如何开始使用 OpenMP *

由Richard Gerber （英特尔） (https://software.intel.com/zh-cn/user/544841)发布于 2012 年 6 月 7 日 简体中文

摘要

您可能知道现在，要获得的最大的性能优势，具有超线程技术，从应用程序需要将并行执行。并行执行需要线程参与，并应用线程化并非易事。您可能不知道是，工具，如 OpenMP * 可以简化此过程许多。

这是三本白皮书，教您 — — 经验丰富的 C/c + + 程序员来说，一系列中的第一个如何使用 OpenMP 来充分利用超线程技术。本文首次向您展示如何进行并行处理循环，称之为工作共享。第二篇白皮书将教您如何利用非循环并行能力和 OpenMP 的其它特性。最后一篇讨论的 OpenMP 运行时库函数、 Intel® c + + 编译器和如何调试您的应用程序，如果出错。

OpenMP 简介

OpenMP 的设计人员希望提供无需程序员，了解如何创建、 同步和毁坏线程，或甚至要求他们以确定要创建的线程数量的线程化应用的一个简单的方法。为实现这些目标，OpenMP 的设计人员开发了一套编译指示、 指令、 函数调用和环境变量，可以明确指导编译器如何和在何处线程插入应用程序之间的独立于平台

的。大多数循环可以插入在循环前的一个制导语句立即进行线程处理。此外，通过在具体细节留给编译器和 OpenMP 情况下，您可以将更多的时间确定的循环进行线程化和如何以最佳的方式重组算法来最大性能。它使用到"热点"，在您的应用程序的最耗时循环的线程时，实现了 OpenMP 的最大性能的。

能耗和简易性的 OpenMP 是最佳展示通过查看示例。下列循环将 32 位 RGB （红色、 绿色、 蓝） 像素转换为 8 位灰阶像素。编译指示即可，在循环前立即插入，就是一切所需的并行执行。

```
01 #pragma omp parallel for
02
03 for (i=0; i < numPixels; i++)
04
05 {
06
07    pGrayScaleBitmap[i] = (unsigned BYTE)
08
09              (pRGBBitmap[i].red * 0.299 +
10
11               pRGBBitmap[i].green * 0.587 +
12
13               pRGBBitmap[i].blue * 0.114);
14
15 }
```

让我们看看循环。首先，示例使用 '工作-sha 环网，在 OpenMP 中用于描述在线程间的工作分配常规术语。工作共享一起使用**的**结构时，此例所示，循环迭代分布在多个线程，每个循环体迭代是由一个或多个线程执行一次完全地并行。OpenMP 决定来创建和如何以最佳方式创建、 同步和毁坏他们的线程数量。所有编程人员需要完成的工作是告知 OpenMP 应对哪个循环进行线程化。

OpenMP 将循环可以进行线程处理在其的以下五个限制：

- 循环变量必须类型 。DWORD 等无符号的整数将无法工作。
- 对比操作必须在窗体*: loop_variable* <，< =、 >，或 > = *loop_invariant_integer*

- 在**for**循环的第三个表达式或增量部分必须为整数添加或整数相减并通过一个循环不变式值。
- 如果对比操作 < 或 < =，则循环变量必须在每次迭代，增量和与此相反，如果对比操作 > 或 > =，则循环变量必须每次迭代减少。
- 循环必须为一个基本块，这意味着除了**退出**语句来终止整个应用程序，允许从循环内部到外部无跳转。如果使用的陈述**goto**或**break**，他们必须在循环中，不是外部跳转。这同样适用于异常处理;例外情况必须困在循环内部。

尽管这些限制条件显得有些苛刻，但不符合条件的循环可以轻松地重新编写来达到这些要求。

编译基本要点

使用 OpenMP 编译指令需要一个兼容 OpenMP 的编译器和线程安全库。理想之选是 Intel® c + + 编译器 7.0 版或更高版本。（Intel® Fortran 编译器还支持 OpenMP。）下面的命令行选项加入编译器指示它需要注意 OpenMP 编译指令、 插入线程。

如果您省略命令行上的**/Qopenmp**，则编译器将忽视 OpenMP 编译指令，提供一种非常简单的方式，无需改变任何源代码生成一个单线程版本。英特尔® c + + 编译器支持 OpenMP 2.0 规范。一定要浏览的发行说明和提供最新的更新 Intel® c + + 编译器的兼容性信息。完整的 OpenMP 规范，请访问 http://www.openmp.org (http://www.openmp.org/)*.

对于条件编译，编译器会规定 _openmp。如果需要如下所示，此定义可进行测试。

```
1 #ifdef _OPENMP
2
3
4     fn();
5
6
7 #endif<
```

所有 OpenMP 编译指令的通用形式是：

如果行不以**pragma omp**开头不是 OpenMP 编译指令。编译指示的完整列表位于http://www.openmp.org (http://www.openmp.org/)在找到对技术指标.

选择 C 运行时线程安全库使用**/MD**或**/MDd** （调试） 命令行编译器选项。选择这些选项，当使用 Microsoft Visual c + + *，通过选择多线程 DLL 或调试多线程 DLL 的 C/c + + 项目设置的代码生成目录中。

几个简单的示例

下列例子将展示如何简单的 OpenMP 就是使用。共同练习中，其他问题需要解决的问题，但是就是为了向您将。

问题 1:下列循环将一个阵列修剪为 0 的范围.....255。您需要使用 OpenMP 编译指示其进行线程化。

解决方案：只需将以下编译指示在循环前立即插入。

问题 2:下列循环生成的数字 0.....平方根表 100。您需要在其使用 OpenMP 进行线程化。

解决方案：循环变量的有符号的整数，因此需要进行更改并不需要添加编译指示。

避免数据相关性和竞态条件

当循环满足以上全部五个循环条件并编译器线程循环时，它可能仍然无法正常工作由于存在着数据相关性。数据相关性即存在时使用不同的线程读上执行的循环-更特别是循环迭代不同迭代或写共享的内存。请考虑以下示例阶乘计算。

```
1 // Do NOT do this. It will fail due to data dependencies.
2     // Each loop iteration writes a value that a different
```

```
3      iteration reads.
4      #pragma omp parallel for
5      for (i=2; i < 10; i++)
6      {
7         factorial[i] = i * factorial[i-1];
8      }
```

编译器将线程循环，但它将失败，因为至少一个循环迭代存在数据相关性在一个不同的迭代。这种情况下被称为竞态情形。竞态条件只发生在使用共享的资源 （如内存） 和并行执行。为了解决这些问题可以重新编写循环或者选择了不同的算法，不包含竞态条件。

竞态条件很难探测到，因为在特定情况下，可能变量"赢得竞争"，在订购 tha t 碰巧正确让程序正常。计划一次工作并不意味着，它将始终配合使用。测试您在各种机器上的程序，部分采用超线程技术，具有多个物理处理器，部分是很好的起点。工具如还可以帮助英特尔线程检测器，作为可以清晰的眼睛。传统调试器在探测竞态条件，因为一旦它们让一个线程停止"竞争"，同时其它线程将继续大幅度地改变运行时行为无法使用。

管理共享数据和个人数据

几乎每个循环 (至少如果它是一个很有用) 读取或写入内存，而编程人员的工作是告知编译器哪部分内存应在线程间共享，以及哪部分应保持私有。当内存被识别为共享，所有的线程访问的确切的同一个内存位置。当内存被识别为私有时，但是，一份该变量的专为若要访问的每个线程。在循环结束时，这些私有副本将被销毁。默认情况下，除了私有的循环变量，共享所有变量。内存可以被声明为 private 以下两种方式。

- 将内部循环 — — 真的在并行 OpenMP 指令内部变量声明-不用静态密码。
- 指定 OpenMP 指令上的私有语句。

下列循环无法正常运行，因为变量**temp**为共享。它需要为私有。

```
06 thread
07
08
09 // is reading variable temp another thread might be writing to
10
11 it
12
13
14 #pragma omp parallel for
15
16
17 for (i=0; i < 100; i++)
18
19
20 {
21
22
23    temp = array[i];
24
25
26    array[i] = do_something(temp);
27
28
29 }
```

下列两例均宣告变量**temp**为私有内存，即可解决该问题。

```
24    array[i] = do_something(temp);
25
26
27 }
28
29
30 // This also works. The variable temp is declared private
31
32
33 #pragma omp parallel for private(temp)
34
```

```
35
36 for (i=0; i < 100; i++)
37
38
39 {
40
41
42    temp = array[i];
43
44
45    array[i] = do_something(temp);
46
47
48 }
```

每次您指示 OpenMP 进行并行处理循环，您应仔细检查所有内存引用，包括所做的被调用函数的参考。并行结构内部声明的变量被定义为私有除时它们已声明为静态声明符，因为静态变量不在堆栈上分配。

降低

积累值的循环十分普遍，，OpenMP 具备特定的子句来支持它们。请考虑下列循环来计算的整数数组的总和。

```
01 sum = 0;
02
03
04 for (i=0; i < 100; i++)
05
06
07 {
08
09
10    sum += array[i];   // this
11
12 variable needs to be shared to generate
13
```

```
14
15              //
16
17  the correct results, but private to avoid
18
19
20               //
21
22  race conditions from parallel execution
23
```

在上一循环变量**sum**必须共享，以生成正确的结果，但它还必须是私有的以允许访问多个线程。为了解决 th 案例，OpenMP 提供的**reduction**子句，用来高效地在循环中的一个或多个变量进行数学归纳结合。下列循环采用了归纳子句生成正确的结果。

```
01  sum = 0;
02
03
04  #pragma omp parallel for reduction(+:sum)
05
06
07  for (i=0; i < 100; i++)
08
09
10  {
11
12
13     sum += array[i];
14
15
16      }
```

实际上，OpenMP 为每个线程提供变量**sum**的专用拷贝，当线程退出时，它将值加在一起，并将结果放在一个全局变量的副本。

下表列出了可能的归纳，以及在初始变量，这也是数学标识值-为临时私有变量。

Operation	Temporary private variable initialization
+ (addition)	0
- (subtraction)	0
* (multiplication)	1
& (bitwise and)	~0
| (bitwise or)	0
^ (bitwise exclusive or)	0
&& (conditional and)	1
|| (conditional or)	0

通过在给定并行结构上指定逗号分隔变量和归纳，在循环中的多个归纳是可能的。唯一的要求是：

- 只需一次归纳中列出归纳变量
- 它们无法声明为一直不变，和
- 它们无法声明为私有并行结构中。

循环调度

负载平衡，平均分配工作线程，它是并行应用性能最为重要的属性。负载平衡至关重要，因为它可以确保处理器的忙碌的大多数，如果不是所有的时间。没有平衡的负载，某些线程可能会很快地完成之前，让处

理器资源闲置，浪费性能机会。

在循环结构，负载平衡不足是的差异通常导致循环迭代之间计算时间。通常很容易通过检查资源代码来确定循环迭代计算时间的差异。在大多数情况下，您会看到循环迭代消耗了统一的时间段。时，不为 true，则它可能有可能找到一套消耗类似时间量的迭代。例如，有时所有甚至迭代消耗的消耗时间大致一样为组的所有奇迭代。同样，它可以发生： 上半年循环的一套所消耗的时间大致一样的第二部分作为。另一方面，它可能无法找到拥有统一执行时间的循环迭代的集合。任何应用程序属于这种情况下，您应该提供此额外的循环调度 informatio n-OpenMP，以便它可以更好地分布在线程 （并因此处理器） 的循环迭代，以实现最佳负载平衡。

OpenMP 中，默认情况下，假定所有循环迭代都消耗同样的时间量。此前提下，OpenMP 分发大致平均和最大限度减少因错误共享引起的内存冲突几率的方式中的线程的循环迭代。这是因为循环通常接触内存按顺序，因此将循环分成在大型数据块的前半和第二半时使用最少两个线程会使重叠内存的机会。尽管这可能是内存问题的最佳选择，它可能不利于保持负载平衡。遗憾的是，反过来也是如此;有哪些可能对负载平衡有利可能对内存性能不利。性能工程师，因此，必须之间达到平衡最佳内存使用和最佳负载平衡，通过测量性能了解哪种方法可生成的最佳结果。

使用以下语法并行结构上情况下，循环调度信息传送给 OpenMP。

```
#pragma omp parallel for schedule(kind [, chunk size])
```

可以为 OpenMP 提供四种不同的循环调度类型 （提示） 下, 表所示。可选的参数 （块），当指定时，必须是一个循环不变量正整数。

Kind	Description
static	Divides the loop into equal-sized chunks or as equal as possible in the case where the number of loop iterations is not evenly divisible by the number of threads multiplied by the chunk size. By default, chunk size is loop count / number of threads. Set chunk to 1 to interleave the iterations.
dynamic	Uses an internal work queue to give a chunk-sized block of loop iterations to each thread. When a thread is finished, it retrieves the next block of loop iterations from the top of the work queue. By default, chunk size is 1. Be careful when using this scheduling hint because of the extra overhead required.
guided	Similar to dynamic scheduling, but the chunk size starts off large and shrinks in an effort to reduce the amount of time threads have to go to the work-queue to get more work. The optional chunk parameter specifies the minimum size chunk to use. By default, the chunk size is approximately the loop count / number of threads.
runtime	Uses the OMP_SCHEDULE environment variable to specify which one of the three loop-scheduling types should be used. OMP_SCHEDULE is a string formatted exactly the same as would appear on the parallel construct.

示例

问题： 将下列循环并行化处理

```
01 for (i=0; i < NumElements; i++)
02
03
04 {
05
06
07    array[i] = StartVal;
```

```
08
09
10      StartVal++;
11
12
13  }
```

解决方案： 注意数据相关性

如上所述，循环包含数据相关性，使其不可能进行快速变换进行并行处理。新的循环，如下所示，填充阵列方式相同，但不包含数据相关性。此外，新循环还可以写使用 SIMD 指令。

```
01  #pragma omp parallel for
02
03
04   for (i=0; i < NumElements; i++)
05
06
07  {
08
09
10      array[i] = StartVal + i;
11
12
13  }
```

观察，该代码并非 100%相同因为变量**StartVal**的值没有增加。因此，当并行循环完成时，变量将获得不同的系列版本生成值。如果在循环后需要**StartVal**的值，则需要，如下所示的附加声明。

```
01  // This works and is identical to the serial version.
02
03
```

```
#pragma omp parallel for

 for (i=0; i < NumElements; i++)

{

   array[i] = StartVal + i;

}

StartVal += NumElements;
```

摘要

编写 OpenMP 主要是为了减轻编程人员从线程化的详细信息支持侧重于更重要的问题。使用 OpenMP 编译指令，大多数循环可将采用线程处理一个简单的声明。这是 OpenMP 和一项重要优势所在。此白皮书介绍了的概念和更简单的侧的 OpenMP，以帮助您起步。接下来的两篇文章将构建打好基础，从而能够使用 OpenMP 线程化更复杂的循环和更多通用的结构。

参考资料

- OpenMP 规范： http://www.openmp.org (http://www.openmp.org)*

相关的链接

- 更多工作共享与 OpenMP * (/en-us/articles/more-work-sharing-with-openmp)
- 高级 OpenMP * 编程 (/en-us/articles/advanced-openmp-programming)

有关编译器优化的更完整信息，请访问平板优化通知 (/zh-cn/articles/optimization-notice#opt-cn)。