

优化程序性能



金戈大王

关注

 0.397 2016.03.31 22:18:24 字数 3,358 阅读 2,685

阅读经典——《深入理解计算机系统》07

本文将介绍非常实用的程序性能优化手段，并用一个案例来详细说明。

1. 为什么要优化程序性能？
2. 衡量性能的指标
3. 未优化版本
4. 提取重复操作
5. 减少函数调用
6. 避免内存读写
7. 还能进一步优化吗？
8. 循环展开
9. 提高并行性
10. 重结合变换
11. 总结

为什么要优化程序性能？

对于c代码而言，从源代码到汇编代码再到机器指令，这中间是有一个编译器在起作用的。编译器发展到现在，它的功能已经不仅仅是将源码编译为机器码，更重要的是它的优化能力。编译器需要根据指令集的特点将代码尽可能地优化，以得到更快的执行速度。

那么，既然有了编译器自动做优化，我们程序员还要手动优化程序性能吗？

答案是肯定的。在很多情况下，编译器无法像程序员一样掌握足够的信息以判断是否可以执行某种优化，更多的情况下，编译器会很谨慎地做少量的优化，以确保程序的正确性。而我们程序员则可以手动采用更深入的优化策略，以获得更高的性能。具体的案例将在下文叙述。

衡量性能的指标

通常性能瓶颈出现在循环处，对于循环遍历的元素数是固定的情况下，所用的时间正比于每个元素消耗的时间。因此我们用CPE（cycles per element）来衡量程序性能。CPE是指对于一个循环操作，平均每个元素所用的周期数。

这样说似乎不是很直观，接下来让案例登场吧。

未优化版本

推荐阅读

面试了一位33岁程序员

阅读 40,451

阿里巴巴为什么能抗住90秒100亿？看完这篇你就明白了！

阅读 52,389

审阅“史上”最烂的代码

阅读 13,643

程序员不满薪资拒绝offer，HR怒称：估计你一辈子就是个程序员了！

阅读 13,287

程序员8年看了15个源代码，阿里6面被淘汰后感叹：技术也没用

阅读 8,711



写下你的评论...

 评论2

 赞21

...

```
3 typedef struct {
4     long int len;
5     data_t *data;
6 } vec_rec, *vec_ptr;
```

这是一个数组结构体，包含两个成员：`len` 表示数组长度，`data` 保存第一个数据的地址。为了通用，将 `data` 设为 `data_t *` 类型，这个类型可以任意定义为 `int`、`float`、`double`。

将数组所有元素乘/加的第一种实现，也就是未优化版本如下：

```
1 #define IDENT 0
2 #define OP +
3
4 /* Implementation with maximum use of data abstraction */
5 void combine1(vec_ptr v, data_t *dest)
6 {
7     long int i;
8
9     *dest = IDENT;
10    for (i = 0; i < vec_length(v); i++) {
11        data_t val;
12        get_vec_element(v, i, &val);
13        *dest = *dest OP val;
14    }
15 }
```

为了通用乘法和加法，我们用 `IDENT`、`OP` 的不同组合来区分两种运算。上面代码定义用于处理加法，如果把 `IDENT` 定义为 `1`，并把 `OP` 定义为 `*` 就可以用来处理乘法了。

这种写法也许是我们最常用的写法，虽然现在看不出有什么问题，但接下来我们将分析它的性能瓶颈。

提取重复操作

这个版本最容易被指出的问题就是循环条件调用了函数 `vec_length`，不管这个函数具体是如何实现的，对于长度固定的数组来说，这样做都是一种冗余。因为其实我们可以在循环开始前定义一个局部变量 `length` 保存数组的长度值，这样就只需要调用一次 `vec_length`，一定会降低程序的运行时间。新的程序版本如下：

```
1 /* Move call to vec_length out of loop */
2 void combine2(vec_ptr v, data_t *dest)
3 {
4     long int i;
5     long int length = vec_length(v);
6
7     *dest = IDENT;
8     for (i = 0; i < length; i++) {
9         data_t val;
10        get_vec_element(v, i, &val);
11        *dest = *dest OP val;
12    }
13 }
```

用实际测得的CPE来表示两个版本间的性能差异比较有公信力，请看下表。

推荐阅读

面试了一位33岁程序员

阅读 40,451

阿里巴巴为什么能抗住90秒100亿？看完这篇你就明白了！

阅读 52,389

审阅“史上”最烂的代码

阅读 13,643

程序员不满薪资拒绝offer，HR怒称：估计你一辈子就是个程序员了！

阅读 13,287

程序员8年看了15个源代码，阿里6面被淘汰后感叹：技术也没用

阅读 8,711

combine1和combine2性能比较

表中，每种实现都给出了五类数据类型的测试结果，包括整数加法、整数乘法、浮点数加法、单精度浮点数乘法和双精度浮点数乘法。由于每类运算本身执行一次需要的时间就不一样，因此需要分开比较。

注意，combine1已经采用了-O1级别的编译器优化，但combine2的用时仍然比前者短了数秒。

编译器为什么不自动做这个优化呢？因为它没那么聪明呗。它不知道 `vec_length` 这个函数的返回值是不变的，因此只能谨慎起见，不优化。

减少函数调用

如何做进一步的优化就不那么容易想到了。不过想想之前《函数调用栈》中讲的函数调用过程是多么的繁琐，提示我们应该尽量减少函数调用。新的版本如下：

```
1  /* Direct access to vector data */
2  void combine3(vec_ptr v, data_t *dest)
3  {
4      long int i;
5      long int length = vec_length(v);
6      data_t *data = get_vec_start(v);
7
8      *dest = IDENT;
9      for (i = 0; i < length; i++) {
10         *dest = *dest OP data[i];
11     }
12 }
```

现在，循环里不再有 `get_vec_element` 方法了，取而代之的是直接用下标索引取数。说实话，这是一个非常鲁莽的做法，因为这样写的代码不具有扩展性，而且大大破坏了原来程序的抽象和封装，不是面向对象程序应有的作风。因此，现在我们讲的是提高程序性能的手段，而不是说所有程序都应该这样写。当性能不是程序首要考虑的因素时，我们根本不需要做任何优化。优化之后，代码会变得难以理解，通常要求附带文档解释说明。

CPE测试结果如下。

函 数	页码	方 法	整 数		浮 点 数		
			+	*	+	F*	D*
combine2	333	移动 vec_length	8.03	8.09	10.09	11.09	12.08
combine3	336	直接数据访问	6.01	8.01	10.01	11.01	12.02

combine2和combine3性能比较

虽然性能提高不多，但关键时候一点点的性能提升也很重要。

避免内存读写

现在，让我们关注循环中唯一的一句代码，如何提高这句代码的性能？

如果查看汇编代码，就很容易发现其中的问题（汇编码就不再贴出来了）：每个循环都要读写

推荐阅读

- 面试了一位33岁程序员
阅读 40,451
- 阿里巴巴为什么能抗住90秒100亿？
看完这篇你就明白了！
阅读 52,389
- 审阅“史上”最烂的代码
阅读 13,643
- 程序员不满薪资拒绝offer，HR怒称：估计你一辈子就是个程序员了！
阅读 13,287
- 程序员8年看了15个源代码，阿里6面被淘汰后感叹：技术也没用
阅读 8,711



```
1  /* Accumulate result in local variable */
2  void combine4(vec_ptr v, data_t *dest)
3  {
4      long int i;
5      long int length = vec_length(v);
6      data_t *data = get_vec_start(v);
7      data_t acc = IDENT;
8
9      for (i = 0; i < length; i++) {
10         acc = acc OP data[i];
11     }
12     *dest = acc;
13 }
```

循环中只用局部变量 `acc` 保存计算的中间值，循环结束后再赋值给 `*dest`。由于现代处理器都优先使用寄存器保存局部变量的值，因此可以大大提高程序性能。

看看现在的CPE吧，简直是飞一般的快啊。

函 数	页码	方 法	整 数		浮 点 数		
			+	*	+	F*	D*
combine3	336	直接数据访问	6.01	8.01	10.01	11.01	12.02
combine4	338	累积在临时变量中	2.00	3.00	3.00	4.00	5.00

combine3和combine4性能比较

再回头想一下，为什么编译器不做这样的优化？肯定还是因为有不确定的因素。假如说 `dest` 和数组 `v` 的地址有交叉，那么原始版本的程序在循环中可能会改变数组元素的值，而当前版本的循环就不会改变数组元素的值，造成不一致的结果。因此编译器在不确定的情况下不敢擅自做优化。

还能进一步优化吗？

我们需要知道处理器的性能极限是多少，以此判断我们的程序还能否进一步优化。

上一篇文章《从零开始制作自己的指令集架构》中详细讲述了指令集架构的内部结构。但是，不得不指出，现代处理器架构完全不同于Y86，虽然在某些具体的实现方面沿用了Y86的技术，但实际的逻辑组成却是这样的：

推荐阅读

面试了一位33岁程序员

阅读 40,451

阿里巴巴为什么能抗住90秒100亿？看完这篇你就明白了！

阅读 52,389

审阅“史上”最烂的代码

阅读 13,643

程序员不满薪资拒绝offer，HR怒称：估计你一辈子就是个程序员了！

阅读 13,287

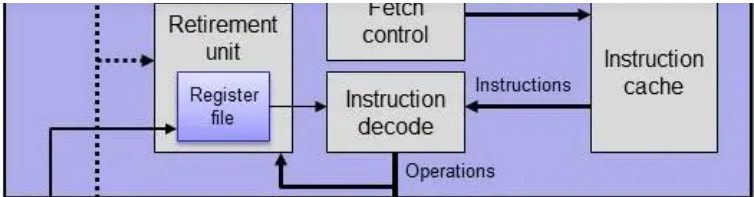
程序员8年看了15个源代码，阿里6面被淘汰后感叹：技术也没用

阅读 8,711



写下你的评论...

评论2 赞21



现代处理器架构

整个处理器分为两大部分，指令控制部分和执行部分。前者负责取指以及写回寄存器，后者又包括多个功能单元，比如FP add、load、store等等，分别负责各自独立的计算和存取内存操作。

上一篇文章提到了现代处理器用的乱序执行技术，这个技术就依赖于这些功能单元。每个单元都是完全流水线化的，意味着FP add单元可以每周期完成一次加法操作。而各个单元间又是完全并行执行的，而且不必照顾指令的执行顺序。如果一条指令需要多个单元执行，那么把每个单元需要执行的任务排队进入流水线就可以了。

这里需要提出两个名词：

- Latency：时延。一条指令从开始到完成所用的时间。
- Issue：发射时间。两条指令连续发射的时间间隔。完全流水线情况下应该为1。

时延限制了顺序执行的运算的性能，而发射时间限制了流水线级别的并行运算的性能。理论上在两种限制下所能达到的最低CPE见下表（吞吐量和发射时间是同一个含义）。

界限	整数		浮点数		
	+	*	+	F*	D*
延迟	1.00	3.00	3.00	4.00	5.00
吞吐量	1.00	1.00	1.00	1.00	1.00

两种基本界限

可见，不同类型数据和不同操作对应的时延不同，但它们在完全流水线下的CPE都可以达到1。

下一步，就要研究怎样才能突破时延对CPE的限制，最终达到吞吐量对CPE的限制。之所以combine4只接近了时延对CPE的限制，是因为代码中每两次运算间是顺序执行的。之所以是顺序执行的，是因为下一次的运算用到了上一次运算的结果，产生了数据依赖。所以，接下来我们应该考虑如何消除数据依赖。

循环展开

将本来需要n次的循环变成n/2次，每次循环内部做两个元素的操作，这种技术就称为循环展开。看代码：

```
1  /* Unroll loop by 2 */
2  void combine5(vec_ptr v, data_t *dest)
3  {
4      long int i;
5      long int length = vec_length(v);
6      long int limit = length - 1;
```

推荐阅读

- 面试了一位33岁程序员
阅读 40,451
- 阿里巴巴为什么能抗住90秒100亿?
看完这篇你就明白了!
阅读 52,389
- 审阅“史上”最烂的代码
阅读 13,643
- 程序员不满薪资拒绝offer，HR怒称：估计你一辈子就是个程序员了!
阅读 13,287
- 程序员8年看了15个源代码，阿里6面被淘汰后感叹：技术也没用
阅读 8,711



```
13     }
14
15     /* Finish any remaining elements */
16     for (; i < length; i++) {
17         acc = acc OP data[i];
18     }
19     *dest = acc;
20 }
```

代码中每次循环处理两个元素，循环展开因数为2。

现在，CPE为

函 数	页码	方 法	整 数		浮 点 数		
			+	*	+	F*	D*
combine4	338	无展开	2.00	3.00	3.00	4.00	5.00
combine5	349	展开 2 次	2.00	1.50	3.00	4.00	5.00
		展开 3 次	1.00	1.00	3.00	4.00	5.00
延迟界限			1.00	3.00	3.00	4.00	5.00
吞吐量界限			1.00	1.00	1.00	1.00	1.00

combine4和combine5性能比较

可以发现，整数运算性能提升了，而且展开3次的情况下整数加法和整数乘法都达到了吞吐量界限，但是浮点数运算性能却毫无改善。这是因为虽然循环展开了，但是两次运算间仍然存在直接的数据依赖，导致流水线的并行能力仍然没有发挥出来。不过整数乘法却越过了延迟界限，原因涉及到重结合变换（reassociation transformation），我们将在后面详细解释。

提高并行性

现在，必须真正地消除数据依赖了。为了让下次运算不再需要上次运算的结果，我们可以将整个运算分为两个并行分支，用两个局部变量分别累加奇数项和偶数项，最后再合并到一起。代码如下：

```
1  /* Unroll loop by 2, 2-way parallelism */
2  void combine6(vec_ptr v, data_t *dest)
3  {
4      long int i;
5      long int length = vec_length(v);
6      long int limit = length - 1;
7      data_t *data = get_vec_start(v);
8      data_t acc0 = IDENT;
9      data_t acc1 = IDENT;
10
11     /* Combine 2 elements at a time */
12     for (i = 0; i < limit; i += 2) {
13         acc0 = acc0 OP data[i];
14         acc1 = acc1 OP data[i+1];
15     }
16
17     /* Finish any remaining elements */
18     for (; i < length; i++) {
19         acc0 = acc0 OP data[i];
20     }
21     *dest = acc0 OP acc1;
22 }
```

推荐阅读

面试了一位33岁程序员

阅读 40,451

阿里巴巴为什么能抗住90秒100亿?
看完这篇你就明白了!

阅读 52,389

审阅“史上”最烂的代码

阅读 13,643

程序员不满薪资拒绝offer，HR怒称：估计你一辈子就是个程序员了!

阅读 13,287

程序员8年看了15个源代码，阿里6面被淘汰后感叹：技术也没用

阅读 8,711



写下你的评论...

评论2 赞21

combine4	338	累积在临时变量中	2.00	3.00	3.00	4.00	5.00
combine5	349	展开 2 次	2.00	1.50	3.00	4.00	5.00
combine6	352	2 次展开, 2 路并行	1.50	1.50	1.50	2.00	2.50
延迟界限			1.00	3.00	3.00	4.00	5.00
吞吐量界限			1.00	1.00	1.00	1.00	1.00

combine4、combine5和combine6性能比较

果然，浮点数运算性能也提高了不少。经过测试，如果提高到3路、4路甚至5路并行，浮点数运算也会下降到吞吐量界限。

重结合变换

另一种提高并行的方式是采用重结合变换。依据加法和乘法的结合律，在循环展开的基础上，重新结合三个数的运算顺序，就可以实现性能提高。代码如下：

```
1  /* Change associativity of combining operation */
2  void combine7(vec_ptr v, data_t *dest)
3  {
4      long int i;
5      long int length = vec_length(v);
6      long int limit = length - 1;
7      data_t *data = get_vec_start(v);
8      data_t acc = IDENT;
9
10     /* Combine 2 elements at a time */
11     for (i = 0; i < limit; i += 2) {
12         acc = acc OP (data[i] OP data[i+1]);
13     }
14
15     /* Finish any remaining elements */
16     for (; i < length; i++) {
17         acc = acc OP data[i];
18     }
19     *dest = acc;
20 }
```

原理何在呢？其实很简单，仍然是数据依赖的问题。先计算的两个数 `data[i]` 和 `data[i+1]` 是没有任何数据依赖的，因此这次计算可以和下一次与 `acc` 的运算完全并行化，这就比之前先计算 `acc` 要好得多了。

实际效果也很明显。

函 数	页码	方 法	整数		浮点数		
			+	*	+	F*	D*
combine4	338	累积在临时变量中	2.00	3.00	3.00	4.00	5.00
combine5	349	展开 2 次	2.00	1.50	3.00	4.00	5.00
combine6	352	2 次展开, 2 路并行	1.50	1.50	1.50	2.00	2.50
combine7	355	2 次展开, 重新结合	2.00	1.51	1.50	2.00	2.97
延迟界限			1.00	3.00	3.00	4.00	5.00
吞吐量界限			1.00	1.00	1.00	1.00	1.00

combine4、combine5、combine6和combine7性能比较

总结

推荐阅读

面试了一位33岁程序员

阅读 40,451

阿里巴巴为什么能抗住90秒100亿？看完这篇你就明白了！

阅读 52,389

审阅“史上”最烂的代码

阅读 13,643

程序员不满薪资拒绝offer，HR怒称：估计你一辈子就是个程序员了！

阅读 13,287

程序员8年看了15个源代码，阿里6面被淘汰后感叹：技术也没用

阅读 8,711



写下你的评论...

评论2 赞21

简书

首页

下载APP

搜索

beta

登录

注册

combine1	331	抽象的 -O1	12.00	12.00	12.00	12.01	13.00
combine6	352	展开 2 次, 2 路并行	1.50	1.50	1.50	2.00	2.50
		展开 5 次, 5 路并行	1.01	1.00	1.00	1.00	1.00
延迟界限			1.00	3.00	3.00	4.00	5.00
吞吐量界限			1.00	1.00	1.00	1.00	1.00

性能优化前后对比

可以看到, 当采用了展开5次, 5路并行的优化后, 无论哪一种运算都达到了吞吐量界限, 表明我们的优化非常成功。

最后, 有两件事情需要提醒大家:

- 本文所讲的这些优化方法, 在大部分编译器中都已经实现了。但是它们有可能不会实行这些优化, 或需要我们手动设置更高级别的优化选项才行。所以, 作为一个程序员, 我们应该做的是尽量引导编译器执行这些优化, 或者说排除阻碍编译器优化的障碍。这样可以使我们的代码在保持简洁的情况下获得更高的性能。迫不得已时, 我们才去手动做这些优化。
- 循环展开、多路并行并不是越多越好。因为寄存器的个数是有限的, x86-64最多只能有12个寄存器用于累加, 如果局部变量的个数多于12个, 就会被放进存储器, 反倒严重拉低程序性能。

想要亲自测试的同学请移步我的GitHub仓库[optimization_demo](#)。

关注[作者](#)或[文集](#)《[深入理解计算机系统](#)》, 第一时间获取最新发布文章。

< 上一篇

查看连载目录

下一篇 >

21人点赞

...

"小礼物走一走, 来简书关注我"

赞赏支持

还没有人赞赏, 支持一下

金戈大王

总资产382 (约26.97元) 共写了19.0W字 获得1,305个赞 共1,128个粉丝

关注

文章来自以下连载

深入理解计算机系统

2.8W字 71,893阅读 437人关注

关注连载

推荐阅读

面试了一位33岁程序员

阅读 40,451

阿里巴巴为什么能抗住90秒100亿?
看完这篇你就明白了!

阅读 52,389

审阅“史上”最烂的代码

阅读 13,643

程序员不满薪资拒绝offer, HR怒称: 估计你一辈子就是个程序员了!

阅读 13,287

程序员8年看了15个源代码, 阿里6面被淘汰后感叹: 技术也没用

阅读 8,711

全部评论 2 只看作者

按时间倒序 按时间正序




狮女柔心_Nicole陶

2楼 2016.04.04 06:50

讲的很详细

赞 回复



金戈大王 作者

2016.04.04 09:02

@狮女柔心 谢谢~

回复

添加新评论

推荐阅读

面试了一位33岁程序员

阅读 40,451

阿里巴巴为什么能抗住90秒100亿？看完这篇你就明白了！

阅读 52,389

审阅“史上”最烂的代码

阅读 13,643

程序员不满薪资拒绝offer，HR怒称：估计你一辈子就是个程序员了！

阅读 13,287

程序员8年看了15个源代码，阿里6面被淘汰后感叹：技术也没用

阅读 8,711

被以下专题收入，发现更多相似内容

 程序员

 iOS移动开发社区

 技术干货

 首页投稿（暂停...

 Android进阶

 JC专题

 设计匠艺

展开更多

推荐阅读

更多精彩内容

《深入理解计算机系统》| 优化程序的性能

编写运行的快的程序有三个因素：①选择合适的算法和数据结构；②理解编译器的能力，使用有效的方式让编译器能进行优化；③...



唐鱼的学习探索

阅读 1,851 评论 1 赞 11



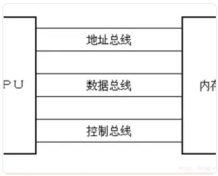
汇编入门（长文多图，流量慎入！！）

8086汇编 本笔记是笔者观看小甲鱼老师（鱼C论坛）《零基础入门学习汇编语言》系列视频的笔记，在此感谢他和像他一样...



Gibbs基

阅读 21,336 评论 8 赞 97



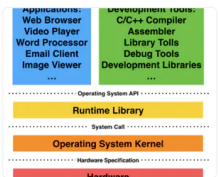
iOS 程序员的自我修养 — 读《程序员的自我修养-链接、装载...

2016年国庆假期终于把此书过完，整理笔记和体会于此。关于书名 书名源于俄罗斯的演员斯坦尼斯拉夫斯基创作的《演员...



李剑飞的简书

阅读 5,133 评论 1 赞 56



写下你的评论...

评论2 赞21

xiaogmail 阅读 3,150 评论 1 赞 30



11月10号

王夕月 阅读 80 评论 0 赞 0



推荐阅读

面试了一位33岁程序员

阅读 40,451

阿里巴巴为什么能抗住90秒100亿?
看完这篇你就明白了!

阅读 52,389

审阅“史上”最烂的代码

阅读 13,643

程序员不满薪资拒绝offer，HR怒称：估计你一辈子就是个程序员了!

阅读 13,287

程序员8年看了15个源代码，阿里6面被淘汰后感叹：技术也没用

阅读 8,711



写下你的评论...

评论2

赞21