转载 2014年08月11日 17:20:05

* 4051

转载的文章，其中谈到了java比C++慢的一些观点。

Why Java Will Always Be Slower than C++

by Dejan Jelovic   
  
为什么Java永远比C++慢？   
  
耍过Java程序，或者用Java码过程序的人都晓得，Java要比用C++写成的原生程序要慢。这是咱用Java时已经承认的事实。   
　　不过，很多人想要说服我们说这只不过是暂时的，他们说Java从设计上来讲并不慢，相反，只是现在的JIT实现相对比较嫩，有很多能优化的地方JIT并没有优化到，拖了后腿。其实不然，不管JIT们多牛，Java永远要比C++慢。   
　　   
　　我想说...   
　　宣扬Java不慢于C++的人往往是觉得，（语法）严格的语言，可以让编译有更大的优化空间。因此，除非你想做人肉编译器优化整个程序，否则通常都是编译器做得更好。   
　　这是真的。在数值计算领域，Fortran仍然胜于C++，的确因为它更严格。不用担心指针瞎搅和，编译器可以更安心地优化。C++想打败Fortran的唯一办法，就是好好设计一个像Blitz++那样的库。   
  
　　测试...   
　　Java可以跟得上C++的地方，就是基准测试。计算起第N个斐波纳契数，或者运行起Linpack，Java没理由不跟C++跑得一样快。当所有的计算都放在一个类里，并且只使用基本的数据类型，比如说int或者double时，Java编译器的确能跟得上C++的脚步。   
  
　　事实...   
　　当开始在程序中使用对象的时候，Java就放松了潜在的优化。这一节会告诉你为什么。   
  
　　1. 所有的对象都是从堆里分配的。   
　　Java从栈里分配的，就只有基本数据类型，如int，或者double，还有对象的引用。所有的对象都是从堆里分配的。   
　　当有大量语义上是一回事的对象时，这不成问题。C++同样也是从堆上分配这些对象。但是，当有值语义不同的小对象时，这就是一个主要的性能杀手。   
　　什么是小对象？对我来说，就是迭代器们。在设计中，我用了很多迭代器。别人可能会用复数。3D程序员可能会矢量或者点类。处理时间序列的人可能会有时间类。使用这些类的人，无一例外地讨厌把不费时间的栈上分配换成花费固定时间的堆上分配。假如把它放在一个循环里，就变成了O(n)对0了。如果再加一层循环，没错，又变成O(n^2)对0了。   
  
　　2. 大量的转换。   
　　得益于模板，好的C++程序员甚至可以写于完全没有转换的牛程序。不幸，Java没有模板，所以Java代码总是充满了转换。   
　　对于性能，它们意味着什么？呃，在Java里所有的转换都是很费时的动态转换。多费时？想想你可能会怎么样实现转换的：   
　　最快的方法就是，给每一个类赋值一个序号，然后用一个矩阵来描述任意两个类是否相关的。如果是的话，需要给指针加上多少的位移才能进行转换。这种方法的伪码看起来应该是这样的：   
DestinationClass makeCast (Object o, Class destinationClass) {   
Class sourceClass = o.getClass (); // JIT compile-time   
int sourceClassId = sourceClass.getId (); // JIT compile-time   
  
int destinationId = destinationClass.getId ();   
  
int offset = ourTable [sourceClassId][destinationClassId];   
  
if (offset != ILLEGAL\_OFFSET\_VALUE) {   
return <object o adjusted for offset>;   
}   
else {   
throw new IllegalCastException ();   
}   
}   
好一堆代码。这只是一个简单的情景——用矩阵来表示类的关系浪费了一部分内存，没有哪个成熟的编译器会这样子做。他们会使用map或者遍历继承树，这样会变得更慢。   
  
　　3. 攀升的内存占用。   
　　Java程序储存数据占用的内存大概是相当的C++程序的两倍。原因如下：   
　　1. 启用了垃圾收集的程序一般都比不使用垃圾收集的程序多花50%的内存。   
　　2. 本来C++里在栈上分配的对象，到了Java就在堆上分配了。   
　　3. Java对象比较大，因为所有的对象都有一个虚表，还要加上对（线程）同步的原生支持。   
大的内存映像让程序更大概率被放到磁盘的交换区去。没有什么比交换文件更慢的了。   
　　4. 缺少更细致的控制。   
Java原来就是作为一种简单的语言来设计的。很多在C++里让程序员控制细节的特性在Java里都被一脚踢开了。   
　　比如说，在C++里可以改进引用的位置(?)。或者一次申请和释放很多个对象。或者用指针耍一些小技巧，更快地访问成员。   
　　5. 没有高层次的优化。   
　　程序员处理高层次的概念。而编译器处理剩下的低层次概念。对于程序员来说，一个叫Matrix的类就代表了比一个叫Vector的类更高层次的概念。而对于编译器来说，这些名字都是符号表的一个入口。他们只关心类里面有哪些函数，函数里面有哪些语句。   
　　这样想一下，比如说要实现一个exp(double x, double y)函数，计算出x的y次幂。对于一个编译器，它能只看一下这个函数，然后指出，exp(exp(x, 2), 0.5)可以优化成x自己吗？当然不行。   
　　编译器能做的优化只是语句层面的，而y是在编译器里面的。即使程序员知道两个函数是对称的，可以把它们都消去，或者函数的调用顺序只是相反的，除非编译器能只瞄一下语句，然后指出来，不然优化是不可能完成的。   
　　所以，如果想要完成一个高水平的优化，必须存在某种方法，可以让程序员来告诉编译器优化的规则。   
  
　　没有哪个流行的程序语言/系统可以做到这点，至少已知的方法，比如微软承诺的智能语言，都不能。即便如此，在C++里可以用模板元编程来实现对高层次对象的优化。临时消除，部分求值，对称函数调用的消去，和其它可以用模板实现的优化。当然，不是所有的高层次优化都可以这样做。并且实现这些东西相当麻烦。但是大多数都可以完成，有人已经用这些技术实现了好些时髦的库。   
  
　　不幸的是，Java没有任何元编程的特质，因此在Java中不会有这种高层次的优化。   
　　所以...   
  
　　由于存在这种语言特性，Java不可能达到C++这种速度。这相当程序上暗示了，对于要求高性能的软件和竞争激烈的COTS舞台上，使用Java不是一种明智的选择。但是因为它和缓的学习曲线，它的容错，和它庞大的标准库，所以适合开发中小型自用和定制软件。   
  
　　附记...   
　　1. 有人向James Gosling（谁？google之...）提交了很多可以改进Java性能的语言特性。文本在这里。不幸的是，Java语言已经有四年没有改动过了，所以看起来这些提议似乎不会在一夜之间被实现。   
  
　　2. 最有可能往Java里加入泛型的是Generic Java。又很不幸的是，GJ只是通过在编译时把所有类型信息去掉来支持泛型。所以最后面执行环境看到的，仍然是缓慢的转换。   
  
　　3. 垃圾收集的FAQ包含了关于垃圾收集慢于定制分配器的信息（上面第四点）。   
  
　　4. 这里是一篇宣称垃圾收集比栈分配的快的文章。但是它的要求是物理内存必须是程序实际需要的内存的七倍之多。还有，它描述的是一种stop-and-copy（是不是那种执行到一半，然后停下来，把内存拷到另外一块内存，同时清除垃圾的那种方法？），而且还不是并发的。   
  
　　反馈...   
　　我收到很多关于这篇文章的反馈。附上一些典型的评论，还有我的回答：   
  
　　“你还忘记了指出在Java里所有的方法都是虚方法，因为没有人会加上final关键字。”   
　　事实上，不使用final关键字不是问题的关键所在，使用者才是。同时，虚函数也没有问题，但是却失去了优化机会。自从JIT们知道怎么样内联虚函数，这就变得不那么显著了。   
  
　　JIT可以内联虚函数，所以Java可以比C++更快。   
　　C++也可以使用JIT编译。不信的可以看看.NET的C++编译器。   
  
　　到最后的时候，速度并不重要。电脑浪费了大部份时间在等待用户输入。   
　　速度仍然很重要。我仍然在等我的笔记本启动起来，我在等我的编译器停下来，我还要等Word打开一个超长的文档。   
　　我在一个金融公司工作。有时候我必须对一个很大的数据集进行模拟。速度在这种情况下都很重要。   
  
　　有些JIT可以在栈上分配一些对象。   
　　当然，一些。   
  
　　你的转换代码看起来很丑。可以在类的继承层次上检查类。   
　　首先，这样只比矩阵查找快一点点而已。   
　　第二，这样只能查找类，类只占多少？低层次的细节往往是通过接口来实现的。   
  
　　哈，那么我们都应该使用汇编？   
　　不是的，我们都要使用对业务有用的语言。Java提供了庞大的标准库，让很多任务变得容易，因此Java是伟大的。它比其它所有的语言更容易移植（但并非100%可移植——不同的平台有不同问题）。它具有垃圾收集机制，简化了内存管理，同时也让某些构造如闭包可实现。   
　　但是，同时，Java和所有的语言一样，也有瑕疵。在值语义的类型上缺少支持。它的同步并不是很有效率。它的标准库建立在异常检查之上，把实现拖进了接口。它的性能可以更好。它的数学库有些恼人的问题。诸如此类。   
  
　　这些缺憾都是大问题吗？看你用它做什么。因此，在几种语言里，连同它的编译器以及可以选择的类库里选择对你的工程有利的一种。