第05章：优化程序性能

我们可以做的更好

视频解说

导读

程序优化涉及的范围实在是太广了，几乎每个层面都可以进行优化，比如撰写<编译器友好型>以及<缓存友好型>的程序，针对不同的目标硬件平台还可能进行特定的优化，等等，优化的难点在于你需要对系统有充分理解，当然了在你做优化之前**首先要保证原始程序功能正确（并且有回归测试）**，否则一切都是徒劳。

> 首先需要理解，哪些因素会影响程序的性能

$$\text{Time} = \text{Seconds/Program} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

Hardware or software component	Affects what?	How?
Algorithm	Instruction count, possibly CPI	The algorithm determines the number of source program instructions executed and hence the number of processor instructions executed. The algorithm may also affect the CPI, by favoring slower or faster instructions. For example, if the algorithm uses more divides, it will tend to have a higher CPI.
Programming language	Instruction count, CPI	The programming language certainly affects the instruction count, since statements in the language are translated to processor instructions, which determine instruction count. The language may also affect the CPI because of its features; for example, a language with heavy support for data abstraction (e.g., Java) will require indirect calls, which will use higher CPI instructions.
Compiler	Instruction count, CPI	The efficiency of the compiler affects both the instruction count and average cycles per instruction, since the compiler determines the translation of the source language instructions into computer instructions. The compiler's role can be very complex and affect the CPI in complex ways.
Instruction set architecture	Instruction count, clock rate, CPI	The instruction set architecture affects all three aspects of CPU performance, since it affects the instructions needed for a function, the cost in cycles of each instruction, and the overall clock rate of the processor.

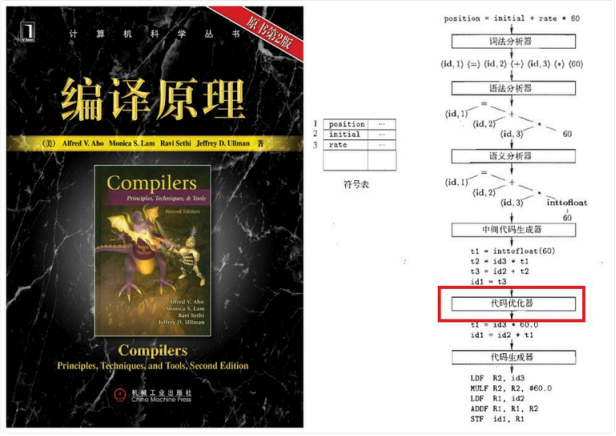
温馨提醒：Nothing can fix a dumb algorithm! == 没有什么能修正一个愚蠢的算法！

历史：第一个编译器是由美国女性计算机科学家葛丽丝·霍普（Grace Hopper）于1952年为A-0 系统编写的。但是1957年由任职于IBM的美国计算机科学家约翰·巴科斯（John Warner Backus）领导的FORTRAN则是第一个被实现出具备完整功能的编译器。1960年，COBOL成为一种较早的能在多种架构下被编译的语言。首个能编译自己源程序的LISP编译器是在1962年由麻省理工学院的Hart和Levin制作的。

题外话：1947年9月9日，葛丽丝·霍普（Grace Hopper）还发现了第一个电脑上的bug。当在Mark II计算机上工作时，整个团队都搞不清楚为什么电脑不能正常运作了。经过大家的深度挖掘，发现原来是一只飞蛾意外飞入了一台电脑内部而引起的故障。这个团队把错误解除了，并在日记本中记录下了这一事件。也因此，人们逐渐开始用"Bug"（原意为"虫子"）来称呼计算机中隐藏的错误。

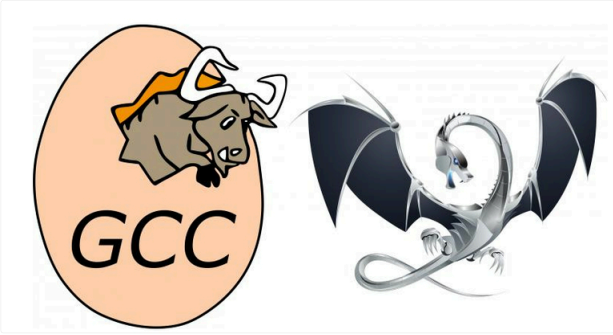
编译器领域最早成名的教材：*Compilers: Principles,Techniques,and Tools*，中文名（龙书）：*编译原理*

本书的一大特点就是抽象难懂，主要讨论了编译器设计的重要主题，包括词法分析、语法分析、语义分析、中间代码生成、代码优化、代码生成等过程。

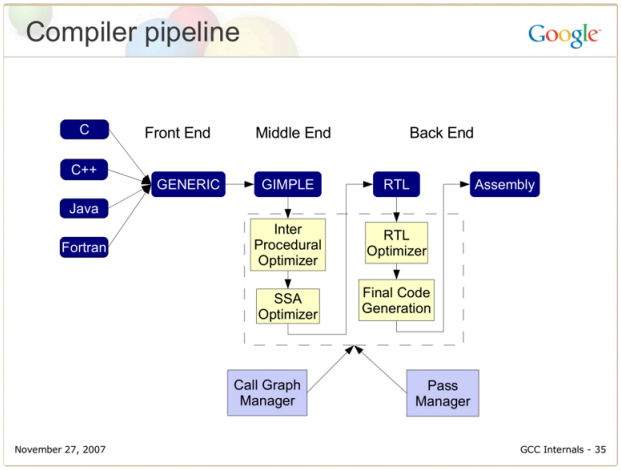


2020年图灵奖宣布授予哥伦比亚大学计算机科学名誉教授 Alfred Vaino Aho 和斯坦福大学计算机科学名誉教授 Jeffrey David Ullman，以表彰他们在编程语言实现（programming language implementation）领域基础算法和理论方面的成就。

从教科书回到现实世界（工业强度的现代编译器）：GCC，LLVM



例如，GCC实现的时候做了两层的中间码，分别是GIMPLE中间码，RTL中间码。代码生成阶段要考虑对应的指令集架构的寄存器使用，以及考虑流水线的调度。



阿里巴巴-系统软件事业部编译器团队招聘（也招实习生）

发布时间： 2019-04-09 发布人： 沈宝庆 浏览次数： 201

团队介绍：
系统软件事业部编译器团队，旨在通过编译器、c库层面的优化工作大幅提升阿里巴巴软件的运行效能，降低机器投入成本。作为系统软件事业部的一份子，团队负有为集团打造稳定、高效的编译器、c库等基础软件的责任。我们相信在编译、c库等基础软件上的深耕细作可以为阿里带来明显的效益并愿为此做持久投入。

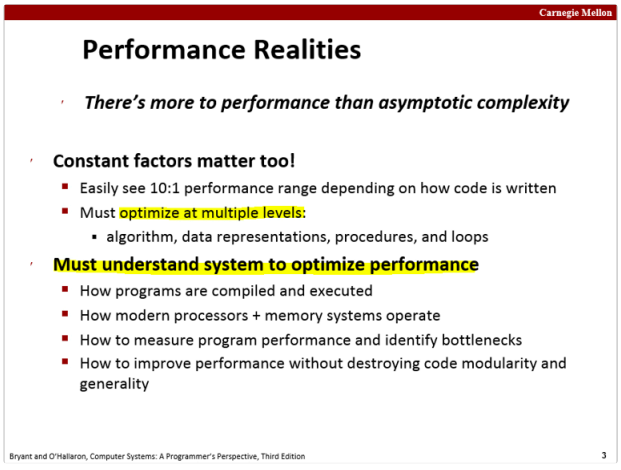
职位描述：
1. 面向复杂集群、分布式线上环境，分析系统和应用性能瓶颈，寻找优化机会。
2. 深入编译器middle-end、back-end优化工作，提升系统和重点应用运行效率。
3. 深入基础软件库（如glibc）维护、优化工作，提升系统运行效率。
4. 通过参与开源社区开发、学术会议等方式建设阿里在基础软件领域的品牌。

岗位要求（无须全部满足）：
1. 对开源编译器、c库等基础软件项目代码有强烈兴趣。
2. 熟悉某编译器middle-end、back-end开发及相关优化技术如ipa、lto、pgo、inline、循环优化等。
3. 对性能分析、编译器或c库优化有深入理解及相关工作经验，通过优化大幅提升过系统、应用性能。
4. 有丰富的gcc、llvm或glibc开发经验，有相关项目社区开发参与记录者优先。

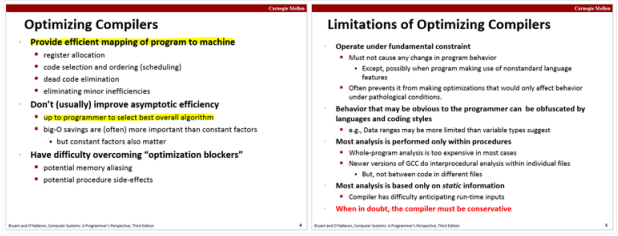
工作地点北京、杭州
有兴趣的朋友，简历发送至liangbin.mj@alibaba-inc.com

学习方式

CMU教授的视频教程 - Lecture10：程序性能优化 (Program Optimization)



系统性能优化的诸多着眼点



编译器优化思路以及它的局限性

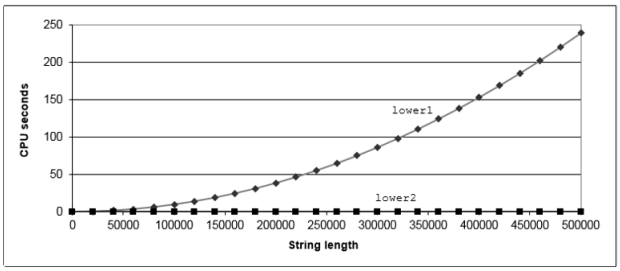
重点示例

例1：编译器聪明但保守，有时候你写的程序会阻止编译器做优化（precedure side-effects）

```
void lower(char *s)
{
  size_t i;
  for (i = 0; i < strlen(s); i++)
    if (s[i] >= 'A' && s[i] <= 'Z')
      s[i] -= ('A' - 'a');
}
```

会这样优化么？

```
void lower(char *s)
{
  size_t i;
  size_t len = strlen(s);
  for (i = 0; i < len; i++)
    if (s[i] >= 'A' && s[i] <= 'Z')
      s[i] -= ('A' - 'a');
}
```



例2：编译器聪明但保守，有时候你写的程序会阻止编译器做优化（memory alising）

```
int fn (int *a, int *b)
{
    *a = 3;
    *b = 4;
    return (*a + 5);
}

// 编译器会把以上代码优化成下面的样子么？不会！谁知道程序员会不会这么调用 f(&x,&x);
int fn (int *a, int *b)
{
    *a = 3;
    *b = 4;
    return (3 + 5);
}

// 但是你可以帮助编译器，使用C99的restrict类型限定符，但还是需要开发者确保两个指针不指向同一数据
// https://gcc.gnu.org/onlinedocs/gcc/Restricted-Pointers.html
int fn (int *__restrict__ a, int *__restrict__ b)
{
    *a = 3;
    *b = 4;
    return (*a + 5); // 这里会被优化为 return (3 + 5)
}
```

例3：善用编译器的特定选项或者开关（当然也需要处理器支持），利用有限的资源创造出最大的价值。

> ⓘ 请参考教学投影片（p22~）以及CMU教授的视频讲解，与原始程序相比，性能达到了数十倍（甚至百倍）的提升，其中使用到了许多优化方面的技巧，比如 code motion，loop unrolling，auto vectorization，etc，相信会对你有所启发。

作业解答

（无）

实验解读

（无）

Previous
第04章：处理器体系结构

Next
第06章：存储器层次结构