软件系统的性能工程
麻省的Technology6.172研究所
教授查尔斯·莱瑟森和教授。顺朱利安讲义3

家庭作业2：分析串行合并排序

[注：此分配使用了OCW用户可能无法使用的AWSand/或Git功能。]

在本家庭作业中，您将被介绍到重要的分析工具。您将学习使用Perf，它将详细介绍在程序中花费时间的位置，以及Valgrind工具套件中的一个缓存和分支预测分析程序。这些工具将帮助您识别代码中进一步优化的部分。最后，您将使用这些工具来优化串行合并排序例程。

一般来说，当您关心程序的性能时，最好的方法是收集正确的内容，然后进行评估。在某些情况下，这个初始实现的许多部分（甚至是整个部分）可能足以满足您的需要。然而，当你需要改进表现时，你必须首先决定把精力放在哪里。这就是配置分析非常有用的地方。配置分析可以帮助您识别程序中的性能瓶颈。

1开始

代码结构

背诵的代码在背诵目录中。您将看到文件isort.c，它包含插入离子排序程序的代码，和sum.c，它分配大量的整数并对其中的一个样本求和。对于家庭作业，你负责改进的程序是在家庭作业目录中。

构建代码

一般来说，使<目标>将为一个特定的目标二进制文件构建代码。请记住，要使用调试符号和断言（对于gdb调试很有用），必须使用构建

```
$ make<target> DEBUG=1
```

每当一个问题询问绑定性能时，我们希望您在使用awsrun的AWS作业机器上运行程序。

家庭作业2：教授串行合并排序2

提交解决方案

请记住，在提交和推送最终更改之前，要显式地将新文件添加到存储库中。

```
$ git  添加
$ git  提交状态
$ git  推
$ git
```

Ifgit状态显示任何修改的文件，那么您可能还没有正确地检查到存储库中的代码。我们经常推荐你。

2背诵：表演和实习

首先，我们将探讨如何使用两个非常有用的性能分析工具： Perf和委托委托。这些工具允许您识别程序中的性能瓶颈，并度量相关的性能属性，例如缓存和分支缺失。使用正确，它们可以帮助您弄清楚为什么您的代码运行得比它应该的要慢。

2.1性能

perf是Linux2.6+基础系统的分析工具。它使用采样来收集有关重要的软件、内核和硬件事件的数据，以便在一个程序中重新定位性能瓶颈。它生成时间的详细记录。

注意：如果性能未安装到您的AWS VM上，请运行

```
$的安装工具-通用工具
```

您可以使用性能记录和性能报告来分析您的程序的性能。pertcrerd生成运行代码时发生的事件的记录，perf报告允许您以交互方式查看它。

注意：为了使perf正确地注释代码，您必须在编译程序时传递-g标记到clang。这个标志不影响性能，它告诉编译器生成调试符号，这允许perf和gdb等工具将机器码行与源代码行关联起来。如果您使用调试=1运行它，则提供的Makefile将执行此操作。

要在一个程序上记录性能事件，请运行以下操作：

```
$ perf记录<程序_name><程序_参数>
```

然后，您可以通过从相同的目录中运行性能报告来查看结果：

```
$ perf报告
```

注意：您可能会看到一个警告屏幕，上面写着“内核地址映射受到限制”。这可以，只要按任何键继续报告。

个人的AWS机器通常的性能（和不一致性）不如专用的AWSRUN机器，后者是通过AWSRUN访问的。你也可以在这里描述性能：

```
$运行记录<程序_name><程序_参数>
```

这将在AWS云队列机器上生成一个报告，而不是在您的个人实例上，因此您需要传递一些额外的参数来查看它。我们为您提供命令报告：

```
$-perf报告
```

注意：所有的性能报告应该在不运行的情况下运行。

花点时间来探索这个报告的内容，试着弄清楚它的意思。

您可以在注释输出中看到C代码和汇编指令。性能计数器事件通常与导致外部事件的指令之后或指令下面的功能结构相关联。

文件排序。c包含一个插入排序例程。使用编译程序

```
$使isort调试=1
```

现在，running./isort nk将对埃勒门斯克时间进行排序。运行时间：

```
$运行记录。/isort10000 10$aws-性能
报告
```

识别分支故障、时钟周期和指令。诊断程序中的性能瓶颈。

检查项目1：记录一个瓶颈。

2.2 Cachegrind

Cachegrind（一个虚拟工具）是一个缓存和分支预测分析器。从课堂上回想一下，从l1恶病肠读取的速度比从mram读取的速度快100倍！优化高速缓存项是性能工程的一个关键部分。

在像onAWS这样的虚拟环境中，提供有关分支和缓存信息的硬件事件经常不可用，而soperf可能没有帮助。显示您的程序与无轴的缓存层次结构和分支预测器交互，甚至可以在没有可用的硬件性能探测器的情况下使用。

下面是一个关于如何识别缓存丢失、分支丢失、时钟周期和程序使用访问程序执行的指令的示例：

家庭作业2：教授串行合并排序4

```
$进程——工具=缓存-分支sim=是的<程序_name><程序_参数>
```

注意：尽管工具=缓存使用模拟器测量缓存和分支预测器行为，但它的模拟基于它运行的架构。在不同的机器上运行时，您应该期望不同的结果，例如在个人实例上运行与awsrun机器时。

文件sum.c包含一个程序，它分配一个U=1000万个元素的数组，然后随机选择N=1亿个元素。使用：

```
$make和
```

检查项目2： Runsum缓存不足，以识别缓存性能。这可能需要一点时间。在输出中，查看D1和LLdmists。D1代表最低级别的
缓存（L1），而LL表示最后一个（最高）级别的数据缓存（在大多数机器上，L3）。这些数字和你的期望相符吗？试着随便玩一下
值N和U的总和。如何减少痉挛的次数？

提示：要找到关于你的CPU和它会扫描的信息，请使用lscpu。
检查：向助教解释你对前两个检查项目的回答。

3作业：分类

分析代码通常可以让你对程序是如何工作的以及为什么执行它有价值的见解。在本练习中，我们提供了一个在家庭作业中合并排序/sort_a.c的简单实现。你可以通过输入make来编写所有类型的代码。制作完成后，使用./sort <num_elements> <num_trials>运行代码。

文章1：比较计算代码调试=1代码rsus调试=0编译优化代码。解释一下，在你比较这个程序的不同版本的性能时，使用指令计数作为时间的替代品的优点和缺点。

确保您正在评估其余的实验的非调试版本。

我们希望识别性能瓶颈，并逐步提高合并排序的性能，如下面的任务所述。当您在每个任务中对排序路由进行改进时，我们希望您在改进之前保留一个排序例程的副本，并在测试套件中不断添加排序例程的新版本（具有不同的名称），因此我们可以配置和比较不同版本的性能。测试代码的设置是为了方便的，您可以轻松地添加新的排序例程来进行测试，只要您保留相同的函数签名

家庭作业2：教授串行合并排序5

这个排序例程，也就是说，您所做的每个排序例程都应该有相同类型的assort_a提供的insort_a.c。另外，不要删除任何内部函数中的静态关键字，保留已编码的合并排序算法的结构，并且不要为这个作业从根本上改变它。

3.1 Inlining

你想看看这些功能能有多少帮助。复制代码fromsort_a.c intosort_i.c，并将所有例程名称从<函数>_a更改为<函数>_i。使用内联关键字，内联一个或多个功能离子insort_i.c和c.c。c。要将thesort_iroutine添加到测试套件，请在te stFunc，thatspecifiessort_i下取消main.c中的行。配置文件和注释内联程序。

报告2：解释您选择在线执行哪些函数，并报告内联和未内联排序例程之间的性能差异。

编译器并不总是使用内联关键字进行内联函数。使用带注释的汇编代码来帮助您验证编译器是否真的内联了您想要的函数。

尝试内联sort_i递归函数（您可能会发现正向声明是有用的）。当您内联一个递归函数时，它将只扩展递归的次数，然后将递归。

文章3：解释递归函数可能的性能。分析使用缓存进程收集的数据如何帮助您衡量这些负面的性能影响？

3.2指针与数组

您了解到，在操作数组时，指针访问可能比数组索引更有效，除了在排序算法中，您还希望在排序算法中实现这种效率。要实现这一点，您首先复制您的排序例程insort_i.c intosort_p.c，并将函数名称insort_p.c更新为<函数>_p。然后修改代码insort_p.c，以使用指针而不是数组和includesort_ p。

写文章4：给出一个为什么使用指针可以提高性能的原因。报告您在实现中观察到的任何性能差异。

3.3粗化

递归ofsort_p例程insort_p.c中的基本情况只是一个元素。您希望压缩递归，以便递归基大小写的排序超过一个元素。请复制所有更改intosort_c.c，并更新函数名。使用最快的正确排序例程作为一个基本的forsort_c.c。然后压缩递归insort_c.c。我们提供了一个插入排序例程inisort.c，您可以用作基本情况下的排序算法。您也可以编写您最喜欢的排序算法，并将它用于您的数据库。请记住，为了使用不在其他地方定义的函数，您首先需要编写一个函数原型。将sort_c包含到您的测试套件中。

文章5：解释你使用什么排序算法，以及你如何选择的数量
元素将在基本情况下进行排序。报告您所观察到的性能差异。

3.4减少内存使用

请注意，在merge_c功能中使用了左右两个临时记忆划痕空间。您希望通过只使用一个临时内存划伤空间（从而将总临时内存使用量减少一半），并在合并操作中使用输入数组作为另一个内存划伤空间来优化内存。与前面一样，c提供最新的更改intosort_m.c并更新函数名。然后实现上面描述的内存优化。

文章6：解释在你的sort_m.c中的任何性能差异。可以有一个编译器吗
自动为你做这个优化，并节省你所有的努力？为什么或为什么不呢？

3.5重用临时内存

虽然你已经减少了临时内存我们年龄insort_m.c一半，你发现每次分配和解除分配临时内存划伤空间是必要的。相反，您希望在排序结束时分配所需的内存，并在排序完成后重新分配它。要完成这个最终的优化，请复制您的

排序常规intosort_f.c。然后，实现上述insort_f.c中所描述的内存增强，并包括sort_fin您的测试套件。

报告7：报告在yoursort_f.c中的任何性能差异，并解释
在使用配置文件分析数据时存在的差异。

6.172软件系统的工程性能

2018年秋季