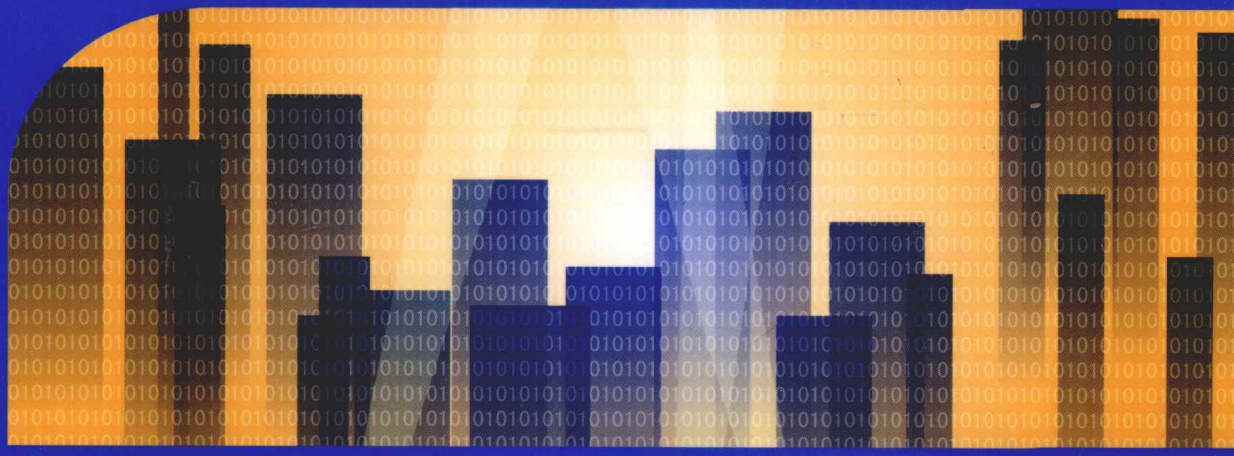
英特尔® 软件学院系列课程



英特尔® 平台编程

Pogramming on Intel® Platform

英特尔® 亚太研发有限公司
英特尔® 软件学院教材编写组

上海交通大學出版社
SHANGHAI JIAO TONG UNIVERSITY PRESS

内 容 提 要

本书从程序员的角度介绍了软件优化过程中如何使用 Intel 公司的高性能 C++编译器和性能分析器 VTune 等工具，寻找性能瓶颈和优化代码，以利用编译器的强大能力和 Intel 处理器等硬件的性能，从而编写出高质量和高性能的应用代码。

本书可作为高等学校学生在学习 C 语言编程之后的软件优化进阶教材，也可供广大 C 和 C++程序员参考。

图书在版编目(CIP)数据

英特尔平台编程 / 英特尔®软件学院教材编写组编.
—上海：上海交通大学出版社，2011
英特尔软件学院系列课程培训教材
ISBN 978-7-313-06868-2

Ⅰ.①英… Ⅱ.①英… Ⅲ.①程序设计—技术培训—教材 Ⅳ.①TP311.1

中国版本图书馆 CIP 数据核字(2010)第 197740 号

英特尔平台编程

英特尔®软件学院教材编写组 编

上海交通大学出版社出版发行

(上海市番禺路 951 号 邮政编码 200030)

电话：64071208 出版人：韩建民

上海崇明南海印刷厂印刷 全国新华书店经销

开本：787 mm×960 mm 1/16 印张：14.5 字数：265 千字

2011 年 1 月第 1 版 2011 年 1 月第 1 次印刷

印数：1～5 030

ISBN 978-7-313-06868-2/TP 定价：49.50 元

序

进入21世纪，信息技术和信息产业在全球范围内迅猛发展的势头更为强劲，如何尽快适应新技术和新应用带来的挑战，及时更新员工知识结构，并动态调整企业人才培养战略，已经成为广大科技公司迫切需要解决的问题之一。对于高等教育、职业教育等专业组织机构来说，则面临着紧跟企业前进步伐，准确接轨社会发展趋势，瞄准世界科技前沿水平，不断进行教育教学创新，提高学生实践能力，开拓学生知识视野的现实需求。在中国，尽管近年来已经在科技人才培养方面取得了长足的进步，但是就整体现状而言，尤其在知识更新和技术创新方面，距离完全满足社会的需求还存在着较大的发展空间。

英特尔公司历来关注技术的发展创新和科技人才的培养。英特尔®软件学院隶属于英特尔软件与服务事业部，作为英特尔公司专业的对外培训机构，为全球的软件开发人员提供了丰富的前沿技术培训课程。多年来，英特尔®软件学院一直致力于培训软件开发人员，与中国的软件开发人员共同发展，帮助其掌握和应用英特尔的最新技术及经验，提高软件开发技术水平，提升产品开发技能。目前，英特尔®软件学院在中国已经发展成为面向软件开发、项目管理及商业运营方向的优秀一站式培训服务基地。依托英特尔公司强大的师资力量，沿袭英特尔用户需求至上的传统，英特尔®软件学院已经与国内多家知名公司、大学和教育机构建立了长期

稳定的合作关系，迄今已有数万名工程师和大学教师参与了英特尔® 软件学院的技术培训，并学以致用。

《英特尔® 软件学院系列课程》由英特尔® 软件学院牵头，联合国内的顶级高等学府合作编写。相信这本教材作为英特尔® 软件学院的重点课程之一，在科技人才培养和知识创新方面必将发挥重要的作用。

英特尔亚太研发有限公司总经理

英特尔公司软件与服务事业部中国区总经理

梁兆柱博士

前言

随着计算机体系结构和硬件技术的发展，硬件的价格越来越便宜，与此同时，其性能也有了飞速的提高，但是这并不意味着在编写代码时无须考虑优化，许多应用，特别是科学计算、多媒体处理等，需要从算法和数据结构的设计、代码的编写和性能瓶颈的解决等方面来提高应用的性能。大多数有关程序语言编程的书籍主要描述该语言支持的数据类型以及控制结构等语言特征，而有关编译的书籍则更多地侧重于介绍编译器的基本原理以及编译器的实现，本书从程序员的角度来介绍在软件优化过程中如何使用 Intel 公司的高性能 C++编译器和性能分析器 VTune 等工具，寻找性能瓶颈和优化代码，以利用编译器的强大能力和 Intel 处理器等硬件的性能，从而编写出高质量和高性能的应用代码。

本书介绍的软件优化中，采用的环境和示例主要以 Windows 操作系统下的 C 语言程序为参考，但是其基本思想也可以用在 Linux 操作系统的 C 语言以及 C++语言编程中。第 1 章首先回顾了 C 语言，以帮助读者复习一下 C 语言编程的基本架构、数据类型和数据结构；第 2 章介绍了如何写出好的代码，如何保证代码的易读和可维护性，同时也介绍了一些在代码编写过程中可以采用的通用编程技巧；第 3 章介绍了为什么需要从软件设计开始就考虑性能并贯穿整个软件开发周期，同时介绍了一个自顶向下的软件优化策略、常用的性能调试工具以及如何选择和设计一个好的 Benchmark；

第4章介绍了高性能的Intel C++编译器的使用，包括常用的编译选项、支持的语言扩展等；第5章则介绍了如何使用Intel性能分析器VTune来寻找和分析应用的性能瓶颈以进行进一步的优化；最后一章介绍了一些特别针对Intel处理器特性的优化，包括分支、缓存优化以及如何利用SIMD来提高并行性等，最后介绍了Intel公司的针对处理器作了特别优化的性能库MKL和IPP的使用。

本书可以作为普通高校学生在学习C语言编程之后的软件优化进阶教材，也可供广大C和C++程序员参考。

编著者

2010年9月

目 录

1 C语言综述

C语言作为应用范围最广的高级语言之一，它表达能力强，目标代码效率高，可移植性好，既具有高级语言的优点，又具有低级语言的特点，因此要想成为一名优秀的程序员，你就必须对C语言有所了解。本章就是为了这样的一个目的而写的，它的目标读者至少已经掌握了一定的编程基础(比如掌握其他一门语言)，而现在想要快速地学会C语言，如果是以前学过C语言但不是很熟的读者，本章将会帮助你重新找回C语言的知识。

1.1 示例程序

设一篇英文文件全部由英文单词和空白符组成，其中英文单词被空白符分隔开。下面的程序用来统计正文文件中的行数、单词和字符数。

程序的编程思想是：顺序读入每个字符，当发现一个新的单词时，单词计数器增1。为了标记一个单词的开始与结束，程序引入一个状态变量。当程序发现正处在单词中时，置其值为“在单词中”，发现不在单词中时，置其值为“不在单词中”。下面是程序的源代码：

stdafx.h

```
1  // stdafx.h : 标准系统包含文件的包含文件，
2  // 或是常用但不常更改的项目特定的包含文件
3  //
4
5  #pragma once
6
7
8  #include <stdio.h>
9  #include <tchar.h>
10 #define TOFILE
11
12 // TODO: 在此处引用程序要求的附加头文件
13
```

Example. c

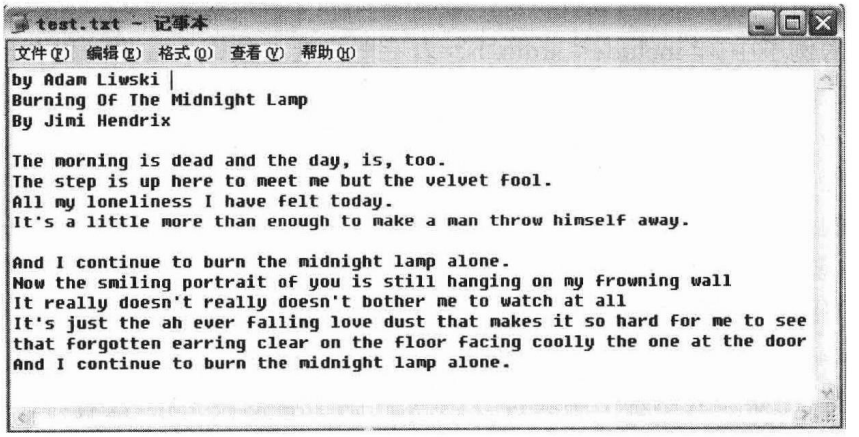
```
// Example.cpp : 定义控制台应用程序的入口点。
//

#include "stdafx.h"
#define IN 1 // 正在单词中
#define OUT 0 //当前不在单词中


FILE *fp ;

typedef struct
{
    int line_num  ;
    int word_num  ;
    int char_num  ;
}INFO ;

void Myout(char fname [40] , INFO *fileinfo);
```

```
int main(int argc, char* argv[])
{
    int  ch , state ;//状态变量
    INFO fileinfo ;
    char fname[40] ; //存储文件名
    printf ("输入文件名! \n");
    scanf("%s%*c", fname );

    if((fp = fopen(fname,"r")) == NULL )
    {//以输入方式打开正文文件
       printf("不能打开文件%s。\n" , fname);
       return 0 ;
    }

    state = OUT ; fileinfo.line_num =fileinfo. word_num = fileinfo.char_num = 0 ;
    while ((ch = fgetc(fp)) != EOF )
    {//这里对刚输入的字符信息ch作某种处理
        ++ fileinfo.char_num ;
       if(ch == '\n')
           ++ fileinfo.line_num ;

       if(ch == ' ' || ch == '\n' || ch == '\t')
            state = OUT ;
       else if(state == OUT)
            {
                state = IN ;
                ++ fileinfo.word_num ;
            }
    }
    fclose(fp) ;

#ifdef  TOFILE
    printf("文件%s有%d行、有%d单词和有%d字符。\n", fname , fileinfo.line_num ,
            fileinfo.word_num , fileinfo.char_num);
#else
    Myout(fname , & fileinfo);
#endif
```

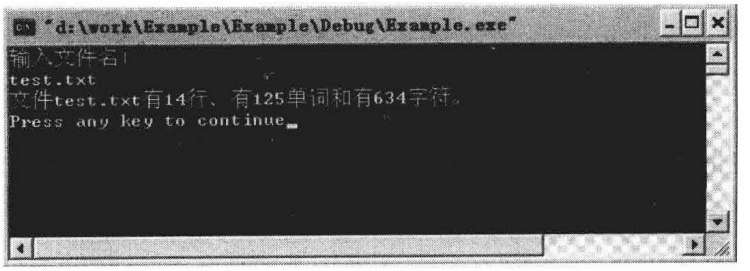
```
57
58      return 1;
59  }
60
61  void Myout(char fname [40] , INFO *fileinfo)
62  {
63      FILE *fp ;
64      char outfilename[40] ;
65
66      printf("请输入输出目标文件：\n");
67      scanf("%s%*c", outfilename );
68
69      if((fp = fopen(outfilename, "w")) == NULL )
70      {
71            fprintf(stderr , "Can't open %s . \n", outfilename);
72            return ;
73      }
74
75      fprintf(fp, "文件%s有%d行、有%d单词和有%d字符。\n", fname , fileinfo->line_num ,
76            fileinfo->word_num , fileinfo->char_num);
77  }
```

上面的程序已经经过作者编译通过，有兴趣的读者可自己在机器上编译试验一下。下面将给出一个测试例子。假设测试例子文件名是 test. txt。

test. txt



运行程序得到的结果如下：



1.2 Hello world

学好一门语言的唯一途径就是用这门语言写程序。学习所有语言都一样，第一个要学会写的程序就是：

打印出语句：hello world！

从上面的示例中我们可以轻易发现C语言的程序入口是main函数。而打印函数就是printf函数。因此，同样可以轻易地写出C语言的hello world：

```
#include <stdio.h>
void main ()
{
    printf("hello world ! \n");
}
```

输出结果：

hello world！

在上面的例子中，#include<stdio.h>表示把标准函数库(standar library)链接到程序。C语言是由一个或多个函数组成的，其中一个函数必须是main函数(即程序入口)。每个C语言都是从main函数开始执行的，也就是main函数的第一个左花括号，也结束于main函数的最后一个右花括号。在C语言中，一对花括号所括起来的语句称为程序块。

在main函数里的语句printf("hello world ! \n");调用了标准库中的函数printf，把“”间的字符串打到屏幕上。在上面的hello world程序中，“”之间的字符串是“hello world ! \n”，这里\n表示的是转义符，意为回车换行。在C语言中还有很多转义符，比如\t表示水平制表符等，要想了解更多的转义符，读者可以参考相关资料，作者不再赘述。printf语句的最后是分号“;”，在C语言中它表示一条语句的结束。

1.3 注　　释

一个好的编程习惯，离不开注释。在程序里加上注释，可以提高程序的可读性。注释不会在程序运行时使计算机执行任何动作，它的存在只是为了帮助自己和其他人阅读和理解程序。

读者也许已经在本章开头的示例中已经注意到“//”的语句了(程序的第1、2句)。对的,这就是C语言中的一种注释。它表示注释掉“//”所在行且属于“//”后面的符号。它的一般写法就是:

//注释语句。

例如:

```
//Example.c: 定义控制台应用程序的入口点。
```

除了“//”语句外,C语言还提供了“/* */”语句。它表示注释掉“/*”和“*/”间的符号。它的用法就是:

/* 注释语句 */

例如:

```
/*
  for(int i = 0 ;i < 20 ;i ++)
  {
        printf("%d\n",i);
  }
*/
```

1.4 基本词汇

C语言的基本词汇包含:

(1) 字面形式的常量,如55、66.0、“good”。

(2) 运算符等。

(3) 关键字,如if 、do等。

(4) 标识符,用于命名程序对象,比如变量的名字。

在C语言中,一个合理的标识符有英文字母或下画线开头,后跟或不跟由字符、下画线、数字符组成的字符列。

1.5 基本数据类型

任何编程语言基础应该就是它提供的数据类型。在C语言中,它提供了整型、

实数型和字符型三种基本数据类型。基本类型允许程序员直接在程序里定义变量，例如示例里的22行和24行。

1.5.1 整型

在示例的22行，该语句定义的就是整型变量。在C语言中，将整型数据的数值范围分成三种：基本整数、短整型、长整型。对这三种整型数据的内部表示的最高位的不同理解又可分为两类：带符号和不带符号。

(1) 基本整型。用int标记。

(2) 短整型。用short int标记，简记short。

(3) 长整型。用long int标记，简记long。

以上三种整型数据的类型都包含有符号的，如果在它们的标记前面加上unsigned，就表示定义的变量是不带符号的正整数。

例如：

```
int   newvar  ;
unsigned int   unsigned_int;
long   long_var;
short   short_var;
```

1.5.2 实数型

C语言的实数类型有三种：

(1) 单精度实数型。用float标记，又称浮点型。

(2) 双精度实数型。用double标记。

(3) 长双精度实数型。用long double型。

例如：

```
float   float_var;
double double_var;
```

在C语言中也会出现实数型的常量，它一般可按下面的格式书写：

正负号　　整数部分　　.　小数部分　　指数部分

其中正负号可有可无，没有正负号表示该常量为正实数。整数部分和小数部分都是字符序列；指数部分的格式是：

E(或者 e) 正负号 十进制数字序列

通常在书写实数型常数的时候整数部分和小数部分是可以任选的,但是必须要有一个部分。小数点和指数部分不可以同时没有。

例如:

```
下面是实数型常数:
    8.      3E6      1.9e-6      .423
```

1.5.3 字符型

字符型顾名思义就是用来表示字符的数据类型。在 C 语言中使用 char 来标记字符型。字符型的计算机内部表示是字符的 ASCII 代码的值,相当于一个整型数据。可如下定义字符型的变量。

例如:

```
char   char_var;
```

字符型常量的书写方法有两种:

(1) 直接使用字符定义。用单引号括住一个字符,如't','h'等对于转义字符可以使用转义符号表示,如回车符可表示成'\n'等。

(2) 直接使用字符的 ASCII 代码来定义。如'A'等价定义是'\101'。

在定义字符型常量的时候,应该注意的是不能与字符串的定义相混淆。在 C 语言中,字符串常量的书写是用一对双引号把字符系列括起来,例如:

"one night in beijing !","only you !","b" 等。

而字符型常量的书写则是使用单引号,因此下面的语句是错误的:

```
char      char_const = "k";
```

在 C 语言中,char 型和 int 型的数据之间是可以通用的,它们之间可以进行混合运算。因此 int 型数据可以使用"%c"来显示 ASCII 码等于该数据的字符,同时也可以使用"%d"来显示出字符型数据所对应的 ACSII 码值。

1.6 指 针

读者也许也已经注意到了 Example.c 示例的第 9 行 FILE * fp 的写法,这就是 C 语言中一个重要的概念——指针。

指针是C语言中最难掌握的概念，它能够模拟传引用调用，并且能够建立和操作动态数据结构。指针可以理解为内存单元的地址，所以指针变量就可以理解为内存中的某个存储单元，该单元存储的是其他内存位置的地址。比如指针 temp_p 是指向 int 变量 temp 的，而 temp 在内存中的地址是 980，那么 temp_p 在内存中的值就是 980。指针变量的定义形式如下：

类型 T　* T类型指针变量名。

对于任何类型 T，都可以建立指向 T 的指针。指针类型使用的两个最重要运算符是地址运算符 & 和间接访问运算符 *，前者生成指针值，即获取变量的地址，后者访问指针所指的对象。

例如：

```
int i = 9,j,* p;
p = &i;
i = 90;
j = * p;
* p = 80;
```

最终，j 的值是 90，i 的值是 80。

这里需要注意的是指针类型有一个特殊的值 null。它表示指针没有指向任何位置。这里需要注意的一个问题是无效指针，所谓无效指针就是指指针变量的值不等于 null，也没有指向任何合法有效的对象。产生无效指针的原因通常是因为在声明指针变量时没有进行初始化。指针在C语言里使用是非常常见的，比如函数的传引用调用、数组指针和指针数组等，读者要想详细地了解这方面的内容，可以阅读C语言的专业书籍。

1.7 自定义类型

Example. c 示例的 11～16 行：

```
typedef struct
{
    int line_num   ;
    int word_num   ;
    int char_num   ;
}INFO  ;
```

自定义了一种类型。在C语言中,程序员可以使用C语言提供的一些基本数据类型组合封装成符合自己需求的类型。C语言提供了以下几种个性类型的定义方法。

1.7.1 结构类型

其实在Example.c示例的11～16行,定义的个性类型就是C语言中最常见的自定义类型。那么什么是结构类型,怎样定义结构类型呢?

结构就是用同一个名字引用的相关变量的集合。结构中可包含多种不同的数据类型的变量。结构是用其他类型的对象构造出来的派生数据类型。定义一个结构类型的一般形式是:

```
Struct 结构类型名{
        成分说明表
};
```

关键字struct引出了结构的定义。接着任选一个标记符来命名要定义的结构类型。用花括号括起来的部分是结构类型的主要部分,它定义了结构类型的成员变量。每个成员变量的定义形式和在程序其他地方的定义变量是一样的,除了不能在定义的时候初始化,形式为:类型 成员变量名。

例如:

```
struct  Test
{
    int  int_member ;
    char char_member ;
    double *  dp_member ;
}
```

定义好了一个结构类型以后,自然地要在程序的其他处用到该类型,这里就包括结构类型变量的定义和结构成员的访问。

在C语言中,结构类型变量的定义和其他类型的变量定义一样,但是也有它特别的地方,定义结构变量的方法有两种。

(1) 已经定义好了结构类型,再利用结构类型名定义变量。形式如:

struct 结构类型名 结构变量名列表;

类如定义Test的变量:

```
struct Test test_var ;
```