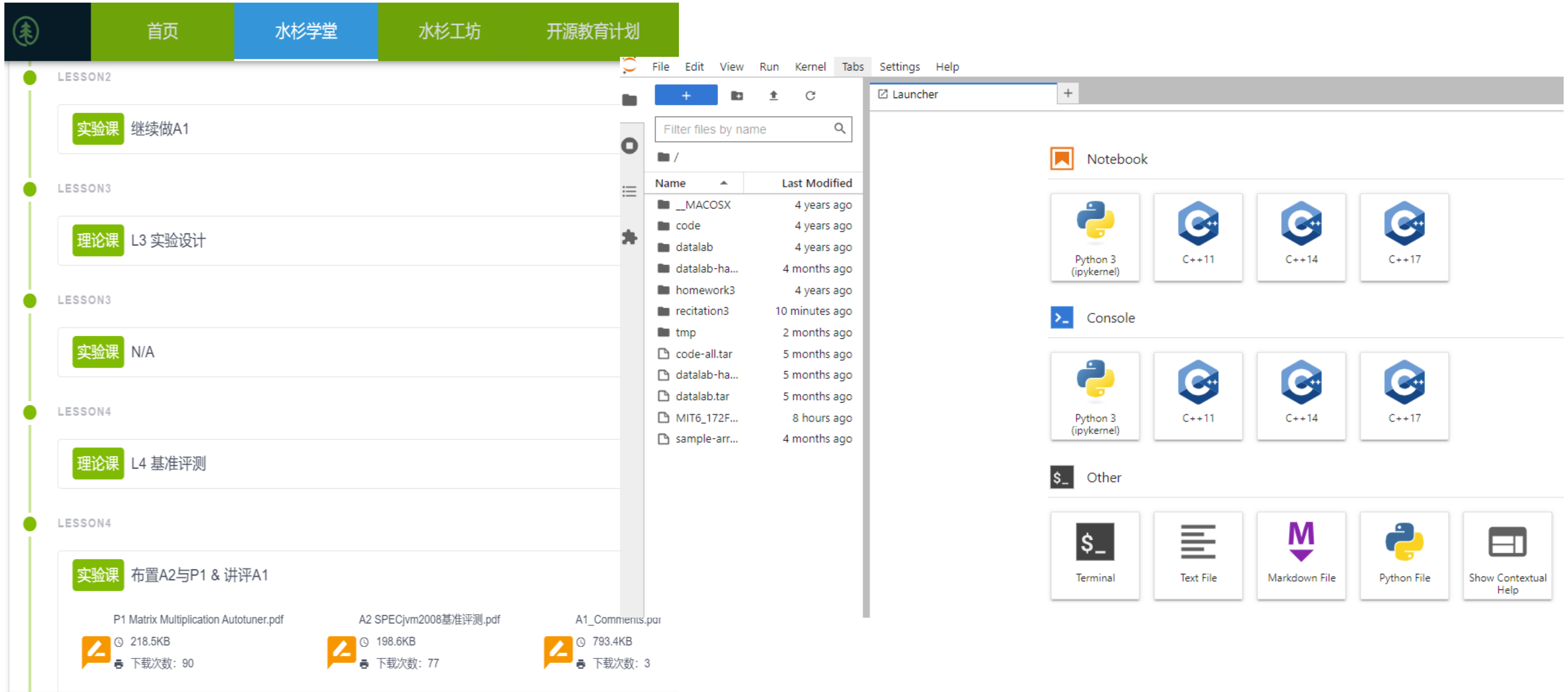
课程设置

- 专业选修课
- 面向大三、大四
- 2021年秋开设
- 2023年立项上海高校市级重点课程

周	理论课		实践课	
	模块	主题	上机作业（2周/个）	实践项目（4周/个）
1	绪论	课程介绍、矩阵乘法优化案例	A1 初试环境和工具	
2	性能工程基础	性能测量		P1 Matrix Multiplication Autotuner
3		配置优化	1. A2 SPECjvm2008基准测试 2. A1 提交	
4		基准评测		
5		性能评价	1. A2 提交 2. A1-QA & Check	
6	编译优化	源程序级别的常见优化方法	A2-QA & Check	P1 提交
7		编译器概述	A3 GCC与Clang /LLVM优化比较	1. P1-QA & Check 2. P2 交叉编译与跨平台应用仿真
8		目标指令集架构及汇编语言		
9		C程序的汇编代码生成	A3 提交	
10		编译器的优化能力	1. A3-QA & Check 2. A4 Vectorization	
11		程序插桩及优化机会识别		P2 提交
12	计算机体系结构优化	计算机体系结构概论	A4 提交	1. P2-QA & Check 2. P3 Profiling Serial Merge Sort
13		多核异构编程	1. A4-QA & Check 2. A5 oneAPI异构编程	
14		高速缓存及相关优化		
15		微体系结构性能分析方法	A5 提交	
16	前沿研究和应用	数据中心和云端服务优化	A5-QA & Check	P3提交
17		机器学习框架优化		P3-QA & Check

感想1：加强动手能力培养，开展即时反馈的实践课

- 采用项目作业“布置-提交-反馈”模式，开展**即时反馈的实践课**学习，切实提高学生的动手能力
- 借助**“水杉在线”**实训平台，每位学生可独占一个统一规格和镜像的云实例
- 补上“计算机教育中缺失的一课”，加强**常用工具**的实训



计算机教育中缺失的一课

The Missing Semester of Your CS Education 中文版

大学里的计算机课程通常专注于讲授从操作系统到机器学习这些学院派的课程或主题，而对于如何精通工具这一主题则往往会留给学生自行探索。在这个系列课程中，我们讲授命令行、强大的文本编辑器的使用、使用版本控制系统提供的多种特性等等。学生在他们受教育阶段就会和这些工具朝夕相处（在他们的职业生涯中更是这样）。

因此，花时间打磨使用这些工具的能力并能够最终熟练地、流畅地使用它们是非常有必要的。

精通这些工具不仅可以帮助您更快的使用工具完成任务，并且可以帮助您解决在之前看来似乎无比复杂的问题。

关于 开设此课程的动机。

日程

文档同步时间 2021-04-24

- 1/13: 课程概览与 shell Solution ✓ Update ✓ Chinese ✓
- 1/14: Shell 工具和脚本 Solution ✓ Update ✓ Chinese ✓
- 1/15: 编辑器 (Vim) Solution ✓ Update ✓ Chinese ✓
- 1/16: 数据整理 Solution ✓ Update ✓ Chinese ✓
- 1/21: 命令行环境 Solution ✓ Update ✓ Chinese ✓
- 1/22: 版本控制(Git) Solution ✓ Update ✓ Chinese ✓
- 1/23: 调试及性能分析 Solution ✓ Update ✓ Chinese ✓
- 1/27: 元编程 Solution ✓ Update ✓ Chinese ✓
- 1/28: 安全和密码学 Solution ✓ Update ✓ Chinese ✓
- 1/29: 大杂烩 Solution ✗ Update ✓ Chinese ✓
- 1/30: 提问&回答 Solution ✗ Update ✓ Chinese ✓

https://missing.csail.mit.edu/

SOLE 系统优化实验室 华东师范大学

感想2：加强对汇编代码的理解，知其所以然

example1.c

```c
#include <stdint.h>
#include <stdlib.h>
#include <math.h>

#define SIZE (1L << 16)

void test(uint8_t * a,  uint8_t * b) {
  uint64_t i;

  for (i = 0; i < SIZE; i++) {
    a[i] += b[i];
  }
}
```

感想2：加强对汇编代码的理解，知其所以然

example1.c

```c
#include <stdint.h>
#include <stdlib.h>
#include <math.h>

#define SIZE (1L << 16)

void test(uint8_t * a,  uint8_t * b) {
  uint64_t i;

  for (i = 0; i < SIZE; i++) {
    a[i] += b[i];
  }
}
```

```
$clang -Wall -g -std=gnu99
-O3 -Rpass=loop-vectorize
-Rpass-missed=loop-vectorize
-S -c example1.c
```

启用循环向量化选项，生成汇编代码

example1.s

```
    #DEBUG_VALUE: test:b <- $rsi
    #DEBUG_VALUE: test:a <- $rdi
    .loc    1 0 3                   # example1.c:0:3
    xorl    %eax, %eax
.Ltmp7:
    .p2align    4, 0x90
.LBB0_3:                                # =>This Inner Loop Header: Depth=1
    #DEBUG_VALUE: test:i <- 0
    #DEBUG_VALUE: test:b <- $rsi
    #DEBUG_VALUE: test:a <- $rdi
    .loc    1 13 13 is_stmt 1       # example1.c:13:13
    movdqu  (%rsi,%rax), %xmm0
    movdqu  16(%rsi,%rax), %xmm1
    .loc    1 13 10 is_stmt 0       # example1.c:13:10
    movdqu  (%rdi,%rax), %xmm2
    paddb   %xmm0, %xmm2
    movdqu  16(%rdi,%rax), %xmm0
    paddb   %xmm1, %xmm0
    movdqu  32(%rdi,%rax), %xmm1
    movdqu  48(%rdi,%rax), %xmm3
    movdqu  %xmm2, (%rdi,%rax)
    movdqu  %xmm0, 16(%rdi,%rax)
    .loc    1 13 13                 # example1.c:13:13
    movdqu  32(%rsi,%rax), %xmm0
    .loc    1 13 10                 # example1.c:13:10
    paddb   %xmm1, %xmm0
    .loc    1 13 13                 # example1.c:13:13
    movdqu  48(%rsi,%rax), %xmm1
    .loc    1 13 10                 # example1.c:13:10
    paddb   %xmm3, %xmm1
    movdqu  %xmm0, 32(%rdi,%rax)
    movdqu  %xmm1, 48(%rdi,%rax)
.Ltmp8:
    .loc    1 12 26 is_stmt 1       # example1.c:12:26
    addq    $64, %rax
    cmpq    $65536, %rax            # imm = 0x10000
    jne .LBB0_3
```

感想2：加强对汇编代码的理解，知其所以然

example1.c

```c
#include <stdint.h>
#include <stdlib.h>
#include <math.h>

#define SIZE (1L << 16)

void test(uint8_t * a,  uint8_t * b) {
  uint64_t i;

  for (i = 0; i < SIZE; i++) {
    a[i] += b[i];
  }
}
```

```
$clang -Wall -g -std=gnu99
-O3 -Rpass=loop-vectorize
-Rpass-missed=loop-vectorize
-S -c example1.c
```

启用循环向量化选项，生成汇编代码

比较和分析不同汇编代码的差异

example1.s

```
    #DEBUG_VALUE: test:b <- $rsi
    #DEBUG_VALUE: test:a <- $rdi
    .loc    1 0 3                       # example1.c:0:3
    xorl    %eax, %eax
.Ltmp7:
    .p2align    4, 0x90
.LBB0_3:                                  # =>This Inner Loop
    #DEBUG_VALUE: test:i <- 0
    #DEBUG_VALUE: test:b <- $rsi
    #DEBUG_VALUE: test:a <- $rdi
    .loc    1 13 13 is_stmt 1           # example1.c:13:13
    movdqu  (%rsi,%rax), %xmm0
    movdqu  16(%rsi,%rax), %xmm1
    .loc    1 13 10 is_stmt 0           # example1.c:13:10
    movdqu  (%rdi,%rax), %xmm2
    paddb   %xmm0, %xmm2
    movdqu  16(%rdi,%rax), %xmm0
    paddb   %xmm1, %xmm0
    movdqu  32(%rdi,%rax), %xmm1
    movdqu  48(%rdi,%rax), %xmm3
    movdqu  %xmm2, (%rdi,%rax)
    movdqu  %xmm0, 16(%rdi,%rax)
    .loc    1 13 13                     # example1.c:13:13
    movdqu  32(%rsi,%rax), %xmm0
    .loc    1 13 10                     # example1.c:13:10
    paddb   %xmm1, %xmm0
    .loc    1 13 13                     # example1.c:13:13
    movdqu  48(%rsi,%rax), %xmm1
    .loc    1 13 10                     # example1.c:13:10
    paddb   %xmm3, %xmm1
    movdqu  %xmm0, 32(%rdi,%rax)
    movdqu  %xmm1, 48(%rdi,%rax)
.Ltmp8:
    .loc    1 12 26 is_stmt 1           # example1.c:12:26
    addq    $64, %rax
    cmpq    $65536, %rax                # imm = 0x10000
    jne .LBB0_3
```

example1-opt.s

```
.LBB0_2:                                  # %vector.body.preheader
    #DEBUG_VALUE: test:b <- %rsi
    #DEBUG_VALUE: test:a <- %rdi
    .loc    1 0 3                       # example1.c:0:3

    movq    $-65536, %rax               # imm = 0xFFFF0000
    .p2align    4, 0x90
.LBB0_3:                                  # %vector.body
                                          # =>This Inner Loop Header: Depth=1
    #DEBUG_VALUE: test:b <- %rsi
    #DEBUG_VALUE: test:a <- %rdi
.Ltmp3:
    .loc    1 13 13 is_stmt 1           # example1.c:13:13

    movdqu  65536(%rsi,%rax), %xmm0
    movdqu  65552(%rsi,%rax), %xmm1
    .loc    1 13 10 is_stmt 0           # example1.c:13:10

    movdqu  65536(%rdi,%rax), %xmm2
    paddb   %xmm0, %xmm2
    movdqu  65552(%rdi,%rax), %xmm0
    movdqu  65568(%rdi,%rax), %xmm3
    movdqu  65584(%rdi,%rax), %xmm4
    movdqu  %xmm2, 65536(%rdi,%rax)
    paddb   %xmm1, %xmm0
    movdqu  %xmm0, 65552(%rdi,%rax)
    .loc    1 13 13                     # example1.c:13:13

    movdqu  65568(%rsi,%rax), %xmm0
    .loc    1 13 10                     # example1.c:13:10

    paddb   %xmm3, %xmm0
    .loc    1 13 13                     # example1.c:13:13

    movdqu  65584(%rsi,%rax), %xmm1
    .loc    1 13 10                     # example1.c:13:10

    movdqu  %xmm0, 65568(%rdi,%rax)
    paddb   %xmm4, %xmm1
    movdqu  %xmm1, 65584(%rdi,%rax)
.Ltmp4:
    .loc    1 12 26 is_stmt 1           # example1.c:12:26

    addq    $64, %rax
    jne .LBB0_3
```

感想2：加强对汇编代码的理解，知其所以然

example1.c

```c
#include <stdint.h>
#include <stdlib.h>
#include <math.h>

#define SIZE (1L << 16)

void test(uint8_t * a,  uint8_t * b) {
  uint64_t i;

  for (i = 0; i < SIZE; i++) {
    a[i] += b[i];
  }
}
```

```
$clang -Wall -g -std=gnu99
-O3 -Rpass=loop-vectorize
-Rpass-missed=loop-vectorize
-S -c example1.c
```

启用循环向量化选项，生成汇编代码

比较和分析不同汇编代码的差异

example1.s

```
    #DEBUG_VALUE: test:b <- $rsi
    #DEBUG_VALUE: test:a <- $rdi
    .loc    1 0 3                      # example1.c:0:3
    xorl    %eax, %eax
.Ltmp7:
    .p2align    4, 0x90
.LBB0_3:                               # =>This Inner Loop
    #DEBUG_VALUE: test:i <- 0
    #DEBUG_VALUE: test:b <- $rsi
    #DEBUG_VALUE: test:a <- $rdi
    .loc    1 13 13 is_stmt 1          # example1.c:13:13
    movdqu  (%rsi,%rax), %xmm0
    movdqu  16(%rsi,%rax), %xmm1
    .loc    1 13 10 is_stmt 0          # example1.c:13:10
    movdqu  (%rdi,%rax), %xmm2
    paddb   %xmm0, %xmm2
    movdqu  16(%rdi,%rax), %xmm0
    paddb   %xmm1, %xmm0
    movdqu  32(%rdi,%rax), %xmm1
    movdqu  48(%rdi,%rax), %xmm3
    movdqu  %xmm2, (%rdi,%rax)
    movdqu  %xmm0, 16(%rdi,%rax)
    .loc    1 13 13                    # example1.c:13:13
    movdqu  32(%rsi,%rax), %xmm0
    .loc    1 13 10                    # example1.c:13:10
    paddb   %xmm1, %xmm0
    .loc    1 13 13                    # example1.c:13:13
    movdqu  48(%rsi,%rax), %xmm1
    .loc    1 13 10                    # example1.c:13:10
    paddb   %xmm3, %xmm1
    movdqu  %xmm0, 32(%rdi,%rax)
    movdqu  %xmm1, 48(%rdi,%rax)
.Ltmp8:
    .loc    1 12 26 is_stmt 1          # example1.c:12:26
    addq    $64, %rax
    cmpq    $65536, %rax               # imm = 0x10000
    jne .LBB0_3
```

example1-opt.s

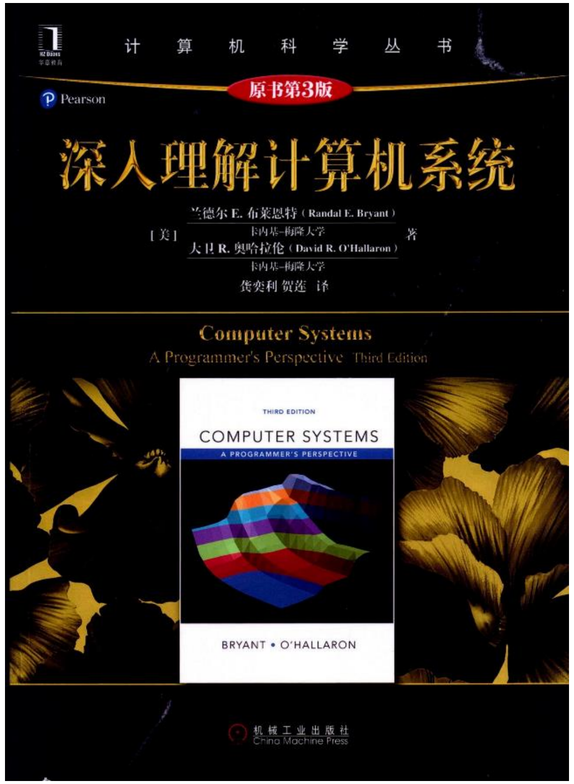
```
.LBB0_2:                               # %vector.body.preheader
    #DEBUG_VALUE: test:b <- %rsi
    #DEBUG_VALUE: test:a <- %rdi
    .loc    1 0 3                      # example1.c:0:3

    movq    $-65536, %rax              # imm = 0xFFFF0000
    .p2align    4, 0x90
.LBB0_3:                               # %vector.body
                                       # =>This Inner Loop Header: Depth=1
    #DEBUG_VALUE: test:b <- %rsi
    #DEBUG_VALUE: test:a <- %rdi
.Ltmp3:
    .loc    1 13 13 is_stmt 1          # example1.c:13:13

    movdqu  65536(%rsi,%rax), %xmm0
    movdqu  65552(%rsi,%rax), %xmm1
    .loc    1 13 10 is_stmt 0          # example1.c:13:10

    movdqu  65536(%rdi,%rax), %xmm2
    paddb   %xmm0, %xmm2
    movdqu  65552(%rdi,%rax), %xmm0
    movdqu  65568(%rdi,%rax), %xmm3
    movdqu  65584(%rdi,%rax), %xmm4
    movdqu  %xmm2, 65536(%rdi,%rax)
    paddb   %xmm1, %xmm0
    movdqu  %xmm0, 65552(%rdi,%rax)
    .loc    1 13 13                    # example1.c:13:13

    movdqu  65568(%rsi,%rax), %xmm0
    .loc    1 13 10                    # example1.c:13:10

    paddb   %xmm3, %xmm0
    .loc    1 13 13                    # example1.c:13:13

    movdqu  65584(%rsi,%rax), %xmm1
    .loc    1 13 10                    # example1.c:13:10

    movdqu  %xmm0, 65568(%rdi,%rax)
    paddb   %xmm4, %xmm1
    movdqu  %xmm1, 65584(%rdi,%rax)
.Ltmp4:
    .loc    1 12 26 is_stmt 1          # example1.c:12:26

    addq    $64, %rax
    jne .LBB0_3
```

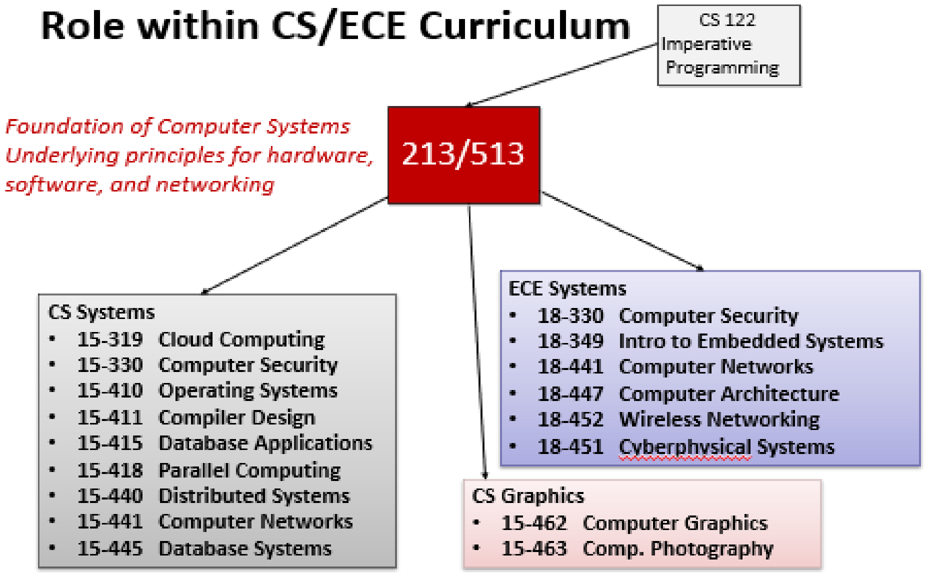
汇报提纲

- 教学I：软件系统优化
- **教学II：计算机系统**
- 实践I：跨平台的硬件性能计数器自适应分组复用 [TACO’23]
- 实践II：不同编译器优化能力的分析与识别 [CC’23]

课程简介

- 从程序员的角度来看计算机系统，贯穿整个计算机科学与工程的基础课程
- 面向计算机科学拔尖基地大一学生





[CMU 15-213 Intro to Computer Systems]

学期	必修课
大一（上）	线性代数 计算机数学分析（上） 程序设计
大一（下）	计算机数学分析（下） 数据结构 计算机系统 离散数学与算法
大一暑假	应用编程实践 设计思维
大二（上）	操作系统 计算机网络 概率与统计
大二（下）	数据管理系统 编译原理 AI 基础
大二暑假	云计算系统 软件工程实践
大四（下）	毕业设计

课程设置

Chapter	Topic	ORG	ORG+	ICS	ICS+	SP
		Course				
1	Tour of systems	•	•	•	•	•
2	Data representation	•	•	•	•	⊙ (d)
3	Machine language	•	•	•	•	•
4	Processor architecture	•	•			
5	Code optimization		•	•	•	
6	Memory hierarchy	⊙ (a)	•	•	•	⊙ (a)
7	Linking			⊙ (c)	⊙ (c)	•
8	Exceptional control flow			•	•	•
9	Virtual memory	⊙ (b)	•	•	•	•
10	System-level I/O				•	•
11	Network programming				•	•
12	Concurrent programming				•	•

Figure 2 Five systems courses based on the CS:APP book. ICS+ is the 15-213 course from Carnegie Mellon. Notes: The ⊙ symbol denotes partial coverage of a chapter, as follows: (a) hardware only; (b) no dynamic storage allocation; (c) no dynamic linking; (d) no floating point.

[Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, 3rd edition (CS:APP3e), Pearson, 2016]

周	理论课主题	教材章节	实践课		Labs
1	概述	1	Bootcamp 1: Linux, Command Line, Git	shell、脚本、vim、命令行、调试	
2	信息的表示和处理	2	Bootcamp 2: Debugging & GDB Bootcamp 3: GCC & Build Automation	git，性能分析，数据整理	
3			L1布置（3月17日）	元编程	L1
4					
5	程序的机器级表示	3	L1提交，L2布置（3月31）	安全和密码学	L2
6					
7			L2提交（4月14日）		
8	指令集架构	4.1	L1&L2 QA, L3布置（4月21日）		L3
9	存储器层次结构	6			
10	程序性能优化	5	L3提交（5月5日）		
11			L3 QA，L4布置（5月12日）		L4
12	虚拟内存	9			
13			L4提交（5月26日）		
14	链接	7	L4 QA，L5布置（6月2日）		L5
15					
16	异常控制流	8	L5提交（6月16日）		
17			L5 QA（6月23日）		
18	期末考试				

感想3：不纯粹依赖书本，结合系统本身去教和学

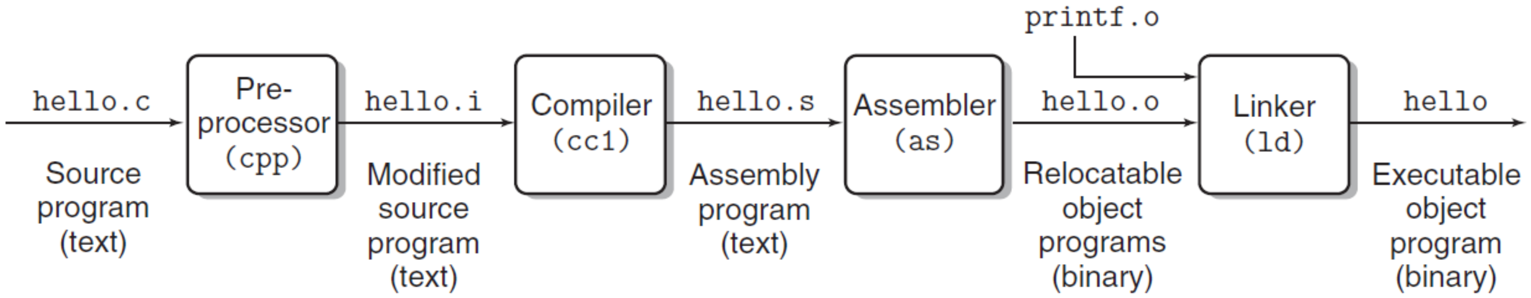


Figure 1.3 **The compilation system.**

[Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, 3rd edition (CS:APP3e), Pearson, 2016]

感想3：不纯粹依赖书本，结合系统本身去教和学

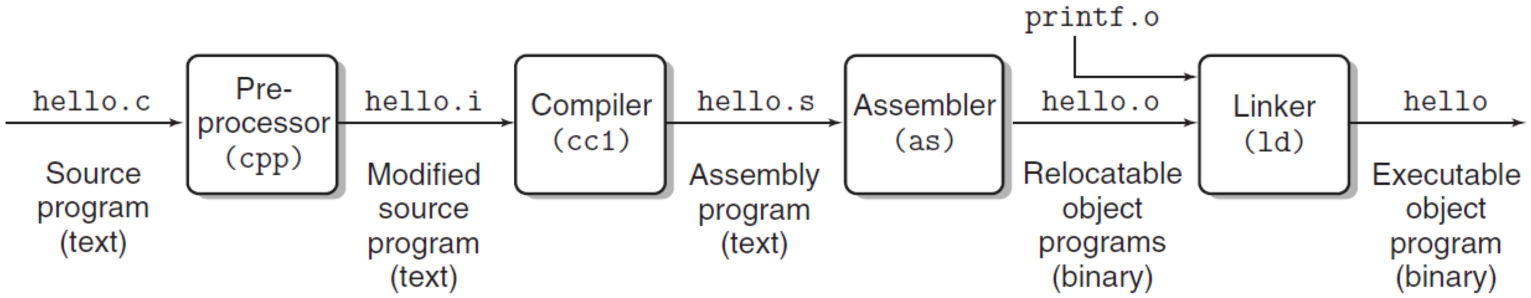
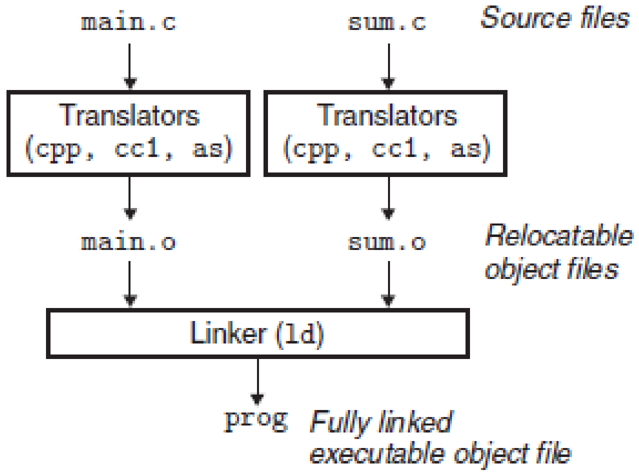


Figure 1.3 The compilation system.

Figure 7.2 **Static linking.** The linker combines relocatable object files to form an executable object file prog.




```
linux> gcc -Og -o prog main.c sum.c
cpp [other arguments] main.c /tmp/main.i
cc1 /tmp/main.i -Og [other arguments] -o /tmp/main.s
as [other arguments] -o /tmp/main.o /tmp/main.s
ld -o prog [system object files and args] /tmp/main.o /tmp/sum.o
```

[Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, 3rd edition (CS:APP3e), Pearson, 2016]

感想3：不纯粹依赖书本，结合系统本身去教和学

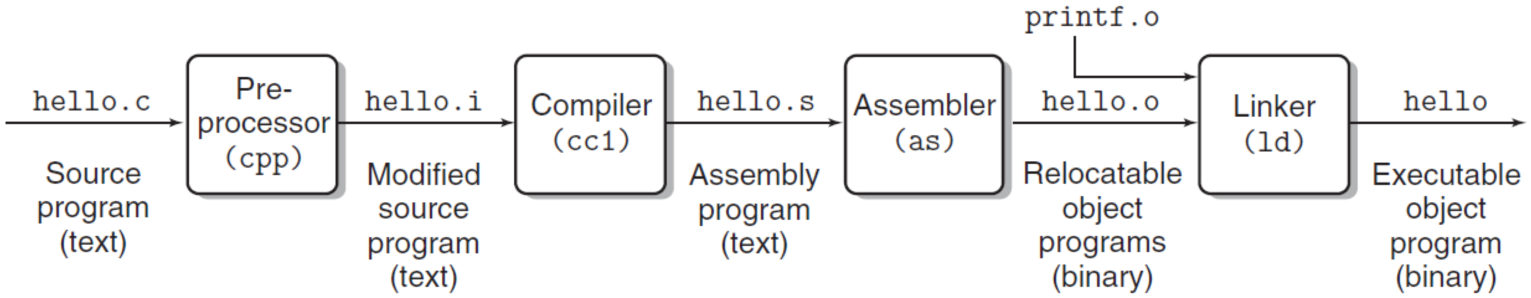
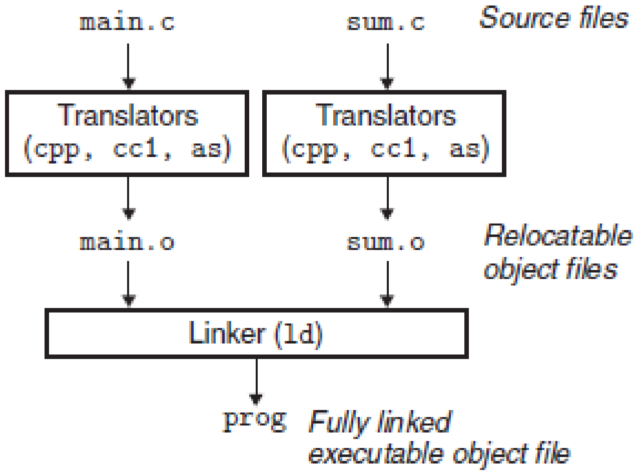


Figure 1.3 **The compilation system.**

Figure 7.2 **Static linking.** The linker combines relocatable object files to form an executable object file prog.




```
linux> gcc -Og -o prog main.c sum.c

cpp [other arguments] main.c /tmp/main.i

cc1 /tmp/main.i -Og [other arguments] -o /tmp/main.s

as [other arguments] -o /tmp/main.o /tmp/main.s

ld -o prog [system object files and args] /tmp/main.o /tmp/sum.o
```

System object files and other arguments?

[Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, 3rd edition (CS:APP3e), Pearson, 2016]

感想3：不纯粹依赖书本，结合系统本身去教和学

$gcc -Og -o prog main.c swap.c -v &> linking.log

configure

cc1 (main)

```
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/7/lto-wrapper
OFFLOAD_TARGET_NAMES=nvptx-none
OFFLOAD_TARGET_DEFAULT=1
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 7.5.0-3ubuntu1~18.04' --with-bugurl=file:///usr/share/doc/gcc-7/README.Bugs --enable-languages=c,ada,c++,go,brig,d,fortran,objc,obj-c++ --prefix=/usr --with-gcc-major-version-only --program-suffix=-7 --program-prefix=x86_64-linux-gnu- --enable-shared --enable-linker-build-id --libexecdir=/usr/lib --without-included-gettext --enable-threads=posix --libdir=/usr/lib --enable-nls --enable-bootstrap --enable-clocale=gnu --enable-libstdcxx-debug --enable-libstdcxx-time=yes --with-default-libstdcxx-abi=new --enable-gnu-unique-object --disable-vtable-verify --enable-libmpx --enable-plugin --enable-default-pie --with-system-zlib --with-target-system-zlib --enable-objc-gc=auto --enable-multiarch --disable-werror --with-arch-32=i686 --with-abi=m64 --with-multilib-list=m32,m64,mx32 --enable-multilib --with-tune=generic --enable-offload-targets=nvptx-none --without-cuda-driver --enable-checking=release --build=x86_64-linux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu
Thread model: posix
gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)
COLLECT_GCC_OPTIONS='-Og' '-o' 'prog' '-v' '-mt... generic' '-march=x86-64'
 /usr/lib/gcc/x86_64-linux-gnu/7/cc1 -quiet -v -imultiarch x86_64-linux-gnu main.c -quiet -dumpbase main.c -mtune=generic -march=x86-64 -auxbase main -Og -version -fstack-protector-strong -Wformat -Wformat-security -o /tmp/ccMkAdhk.s
GNU C11 (Ubuntu 7.5.0-3ubuntu1~18.04) version 7.5.0 (x86_64-linux-gnu)
        compiled by GNU C version 7.5.0, GMP version 6.1.2, MPFR version 4.0.1, MPC version 1.1.0, isl version isl-0.19-GMP

GGC heuristics: --param ggc-min-expand=100 --param ggc-min-heapsize=131072
ignoring nonexistent directory "/usr/local/include/x86_64-linux-gnu"
ignoring nonexistent directory "/usr/lib/gcc/x86_64-linux-gnu/7/../../../../x86_64-linux-gnu/include"
#include "..." search starts here:
#include <...> search starts here:
 /usr/lib/gcc/x86_64-linux-gnu/7/include
 /usr/local/include
 /usr/lib/gcc/x86_64-linux-gnu/7/include-fixed
 /usr/include/x86_64-linux-gnu
 /usr/include
End of search list.
GNU C11 (Ubuntu 7.5.0-3ubuntu1~18.04) version 7.5.0 (x86_64-linux-gnu)
        compiled by GNU C version 7.5.0, GMP version 6.1.2, MPFR version 4.0.1, MPC version 1.1.0, isl version isl-0.19-GMP

```

cpp? ld?

as (main)

cc1 (sum)

as (sum)

collect2

```
GGC heuristics: --param ggc-min-expand=100 --param ggc-min-heapsize=131072
Compiler executable checksum: b62ed4a2880cd4159476ea8293b72fa8
COLLECT_GCC_OPTIONS='-Og' '-o' 'prog' '-v' '-mtune=generic' '-march...
 as -v --64 -o /tmp/cc2ZNupk.o /tmp/ccMkAdhk.s
GNU assembler version 2.30 (x86_64-linux-gnu) using BFD version (GNU Binutils for Ubuntu) 2.30
COLLECT_GCC_OPTIONS='-Og' '-o' 'prog' '-v' '-mtune=generic' '-march...
 /usr/lib/gcc/x86_64-linux-gnu/7/cc1 -quiet -v -imultiarch x86_64-linux-gnu sum.c -quiet -dumpbase sum.c -mtune=generic -march=x86-64 -auxbase sum -Og -version -fstack-protector-strong -Wformat -Wformat-security -o /tmp/ccMkAdhk.s
GNU C11 (Ubuntu 7.5.0-3ubuntu1~18.04) version 7.5.0 (x86_64-linux-gnu)
        compiled by GNU C version 7.5.0, GMP version 6.1.2, MPFR version 4.0.1, MPC version 1.1.0, isl version isl-0.19-GMP

GGC heuristics: --param ggc-min-expand=100 --param ggc-min-heapsize=131072
ignoring nonexistent directory "/usr/local/include/x86_64-linux-gnu"
ignoring nonexistent directory "/usr/lib/gcc/x86_64-linux-gnu/7/../../../../x86_64-linux-gnu/include"
#include "..." search starts here:
#include <...> search starts here:
 /usr/lib/gcc/x86_64-linux-gnu/7/include
 /usr/local/include
 /usr/lib/gcc/x86_64-linux-gnu/7/include-fixed
 /usr/include/x86_64-linux-gnu
 /usr/include
End of search list.
GNU C11 (Ubuntu 7.5.0-3ubuntu1~18.04) version 7.5.0 (x86_64-linux-gnu)
        compiled by GNU C version 7.5.0, GMP version 6.1.2, MPFR version 4.0.1, MPC version 1.1.0, isl version isl-0.19-GMP

GGC heuristics: --param ggc-min-expand=100 --param ggc-min-heapsize=131072
Compiler executable checksum: b62ed4a2880cd4159476ea8293b72fa8
COLLECT_GCC_OPTIONS='-Og' '-o' 'prog' '-v' '-mtune=generic' '-march=x86-64'
 as -v --64 -o /tmp/ccq2qxFk.o /tmp/ccMkAdhk.s
GNU assembler version 2.30 (x86_64-linux-gnu) using BFD version (GNU Binutils for Ubuntu) 2.30
COMPILER_PATH=/usr/lib/gcc/x86_64-linux-gnu/7/:/usr/lib/gcc/x86_64-linux-gnu/7/:/usr/lib/gcc/x86_64-linux-gnu/:/usr/lib/gcc/x86_64-linux-gnu/7/:/usr/lib/gcc/x86_64-linux-gnu/
LIBRARY_PATH=/usr/lib/gcc/x86_64-linux-gnu/7/:/usr/lib/gcc/x86_64-linux-gnu/7/../../../x86_64-linux-gnu/:/usr/lib/gcc/x86_64-linux-gnu/7/../../../../lib/:/lib/x86_64-linux-gnu/:/lib/../lib/:/usr/lib/x86_64-linux-gnu/:/usr/lib/.. ...u/7/../../../:/lib/:/usr/lib/
COLLECT_GCC_OPTIONS='-Og' '-o' 'prog' '-v' '-mtune=generic' ...
 /usr/lib/gcc/x86_64-linux-gnu/7/collect2 -plugin /usr/lib/gcc/x86_64-linux-gnu/7/liblto_plugin.so -plugin-opt=/usr/lib/gcc/x86_64-linux-gnu/7/lto-wrapper -plugin-opt=-fresolution=/tmp/ccY5rkYk.res -plugin-opt=-pass-through=-lgcc -plugin-opt=-pass-through=-lgcc_s -plugin-opt=-pass-through=-lc -plugin-opt=-pass-through=-lgcc -plugin-opt=-pass-through=-lgcc_s --build-id --eh-frame-hdr -m elf_x86_64 --hash-style=gnu --as-needed -dynamic-linker /lib64/ld-linux-x86-64.so.2 -pie -z now -z relro -o prog /usr/lib/gcc/x86_64-linux-gnu/7/../../../x86_64-linux-gnu/Scrt1.o /usr/lib/gcc/x86_64-linux-gnu/7/../../../x86_64-linux-gnu/crti.o /usr/lib/gcc/x86_64-linux-gnu/7/crtbeginS.o -L/usr/lib/gcc/x86_64-linux-gnu/7 -L/usr/lib/gcc/x86_64-linux-gnu/7/../../../x86_64-linux-gnu -L/usr/lib/gcc/x86_64-linux-gnu/7/../../../../lib -L/lib/x86_64-linux-gnu -L/lib/../lib -L/usr/lib/x86_64-linux-gnu -L/usr/lib/../lib -L/usr/lib/gcc/x86_64-linux-gnu/7/../../.. /tmp/cc2ZNupk.o /tmp/ccq2qxFk.o -lgcc --push-state --as-needed -lgcc_s --pop-state -lc -lgcc --push-state --as-needed -lgcc_s --pop-state /usr/lib/gcc/x86_64-linux-gnu/7/crtendS.o /usr/lib/gcc/x86_64-linux-gnu/7/../../../x86_64-linux-gnu/crtn.o
COLLECT_GCC_OPTIONS='-Og' '-o' 'prog' '-v' '-mtune=generic' '-march=x86-64'
```

汇报提纲

- 教学I：软件系统优化
- 教学II：计算机系统
- **实践I：跨平台的硬件性能计数器自适应分组复用 [TACO’23]**
- 实践II：不同编译器优化能力的分析与识别 [CC’23]

RESEARCH-ARTICLE FREE ACCESS

Efficient Cross-platform Multiplexing of Hardware Performance Counters via Adaptive Grouping

Just Accepted

Authors: Tong-yu Liu, Jianmei Guo, Bo Huang Authors Info & Claims

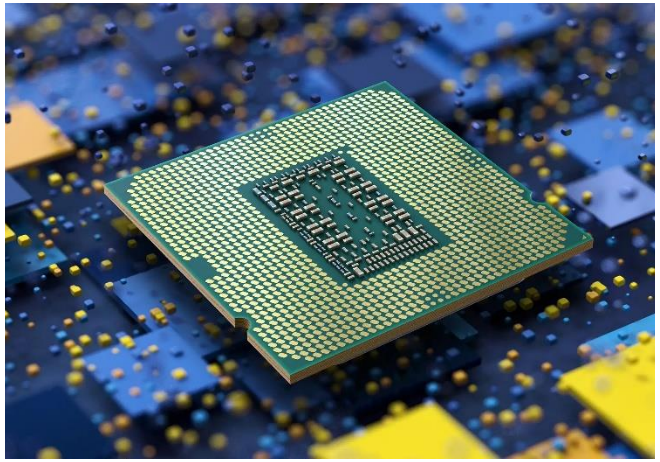
ACM Transactions on Architecture and Code Optimization • Accepted on September 2023 • https://doi.org/10.1145/3629525

开源项目：https://jihulab.com/solecnu/hperf

问题

Software performance

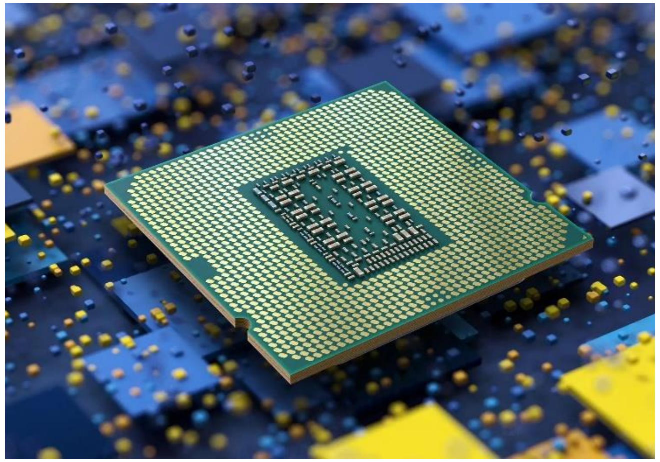
Hardware performance



问题

Software performance

Performance metrics



Hardware performance

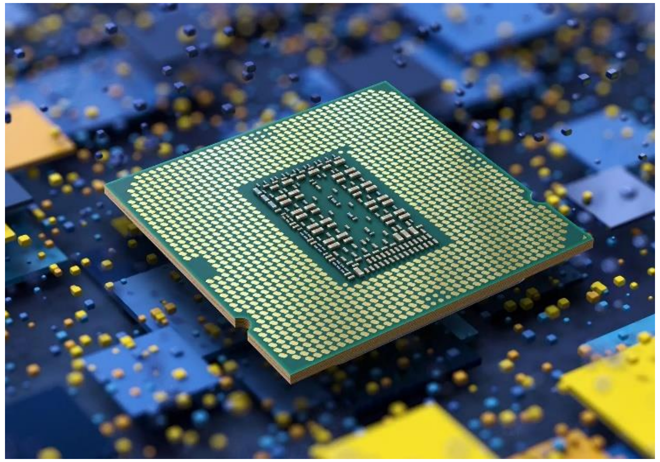
Hardware Performance Counters

问题

Software performance

Many (e.g., hundreds)
Performance metrics

Hardware performance



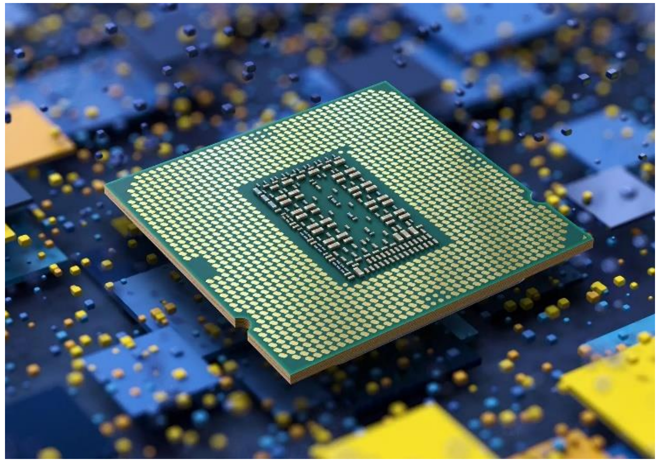
A few (e.g., 7 on Intel CLX)
Hardware Performance Counters

问题

Software performance

Many (e.g., hundreds)
Performance metrics

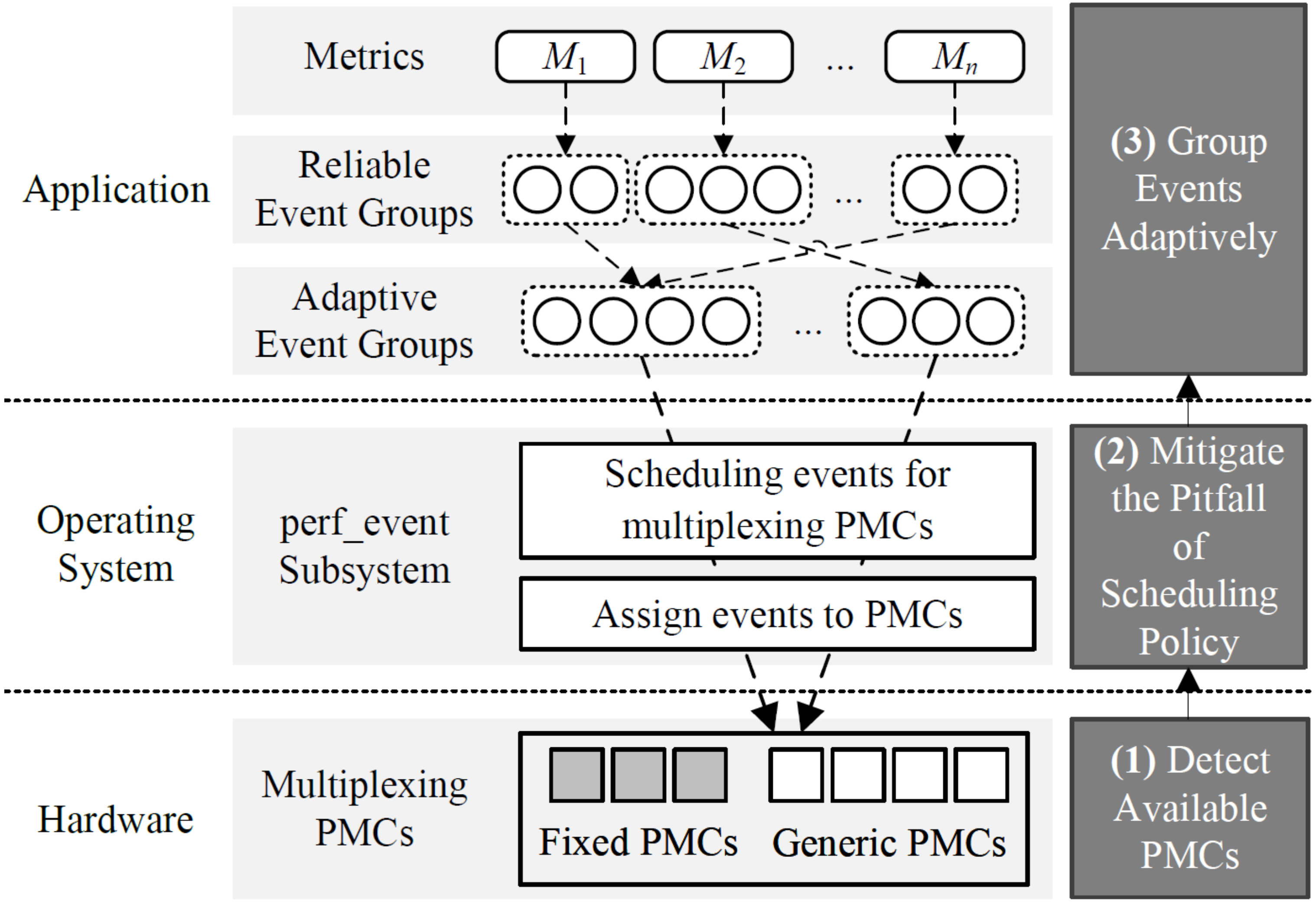
Hardware performance



A few (e.g., 7 on Intel CLX)
Hardware Performance Counters
on different platforms

intel AMD AMPERE Kunpeng

方法



Key concerns:

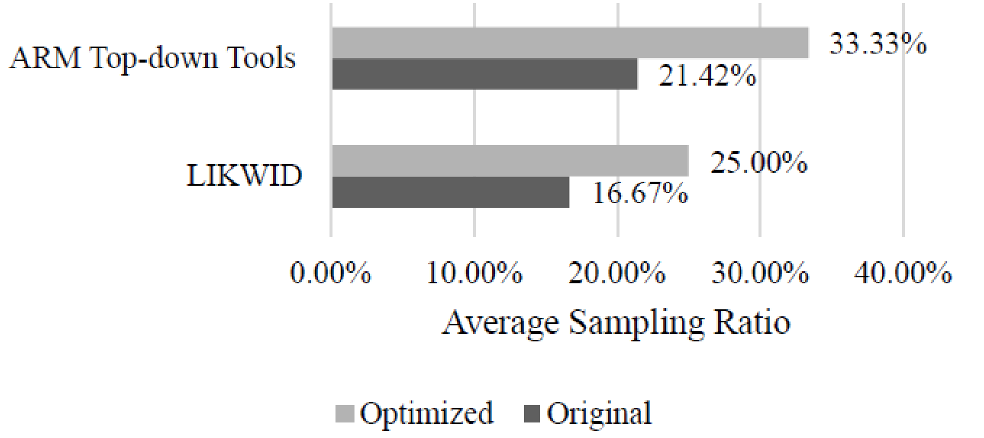
- Inefficient grouping
- Event scheduling pitfall in Linux perf_event
- Unawareness of available counters

结果

Table 7. Results of the Detection of Available PMCs for Mainstream Processors [9, 29]

Machine	ISA	Processor	From detection # Fixed PMCs	From detection # Generic PMCs	From documentation # Fixed PMCs	From documentation # Generic PMCs
A	x86-64[1]	Intel Xeon Gold 5218R (CascadeLake)	3	4	3	4
B	x86-64[1]	Intel Xeon Platinum 8352Y (IceLake)	4	8	4	8
C	AArch64	Hisilicon Kunpeng 920	1	**12**	1	**6**
D	AArch64	Ampere Altra	1	6	1	6

[1] NMI watchdogs are disabled.



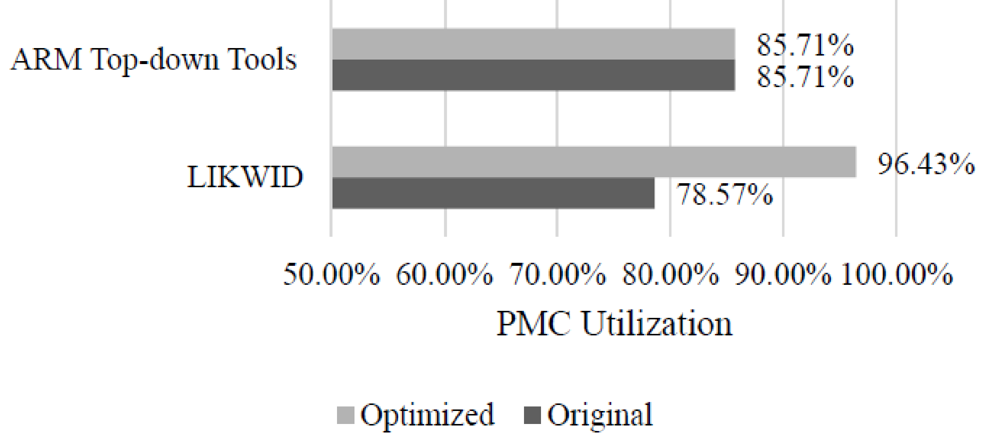


Fig. 9. Comparison of efficiency between before and after optimization.

汇报提纲

- 教学I：软件系统优化
- 教学II：计算机系统
- 实践I：跨平台的硬件性能计数器自适应分组复用 [TACO’23]
- **实践II：不同编译器优化能力的分析与识别 [CC’23]**

ACM SIGPLAN 2023 International Conference on Compiler Construction

February 25-26, 2023, Montréal, Québec, Canada.

A Hotspot-Driven Semi-automated Competitive Analysis Framework for Identifying Compiler Key Optimizations

Wenlong Mu East China Normal University, Yilei Zhang East China Normal University, Bo Huang East China Normal University, Jianmei Guo East China Normal University, Shiqiang Cui Hangzhou Hongjun Microelectronics Technology

开源项目： https://jihulab.com/solecnu/hotspotdrivenframework

问题

Open source compilers

GCC

LLVM COMPILER INFRASTRUCTURE

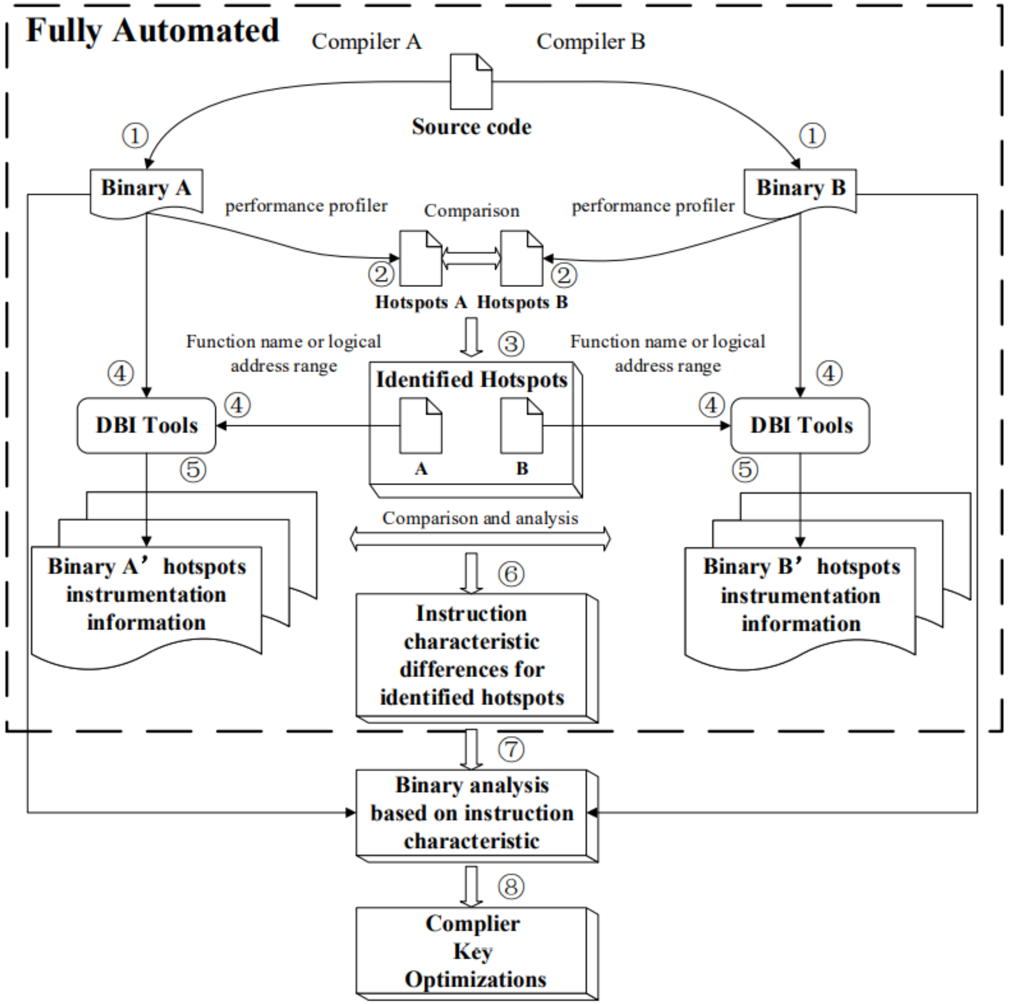
Proprietary compilers

华为 毕昇编译器

Intel® oneAPI DPC++/C++ Compiler
A Standards-Based, Cross-architecture Compiler

AMD Optimizing C/C++ and Fortran Compilers (AOCC)

How to "learn" from other compilers?



Major steps(automation part):

- Hotspot detection: generate the top N hotspots with *perf* for each compiler-generated binary, reformatting the hotspot representation as required by our *DynamoRIO* tools
- Hotspot selection: select the "identified hotspots" according to the predefined rules respectively for the two hotspot lists
- Instrumentation only for "identified hotspots": build *DynamoRIO* tools so that only "identified hotspots" are instrumented, and the instrumentation for each identified hotspot can be done in parallel
- Instruction characteristic comparison: compare the instruction characteristic difference for associated "identified hotspots" respectively in two compiler-generated binaries

结果 (gcc vs. icc/bisheng)

Top 10 hotspot lists of 648.exchange2_s using BiSheng and GCC

Rank	BiSheng			GCC		
	relative time(%)	absolute time(sec)	function name	relative time(%)	absolute time(sec)	function name
1	**57.32**	**14.33**	***digits_2.7***	**82.38**	**32.05**	***MOD_digits_2***
2	**19.38**	**4.85**	***digits_2.4***	**7.30**	**2.84**	***gfortran_mminloc***
3	**18.30**	**4.58**	***logic_new_solver***	**4.15**	**1.61**	***specific.4***
4	1.46	0.37	*free*	**2.31**	**0.90**	***logic_MOD_new_solver***
5	0.80	0.20	*malloc*	1.08	0.42	*hidden_triplets.0*
6	0.43	0.11	*covered*	0.78	0.30	*free*
7	0.32	0.08	*brute*	0.55	0.21	*naked_triplets.1*
8	0.17	0.04	*f90_dealloc*	0.48	0.19	*hidden_pairs.2*
9	0.15	0.04	*f90_alloc*	0.25	0.10	*MOD_brute*
10	0.14	0.04	*f90_set_intrin*	0.25	0.10	*malloc*

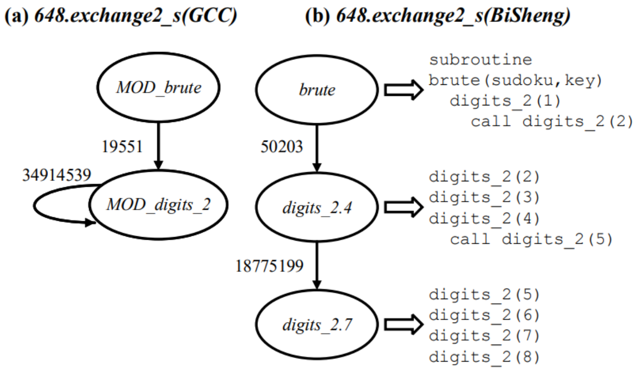
Optimization results on a GCC-based production compiler

Benchmark	before		after		improvement
	runtime(sec)	score	runtime(sec)	score	(score)
548.exchange2_r	527	636	306	1100	72.96%
505.mcf_r	1350	153	776	234	52.94%
525.x264_r	344	651	271	828	27.19%

Setup: Huawei Kunpeng 920 7261K (2 sockets, 128 cores in total); Operating System: CentOS Linux release 7.9.2009; SPEC2017 input data size: reference.

Instruction characterization

Category	*digit_2.4*	*digit_2.7*	*MOD_digit_2*
Branch	**3593142992**	**11609217450**	**29321533296**
Indirect	**50203**	**18775199**	**34934090**
Jump	0	0	0
Call	0	0	0
Ret	50203	18775199	34934090
Cond	**3361685141**	**10733730071**	**27698651020**
Uncond Direct	231407648	856712180	1587948186
Operation	**15199221930**	**41803072569**	**116664503804**
Binary Arith	14686454611	39759170962	102813533806
Logical	498710685	1733736947	12559811188
Shift	14056634	310164660	1291158810
Data Transfer	32692266	214161627	1636029315
Load/Store	**11911031090**	**37507748969**	**81757176763**
Others	534241804	1514087575	2581871188
Vectorization	**5426852903**	**10429222578**	**268662027**



Function inlining

Optimization guidance for GCC (AArch64)

Benchmark	Optimization suggestions for GCC
605.mcf_s	(1) Optimize the capacity of inlining callback functions to eliminate indirect subroutine calls. (2) Optimize memory layout of the data structure, including structure peeling and unused field elimination.
648.exchange2_s	(1) Improve the optimization of inlining the non-tail recursive function calls. (2) Implement function specialization. (3) Optimize the capacity of automatic vectorization. (4) Inline the built-in functions for Fortran code.
625.x264_s	(1) Optimize the capacity of automatic vectorization. (2) Optimize the decisions of whether to inline a function.

汇报总结

- 教学I：软件系统优化
- 教学II：计算机系统
- 实践I：跨平台的硬件性能计数器自适应分组复用 [TACO' 23]
- 实践II：不同编译器优化能力的分析与识别 [CC' 23]

➢加强动手能力培养，开展即时反馈的实践课
➢加强对汇编代码的理解，知其所以然
➢不单纯依赖书本，结合系统本身去教和学
➢结合产业需求，打破性能瓶颈，做实在的研究

感谢观看