自调节软件 计算机 应用程序 (6)

计算机科学 2000Vol. 27№. 4

21-22,8

# 自调节软件

Self-Tuning Software

TP31

苏运霖
(暨南大学计算机科学系 广州 510632)

**Abstract** This paper introduces a new concept——Self-tuning software which has just been put forward recently abroad. It also surveys the development of the new thing.

**Keywords** Computer software, Self-tuning software

## 1 问题的提出

凡学过算法分析或具有编程经验的同行都知道，相同的算法，对于不同的问题或数据或数据的安排可能有不同的效率。如快速排序和合并排序，在最坏情况下其复杂性都是$O(n^2)$，但在一般情况下，却是$O(n \log n)$。自然，我们想知道，有没有办法，在排序之前，就能去避免最坏情况的出现，使得算法的运行最坏也不至于成为$O(n^2)$，而若能成为最佳情况就更好了。

其次，有经验的计算机程序员也都知道，被裁制成为在一个特定的机器上快速运行的程序，即使它是可搬家的，但搬到其它机器上之后，效率便大大降低了，号称可搬家程序往往在所有平台上都低效地运行，唯一例外的就是在它原来运行的那个平台。

今天，计算机处理器的频率大大提高，意味着计算机的运算速度已经达到很高的水平。但是今天的高速处理器相对于数据从内存中被收集或送回去的缓慢速度，仍然形成强烈反差，从而影响了整个计算机系统的运行效率。应当如何来解决这个问题，使得计算机处理器的高速度真正发挥出作用来？这也是当今许多计算机科学家或勤于思考的计算机程序人员竭力试图解决的问题。

上述所有这些问题，都唤起了一个新思想的产生。这就是目前已在美国某些地方出现的自调节软件(Self-tuning Software)。按照他们的话来说，程序人员正在使计算机来做调整软件的艰苦工作，以使软件运行于其上的硬件获得最好的利用，或者说，自调节软件去适配它的环境，更好地运行于其环境上，从而也使环境得到最好利用。

苏运霖 教授

## 2 自调节软件的概念和实例

现在一些研究人员正在思考如何让计算机做这件事。自调节软件考察它在上边运行的硬件的能力并且生成程序代码。它利用它所求得的代码来进行工作。这个软件主要是针对科学计算，建立起进行普通的计算任务一例如矩阵乘法一的子程序。这样就可以来进行自调节了。用他们的话说："过去，人们得到一台新的机器，就人工地运行它、来获得某些经验。然后就对软件结构进行一些改动以使它非常有效地运行。现在的实用学就是把这类的机灵放到一个程序当中。"

他们也把注意力放在处理计算机的系统结构同内存的不匹配上，其办法是建立起多层次的内存。打个比方来说，如果把计算机的存储看作是一个大的超市，超市的货物的放置是一个相当复杂的问题。如果把那些大量顾客需要的商品放在偏僻不易寻找的位置，那肯定会使顾客感到极为不便。因此，宜于把大批的抢手物品放在收款处的附近(这相当于寄存器)，其它一些物品有的放在货架上，有的放在过道旁专设的铺位，还有更多的东西放在储藏室(仓库)里。软件就相当于货物清单。顾客通过这个货物清单可以很容易地找到自己所需要的商品，然后把它装到小货车上。当然，这就要求货物清单必须同货物存放的具体位置"对号"，否则就会出现顾客在通道上往返穿梭，仍然找不到自己要的东西的现象。计算机的建筑师们也就运用这个思想来处理刚才所说的速度差异性一建立起储存器的层次体系—即从大而缓慢的内存，到若干层次较小但都是较快的"高速缓冲"，直到寄存器一处理器的运算在这里进行。但是，为此数据就必须适当地排序，最困难的部分也就是找出一个最优的重新排序。

自调节软件就是运用这样一些思想而被研制出来的，以下我们介绍最近才公诸于世的三个自调节软件。

第一个是由位于 Knoxville 的田纳西大学的计算机科学家 Jack Dongarra 和他的同事 R. Clint Whaley 及 Antoine Petitet 所研制的 ATLAS（Automatically Tuned Algebra Software—自动调节线性代数软件），用于建立矩阵相乘，数的矩形数组运算的有效算法—这实际上也就奠定了所有有效的科学计算的基础。据 Dongarra 称："从这样的子程序所获得的效益的改进"是"十分显著的"，"我所说的改进，不是 10%或 20%的改进，而是 300%的改进"。这个有效的矩阵乘法把每一个矩阵分成为一些较小的方块，对这些方块进行乘法，然后把结果重新组合回来。这里的技巧是，比如，这些方块应该有多大；并且按照什么顺序来把它们相乘。这就依赖于计算机的存储结构。

第二个软件叫做 PHiPAC（Portable High Performance ANSI C-可搬家高性能的美国国家标准局的 C 语言）。这是由加州大学伯克利分校的 Jim Demmel 和他的同事们研制的。PHiPAC 也是利用计算机的高速缓冲结构并且把它同描述一些可能的算法的一组参数相匹配。然后它就固定于一个算法上，来极小化数据在存储器的层次上上下移动的次数。

第三个自调节软件取了个怪名，称为西方最快速的富里叶变换，简称为 FFTW。实际上这是一个免费软件。对于一个给定的机器，它只需用几秒钟来调节自己，然后就开动计算，其速度可以同为某个特定计算机编制的实现相抗衡。这个程序是麻省理工学院的两名博士研究生 Matteo Frigo 和 Steven Johnson 智慧的产物。

FFTW 为计算富里叶变换建立一些子程序，这在所有类型的科学计算中都是十分关键的，一个富里叶变换寻找在似乎杂乱无章的数据中的周期模式，例如在由地理形成所记录的冰川时期的盛衰模式。

用于快速计算富里叶变换的基本过程即 FFT，是于 60 年代引进的。它的工作过程是把一个大的变换分解成一些小的，而这是通过分析大的数据集开始。每个小的变换又继续作分解，这些小的变换计算起来要比大的变换的计算容易得多。即使把所有这些计算合在一起，其计算的纯节省量也仍十分可观。

Johnson 指出，FFT 算法实际上为你提供了分解变换的广泛自由。例如，一个大小为 100 的变换，可以分解为两个大小为 50 的变换，进一步，又可分成四个大小为 25 的变换等等。这里的关键是如何找出一个恰好适合于计算机的内存体系的分解。FFTW 的调节阶段解决一个测试问题，尝试所有可能的分解直到找出对于一个特定大小的变换来说最快的一个。比起矩阵程序所面对的问题，它是一个更有限制的问题。这就是为什么 FFTW 可以运行得如此快速。Frigo 和 Johnson 说，在他们的 FFTW 网（www.fftw.org）上，他们每天都要收到几千个访问。这些访问的范围极其广泛，如有的是要求帮助分析脉冲星的天文学家，有的是想要通过计算机调节他的吉它的音乐家。

前边我们提到了 FFTW 调节的速度，这里再补充介绍 ATLAS 和 PHiPAC 的调节速度，相比于 FFTW，这两者的调节速度就要慢得多了。ATLAS 要花费个把小时来运行调节，而 PHiPAC 典型地要花费几天时间。但一旦自调节软件找到了一个好的算法，则其结果可以永久地用于需要作矩阵乘法的任何程序且在机器上能快速运行。由于这些程序一般都是很大的，运行时间都较长，因此这样的代价应该说是用户可以接受的。这可以从两个方面来说明，一是如果这是一个生产性程序，即它要千百次地运行，如气象预报程序，那么，即使每次运行的改进不是很大，但日积月累，效果仍旧是明显的。二是如果这是一个以天或月计的大型计算，举例来说，如果自调节软件把运算复杂性从 $O(n^3)$ 改进成为 $O(n \log n)$，设 $n=100$，则在原来的计算机上的计算复杂性以 10000 为阶，而在改进之后计算机的计算复杂性锐减为以 200 为阶，即若原来需要运行 10000 小时，那么花上几天时间来进行调节，充其量也不过上百个小时，加上刚才的 200 个小时，也仍然不会超过 1000 小时，这就是平摊分析的思想。下图示出在许多不同的处理器上 ATLAS 所生成的算法比计算机厂商提供的软件运行得还要快。

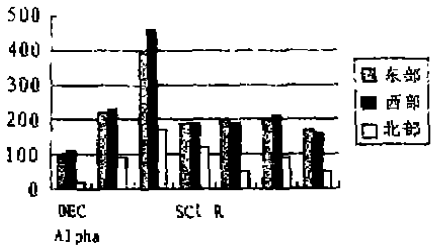


图 1　调节了的性能（左边线为厂商提供软件性能，中间线为 ATLAS 性能，右边线为可搬家软件性能。上图摘自 Science v286. 1999）

计算机的供应商经常在他们自己的机器上提供快速运行的程序库，现在他们对于自调节软件也很感兴趣。Intel 公司的一位高级软件设计专家 Bruce Greer 说，"这些确实是伟大的工具"。他说"Intel 的程序家们将使用 FFTW 作为基准来判断我们软件的质量。如果在推出更好的性能之前，我们不能同 FFTW 相匹敌，那就表明我们的努力还不够"。

（下转第 8 页）

要的原因。一般，有两种提高性能的策略，一是提高对象指令和对象管理系统的效率，二是降低 Java 程序的特性的使用而提高性能。

JVM 中对象指令最低效的部分在于对象的存储管理，特别是新的对象分配，针对存储管理进行改进是提高效率的有效手段。根据当前 JVM 对 new_quick 指令优化实现的思想，可以得到一种对象分配的优化方法。为所有对象开设一个全局池，具有较短生命周期的临时对象和诸如例外、中断之类的高优先级对象能够有效地在这种全局池上进行分配。垃圾回收策略将急剧降低某些分配的性能，采用高效的垃圾回收算法可以大大提高效率。如采用"世代回收技术"对多线程应用十分有效。

JVM 极大地依赖于诸如存储器管理等操作系统、本地机器结构的支持，Java 必须花费时间和空间将其服务转化成为宿主系统所使用的模型、在操作系统级别实现一个自动存储器管理器将会提高 Java 的性能。

提高 Java 程序执行性能的另外措施是在编译时尽量减少对象指令的数目，在 C++ 编译器中，这种优化称之为插入(inlining)。插入通过将调用替换为方法的实际代码而实现，所以是直接执行代码，没有额外的方法调用开销，可以采用同样的"方法展开"优化 Java 中方法调用的开销。Java 程序中的字符串类、数字类和例外实例通常生成临时对象以调用方法。在这种情况下，插入方法代码和对象构造器的代码，可以免去费时繁琐的对象分配和方法调用，降低在程序中所需要的垃圾回收次数，达到提高性能的目的。

即时编译技术也是提高 Java 程序执行的有效手段，即时编译器将 Java 字节代码转化成本地机器代码，显然本地代码的执行比解释执行要快，减轻存储器管理压力的另外一种技术是对类和对象进行压缩，Netscape Navigator 当前就是采用压缩格式读取和使用其 Java API 类，这种压缩降低了对存储器的需求，然而这种技术对性能究竟有多大的影响还需进一步进行研究。

## 参 考 文 献


1 Java(tm) Virtual Machine Specification Sun Microsystems, Inc ,1995

2 Gosling J, McGilton H. The Java Language Environment, A White Paper. Sun Microsystems, Inc. ,1995

3 Patterson D A, Hennessy J L. Computer Architecture: A Quantitative Approach, 2nd Edition. Morgan Kaufmann Publishers, Inc. ,1995


(上接第 22 页)

## 3 自调节软件往何处去

上边我们已经简要介绍了自调节软件的一些概况，它的出现已经引起了人们相当大的关注，因此有必要对于它的今后发展谈谈我们的看法。

自调节软件是一个崭新的事物，尽管目前它还处于幼年时期，但它代表了人们在软件上所追求的方向。因此可以肯定地说，某一个具体的自调节软件不一定会永久地存在下去—它肯定会被性能更好的自调节软件所代替，但是作为一个整体，自调节软件必将成为人们追求的目标，就如同人们曾经努力去研制可搬家的软件一样。在 21 世纪里，人们一定会努力去研制自调节软件，以期这样的软件可以充分利用它在上边运行的硬件的所有能力。

其次，适应于自调节软件的工作，或许在计算机的硬件结构上也有必要有一个调整，以便使这种软件的调节可以更容易进行。比如，在寄存器的数量的设置上，或许就应该考虑增加寄存器的数量才能使计算得以更快进行，以解决上边谈到的处理器处理的快速与存储器存取相当缓慢之间的不匹配。

第三个问题是，目前的自调节软件几乎都是面向科学计算的，因此我们自然要说科学计算确实很重要，但在今天，计算机的应用领域已经是如此广泛，以致使科学计算在整个计算机应用所占有的份额已降到较低的地位。我们希望见到适用于各种各样需求的自调节软件，特别是希望能有适应于网上工作或在分布式系统上工作的自调节软件，如果那样，自调节软件就会发挥更大的作用。

第四个问题是，我们看到，无论是 ATLAS 还是 FFTW，其工作原理都是基于分而治之的思想，因此，也就有这样一个问题，当软件所处理问题不具有可分性，是否能构造出自调节的软件，或者在这样一个问题中，如何去找出可分的部分来，针对这个部分来作些自调节。显然，这种情况，也仍值得我们加以研究。

## 参 考 文 献


1 Cipra B. Self-Tuning software Adapts to Its Environment. Science, 1999,286(5437)

2 Thomas W. Parsons Introduction to Algothms in Pascal. John Wiley & Sons, 1995