计算机科学2000Vol. 27№. 1

可移动代码的一种优化技术

An Optimization Technique for Mobile Code

王明文　孙永强

（上海交通大学计算机科学与工程系　上海200030）

Abstract　Partial evaluation provides a unifying paradigm for a broad spectrum of work in program optimization, compiling, interpretation and the generating of automatic program generators. To improve the performance of mobile codes system, we use the partial evaluation technique to specialize the mobile code in the server, which can speed up the running of the mobile code in clients and decrease the communication quantity in the network.

Keywords　Mobile code, Partial evaluation, Program optimization

1. 引言

关于可移动的程序代码的研究起源于异构网络环境的需求。象 Scheme[2]，Tcl[3]和 Telescript[4]这样的解释型语言都在一定程度上支持程序代码的可移动性。和移动代理的迁移所不同的是，可移动代码只是程序代码的迁移而不是运行程序（包括程序代码、数据以及运行状态）的移动。最为我们所熟悉和使用的可移动代码当然是由 Sun 微系统公司开发的 Java 语言[5]了。Java applets 和 Java servlets 是目前使用最为广泛的可移动代码技术。

Java applets 是用于 WWW 网页的 Java 程序。当用户使用内置 Java 解释器的浏览器浏览 Web 站点上的网页时，网页服务器上的程序被自动下载到用户的本地机器上并被装载执行。由于程序的执行是在浏览者的本地计算机上进行，也就没有了网络延迟，因此程序可以给用户提供复杂的图形化界面并且能对用户的动作作出快速反应。不受信任的 applets 程序只能在安全的 Java 解释器中解释执行，因而它不能存取或破坏敏感信息。Metis 的瘦客户（Thin-client）框架可以被看成是 Java applets 的一般化。在 Metis 中任意客户可以下载一个用 Java 写的应用的前端程序，它负责寻找一个或多个实现整个应用的网络服务并与之进行交互以完成任务。其它支持 applets 的系统还有：CNRI 提供的支持用 Python 写的 applets 程序的 Web 浏览器 Grail；Sun 微系统公司提供的支持用 Tcl/Tk 写的 ap-

础，扫描方法可以提供给 JAPS 的其余模块使用。

总结　本文所涉及的项目 JAPS 属于国家攀登计划 B 的课题“大规模并行编译和操作系统的某些关键技术”的一项研究内容，同时受到了“863”课题“基于 Java 的使用化分布并行工具包”的支持，已经通过了国家863项目鉴定。

在系统设计和实现中，还存在着一些问题，如与并行性分析的结合不是太紧密，导致一些分析结果不够准确；数据流的分析还不太完善；简单性原则不易确定；判断语句的分支概率的确定方法、粒度分析的方法等还有待探讨；并且没有实现表达式的运行时刻的动态调整。为了解决这些问题，还要进行进一步的分析和探讨。

致谢　感谢杜建成博士，对本文工作提出了许多有益的启发。

参考文献

1　张德富. 并行处理技术. 南京大学出版社，1992. 13～15
2　Gosling J, McGilton H. The Java Language Environment: A White Paper, 1995
3　Zhu Genjiang, et al. PRG: Procedure Reference Analysis in Extracting Non-data Parallelism. In: 1996 IEEE Second Interl. Conf. on algorithms & Architectures for Parallel Processing. Singapore, 1996. 76～83
4　Zhu Genjiang, et al. A path-based method of parallelizing C++ programs. ACM SIGPLAN NOTICES, 1994, 29(2). 54～64
5　杜建成. 基于 NOW 的面向对象语言自动并行化中几个问题的研究：[博士论文]. 南京大学，1999
6　[美]Banerjee U. 超级计算中的依赖关系分析. 沈志宇，赵克佳译. 湖南科学技术出版社，1991. 26～42
7　Aho A V, et al. Compilers, Principles, Techniques, and Tools. Addison-Wesley Publishing Company, 1996. 660～694

plets 的接插件 Tcl plugin，它可以被嵌入到现有的浏览器中用以支持用 Tcl/Tk applets 程序的执行。

Java servlets 是能够被动态地装载到执行支持 Java 的 Web 服务器上的 Java 程序。目前支持 Java servlets 的 Web 服务器有 Sun 微系统公司的 Web Java Server 1.0。Java servlets 实现了新服务器功能的快速引入。由于没有存取操作的启动开销，Java servlets 有着比 CGI 脚本更高的效率。象不被信任的 applets 程序一样，不被信任的 servlets 程序也只能在安全的 Java 解释器中运行以确保它们不能对敏感数据进行存取或破坏。

可移动代码的迁移只是程序代码的迁移，它并没有携带程序的运行状态和数据等其它信息。对一段可移动代码，可看成一个程序，而程序由程序代码和输入参数构成，且输入参数又可分为静态（不变或不常变）和动态两部分，本文讨论应用部分计值技术来优化可移动代码，在异构网络环境中能提高可移动代码的运行效率并减少网络的通讯量。

2. 部分计值

部分计值是一种对给定程序和它的部分数据生成一个高效程序的系统方法。其抽象描述为：设 f 为两个参数 k（已知）和 u（未知）的程序（函数），先完成 f 中只依赖已知参数 k 的所有计算，把 f 中依赖未知参数 u 而不能计算的部分称作为剩余程序记作 f_{k0}。则 f_{k0} 有性质：

$$f_{k0}(u)=f(k0,u) \quad (1)$$

这里 k0 表示 k 的值，由于在 f_{k0} 中关于 k 的有关部分计算已完成，因此当给定 u 的值 u0 时，$f_{k0}(u0)$ 的计算要比 f(k0,u0) 的计算快得多。

我们可以用一个比较通俗的例子来说明部分计值的思想，设 human 是含有两个参数 knowledge 和 problem 的程序，则从 human 和 knowledge 可得到专家 $human_{knowledge}$，即：

$$human_{knowledge}(problem)=human(knowledge,problem)$$

显然，当把问题限制在一个特殊的知识（knowledge）中，$human_{knowledge}$ 解决问题的速度要比一个一般的 human 解决的速度快得多。

进一步，我们考虑具有自作用的部分计值器，设部分计值器 α 是含有两个参数的程序，满足：

$$\alpha(f,k)=f_k \quad (2)$$

由(2)可导出：

$$\alpha(\alpha,k)=\alpha_k \quad (3)$$

$$\alpha(\alpha,\alpha)=\alpha_\alpha \quad (4)$$

用 α，由(2)式则专家 $human_{knowledge}$ 可由 human 和 knowledge 生成：

$$\alpha(human,knowledge)=human_{knowledge}$$

由此可见，α 可看作一个专家训练器，用同样的方法，我们可得到如下两个等式：

$$\alpha(\alpha,human)(knowledge)=\alpha_{human}(knowledge)=human_{knowledge}$$

$$\alpha(\alpha,\alpha)(human)=\alpha_\alpha(human)=\alpha_{human}$$

这两个等式分别意味着：α_{human} 是一个专家训练器，α_α 是一个训练器的生成器。

设给定的一阶函数程序有如下形式：

f1(s,d)=expression1

g(u,v,…)=expression2

…

h(r,s,…)=expressionm

这里，f1中的 s 为静态参数集，d 为动态参数集，g、…、h 为在 f1的表达式中要出现的函数，为方便起见，我们记 $g_{statvalues}\triangleq(g,statvalues)$，g 是在原程序中所定义的，statvalues 是 g 中所有静态参数构成的元组。

在部分计值器中，对一个给定的程序 f 和部分已知参数 s，得到其剩余程序 f_s 的算法[7]为：

1. Read 程序 Program 和静态参数集 s；

2. Pending：={f1.}；Seenbefore：={}；

3. While Pending≠{} do 4—6；

4. 从 Pending 中选择一元组 $g_{statvalues}$ 并在集合 Pending 中消去它，且把它添加到 Seenbefore 中；

5. 从程序中找出 g 的表达式：g(x1,x2,…)=g-expression，并记其所有动态参数为 D1,…,Dm；

6. 生成 g 的剩余程序部分（即在运行时计算的部分）：

$$g_{statvalues}(D1,\cdots,Dm)=Reduce(E);$$

把表达式 E 约简为剩余程序的算法如下（记 RE=Reduce(E)）：

1. 若 E 是 g 的常数或动态变量，则 RE=E；

2. 若 E 是 g 的静态变量，则 RE=从 statvalues 表中抽取此变量所对应的值；

3. 若 E=operator(E1,…,En)，则计算出 Reduce(E1),…,Reduce(En) 的值 $v_1,\cdots,v_n$，RE=operator($v_1,\cdots,v_n$)；

4. 若 E=operator(E1,…,En)，则计算 E1'=Reduce(E1),…,En'=Reduce(En)，RE=operator(E1',…,En')；

5. 若 E=if E0 then E1 else E2，则计算 Reduce(E0)，若 Reduce(E0)=true，则 RE=Reduce(E1)，否则，RE=Reduce(E2)；

6. 若 E=if E0 then E1 else E2，则 RE=E=if E0' then E1' else E2'，这里，Ei'=Reduce(Ei)；

7. 若 E=f(E1,E2,…,En)且程序中有 f(x1,x2,…,xn)=f-expression,则 RE=Reduce(E'),这里 E' 是 f 中的静态参数 xi 用相应的 Reduce(Ei)替换所得;

8. 若 E=f(E1,E2,…,En),则

①通过调用 Reduce,计算 f 中的静态参数元组;

②通过调用 Reduce,计算 f 中的动态参数,会得到一个表达式表;

③RE=call $f_{statvalues}$(Dynvalues);

④如果 $f_{statvalues}$即不在 Seenbefore 中又不在 Pending 中,则把它添加到 Pending 中去。

3. 可移动代码的优化

设在异构网络环境的服务器中有一代码 P:

```
P=f(n,x)=if n=0 then I
    else if even(n) then f(n/2,x)↑2
        else x*f(n-1,x)
```

若有静态输入 n=5,则对 P 中的各变量进行分析,暂时不能计算的表达式作下划线标记,可得:

```
P=f(n,x)=if n=0 then I
    else if even(n) then f(n/2, x)↑2
        else x*f(n-1,x)
```

用上节的部分计值算法则可得:

$P_5=f_5(x)=x*((x\uparrow 2)\uparrow 2)$

显然,当可移动代码 P 的部分参数为已知(或相对固定不变)时,迁移代码 P_5到客户机比迁移代码 P 到客户机所需的通讯量要少得多,且在客户机中的运行效率要高得多,而且这种经部分计值后的代码并没有损失其异构性。进一步地,由于在异构环境中,可移动代码在客户机中大都是解释执行的,因此其运行效率较低,我们可以用部分计值的自作用能力,在语义上生成其编译代码,这样可提高其运行速度,为了说明这,我们设 I 是一个程序语言的解释器,c^I、p、d 分别为 I-语言的编译器、程序、输入数据,则有:

$$c^I(p)(d)=I(p,d) \quad (5)$$

由(1)和(2)可得:

$$f(k,u)=\alpha(f,k)(u) \quad (6)$$

(6)式作代换{I/f,p/k,d/u}可得:

$$I(p,d)=\alpha(I,p)(d) \quad (7)$$

(6)式作代换{α/f,I/k,p/u}可得:

$$\alpha(I,p)=\alpha(\alpha,I)(p) \quad (8)$$

(6)式作代换{α/f,α/k,I/u}可得:

$$\alpha(\alpha,I)=\alpha(\alpha,\alpha)(I) \quad (9)$$

因此有:

$$c^I(p)(d)=I(p,d)=\alpha(I,p)(d)$$
$$=\alpha(\alpha,I)(p)(d)=\alpha(\alpha,\alpha)(I)(p)(d)$$

可见 α(α,I)就是 P 的编译器,即由解释器可生成语言的编译器,若一个程序在客户端要多次使用,显然,编译程序比解释程序的运行要快得多,可以极大地提高程序的运行速度。

结论 随着计算机网络的不断发展,可移动代码的优化问题将成为人们研究的热点之一,本文提出的部分计值技术可应用到可移动代码中,且在并行计算中若大部分输入数据为已知时,用部分计值技术也可对数据的分布进行优化,达到较好的负载平衡,但对不同的程序设计语言构造出其相应的部分计值器将是一个复杂而有趣的工作,这也是我们进一步的工作。

参考文献

1 Futamnura Y, Nogi K, Takano A. Essence of generalized partial computation. Theoretical Computer Science, 1991,90:61~79

2 Kelsey R, Rees J. A tractable scheme implementation Lisp and Symbolic Computation, 1995,17(4)

3 Outerhout J K. Tcl and the Tk Toolkit. Addison-Wesl, Reading, MA, 1994

4 White J E. Telescript Technology: the Foundation of the Electronic Marketplace. General Magic White Paper, General Magic Inc., Sunnyvale, CA, 1994

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

6 Jones N D, Gomard C K, Sestoft P. Partial Evaluation and Automatic Program Generation, Printice Hall, 1993

7 Jones N D. An introduction to partial evaluation. ACM Computing Surveys, 1996,28(3):480~503

(上接第8页)

7 Brown A W, Wallnau K C. Engineering of Component-Based Systems. Same to[1]

8 Brownsword L, et al. The Opportunities and Complexities of Applying Commercial-Off-the-Shelf Components. CROSSTALK. April 1998

9 Vidger M R, et al. COTS Software Integration: State-of-the-Art[online]. Available at: URL: http://wwwsel.iit.nrc.ca/abstracts/NRC39198.abs

10 Whitehead E J Jr, et al. Software Architecture: Foundation of a Software Component Marketplace. In: Garlan D, ed., Proc. of the First Intl. Workshop on Architectures for Software Systems. April 1995