部分计值综述*)

拜朝峰　林　琳　黄林鹏　孙永强
(上海交通大学计算机科学与工程系　上海 200030)

摘　要　部分计值是一种程序转换技术，在给定程序部分输入的情况下，可使用该技术对程序进行例化，完成程序中尽可能多的计算，最终得到高效的剩余代码。人们已经研究了许多程序设计语言的部分计值系统，并把它们应用到编译和编译器生成、计算机图形学等领域。本文介绍了部分计值理论及其应用，讨论了 Java 部分计值器的研究现状，并简单描述了本课题组设计的一个 Java 分布式部分计值系统 DJmix[1]。

关键词　部分计值，Java

Introduction to Partial Evaluation

BAI Chao-Feng　LIN Lin　HUANG Lin-Peng　SUN Yong-Qiang
(Department of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai 200030)

Abstract　Partial evaluation is a program transformation technique for specializing programs which finish the computation of programs under some known input as much as possible and produce more efficient residual programs. Partial evaluation has been studied in the context of a wide variety of programming languages and applied to a wide variety of areas that include compiling and compiler generation, computer graphics etc. In this paper, we survey the theory of partial evaluation and some application, and introduce the state of the study of partial evaluation of Java. We also describe our research work DJmix, a distributed partial evaluation of Java.

Keywords　Partial evaluation, Java

1　引言

部分计值，又称程序例化(特化)，是一种源程序到源程序的代码转换技术：在给定部分输入参数的情况下，对程序进行转换约简，完成程序中只依赖于给定参数的计算而产生剩余程序，当剩余程序作用在其余的参数时，产生与原来程序作用在全部参数一样的结果。作为代码转换研究领域的一个分支，它能够从程序运行环境和程序自身所包含的已知条件，自动对程序进行例化，生成效率更高的程序代码。

1.1　一个示例

考虑如图 1 计算 x 的 n 次幂的 Java 方法 pow(int n, long x)。给定值 n=5, x=2，该方法计算 pow(5,2)，得到结果 $2^5=32$。

```
public long pow(int n,long x){
    long result
    result=1;
    while(n>0){
      if(n%2==0){x=x*x;n=n/2;}
      else{result=result*x;n=n-1;}
    }
    return result;
}
```

图1　pow 方法

假设需要对于给定值 n=5和一系列不同的 x 值计算 pow(n,x)，(例如在一个循环中，x 是循环变量)。可以对方法 pow 按 n=5进行部分计值得到如图2所示剩余方法 pow_5(long x)，这时计算 pow_5(2)得到结果32。事实上，对于 x 的任意输入值，计算 pow_5(x)得出的结果与计算 pow(5,x)得出的结果都相同。在这里，因为 n 的取值在部分计值时是已知的，把 n 叫做静态的；因为 x 的取值是未知的而把 x 叫做动态的。从这个例子中，我们可以看到：由于原程序的控制流(这里，就是 while 和 if 语句)完全被静态参数所决定，在剩余程序 pow_5中，所有的判断语句和有关 n 的算术操作语句都被移除了。

在很多情况下，程序的控制流都是由静态参数决定的，通过部分计值就可以得到代码长度短、运行时间少的剩余程序。

```
public long pow_5(long x){
      log result;
      result=x;x=x*x;x=x*x;result=result*x;
      return result;
}
```

图2　pow_5方法

2　部分计值理论与实践

2.1　什么是部分计值

首先引入一些符号和定义。用 p 表示程序代码本身；用 $[\![p]\!]$ 表示 p 所计算的函数或在机器上运行的程序，有时为了明确表示 p 是用 L 语言编写的我们记 $p\in L$，此时 p 可在 L 语言的机器上运行；用 $[\![p]\!]_L$d 表示在 L 机器上当输入为 d 时运行 p 得到的结果。

下面我们来描述解释器、编译器和部分计值器。

定义1　一个用 L 语言编写的语言 S 的解释器 $Sint$，按照解释器的定义对于任意程序 s 和数据 d 要满足如下等式：

$[\![s]\!]$, d=$[\![Sint]\!]_L[s,d]$

定义2　一个用 L 语言编写的语言 S 的编译器 $STcomp$，

*)本课题的研究得到 863 高科技项目(项目编号：2001AA113160)资助。

生成的目标代码是 T 语言的，那么按照编译器的定义对于任意程序 p 要满足如下等式：

$[\![STcomp]\!]_L p = p'$ 并且 $[\![p']\!]_T d = [\![p]\!]_L d$ for all d

定义3 部分计值器也是一个程序，我们记做 *peval*，考虑一个程序 p 和它的输入 i=(s,d)，其中 s 是静态(已知)部分，d 是动态(未知)部分。*peval* 作用在 p 和 s 上，生成剩余程序 p_s，表示为：

$[\![peval]\!][p,s] = p_s$

根据部分计值的定义，*peval* 必须满足如下等式：

$[\![p_s]\!]d = [\![p]\!][s,d]$

这个等式就叫做部分计值等式。根据引言中的例子，可知道 p_s 的运行速度要比 p 快。

2.2 在线与离线部分计值

部分计值器分为两类：离线部分计值器和在线部分计值器。

离线部分计值包括两个阶段：预处理阶段和处理阶段。预处理阶段通常包括约束时间分析(BTA)。给定程序的约束时间标识(例如，哪些参数是静态的，哪些参数是动态的)，约束时间分析器将跟踪参数在程序中的传播和运算过程，判断每个表达式是在编译时就可以计算还是必须等到运行时才可以计算，并对原程序进行标注。然后，处理阶段利用得到的约束时间信息执行部分计值生成剩余程序。

一个在线部分计值器是一个非标准的解释器。每一个表达式的处理都是在例化时动态确定的。在线部分计值器在计算的每一步都要了解变量的相关信息，即变量值和变量状态(已知/未知)，并在每次给变量赋值时更新变量的信息，由此还可以在完成一条语句的分析后，确定下一条语句的例化结果。一般来说，在线部分计值比较精确，但缺点是解释开销较大。

3 部分计值研究现状

3.1 函数式语言的部分计值

因为函数式语言具有良好的代数基础，在部分计值过程中，任意提前或推迟某些运算不会影响整个程序的语义，部分计值的正确性比较容易证明，所以早期部分计值的研究工作主要集中于这些语言。

Weise 的部分计值器 Fuse 是一个在线部分计值器，目的在于把部分计值方法应用于实际应用，例如电路模拟等。Fuse 以图作为中间语言并且使用了提高剩余代码可重用性的策略。

Bondorf 的部分计值器 Similix[1]是一个离线系统，它基于一个固定的例化策略，并具有自作用能力。它也包括一个约束时间分析器。后来该系统被扩充为能够处理高阶函数和部分静态值。现在 Similix 的新版本基于 Henglein 的类型推导理论进行约束时间分析。

Consel 的部分计值器 Schism 也是一个离线系统，但它基于一个可变的例化策略，可以处理高阶函数和部分静态值。这个系统包括一个基于约束时间的编程环境。原程序和例化程序都用 Scheme 语言表示。

目前，关于 ML 语言，Scheme 语言的部分计值系统仍得到了广泛的研究，例如 Thiemann 在1998年提出了一个标准 Scheme 语言的例化系统。

3.2 逻辑语言的部分计值

最早的 Prolog 语言的部分计值由 Jan Komorowski 于20世纪80年代初提出，此后，许多人开始了此项目的研究工作。Fujita 和 Furukawa 设计了一个非常小的部分计值器，虽然他们宣称这个计值器是可以自作用的，但实际上当它自作用时无法保持操作语义的一致。第一个能有效自作用并语义安全的 Prolog 语言的部分计值器由 Bondorf 在一篇论文中提出，在这里原程序和值都用基项表示，同时使用约束时间标注达到自作用能力。目前，Prolog 语言的另一个部分计值器 Logimix[2]通过禁止值表示成基项改进了 Bondorf 的计值器，从而使效率得到了显著提高。

3.3 命令式语言的部分计值

第一批关于命令式语言部分计值的论文是由 Ershov 发表的，他使用 Algol-68语言的一个子集对部分计值做了探讨。此后陆续有一些论述命令式语言部分计值理论的论文发表。在这中间，最引人注目的是 Copenhagen 大学 Jones 研究小组研究并开发了基于 C 语言的部分计值器 C-mix[3]，它可以处理标准 C(ANSIC)的一个子集并具有自作用能力，但约束时间标注要由用户提供。此外，Uwe Meyer[4]开发了一个 Pascal 语言的子集的在线部分计值器，并对部分计值器如何构造和证明正确性做了论述。目前，命令式语言的部分计值仍然还是研究的热点，其中出现了 C 语言的另一个部分计值器 Tempo 和 Fortran 语言的部分计值系统。

3.4 面向对象语言的部分计值

面向对象语言是现在主流的程序设计语言，也成为了部分计值研究的一个新的领域，本节主要介绍 Java 语言的部分计值。

Java 语言是一种非常优秀的程序设计语言，它不依赖机器的体系结构和操作系统，具有可移植性。如今，许多公司都致力于改进 Java 编译系统，尤其是 Java 虚拟机的工作，以改善 Java 程序执行速度慢的缺点。即时编译器(JIT)和第三代 Java 编译系统 Hotspot 的出现对 Java 的性能有了很大的改善。

Java 语言的部分计值致力于提高字节代码的效率，是部分计值领域中的一个崭新的研究方向。由 Tokyo University 的 Masuhara 和 Yonezawa 提出的字节代码例化[5](BCS)系统以 Java 虚拟机语言(JVML)的子集 JVMLi 为计值对象，在 Stata 和 Abadi 的 Java 系统类型规则的基础上描述约束时间分析算法，部分计值后生成经例化的字节码程序。

Schultz、Lawall[6] 等人采用通过 Java-to-C 编译器 Harissa 将 Java 程序转换成 C 程序，然后用 C 语言的部分计值器 Tempo 计值的办法，来构造一个 Java 计值器。

Korea University 的 Jung Gyu Park 和 Myong-Soon Park[7]提出了一种新的 Java 程序分布式部分计值策略，把静态表达式再细分成多值静态和单值静态，然后把多值静态表达式编码为索引数据结构，把单值静态表达式编码为运行时代码，从而解决了例化代码指数式增长问题。

Djmix 是上海交通大学设计的一个面向 Java 字节代码的分布式部分计值器，研究工作包含四个部分：①方法的部分计值，②单个对象的部分计值，③本地多个对象的部分计值，④分布式部分计值。研究的目标是优化 Java 字节码，提高指令的执行效率，将部分计值技术和面向对象技术，分布式计算技术有机结合在一起，为 Java 语言的高效运行提供一个合理、高效的支持平台。

此外，C++作为最为广泛使用的面向对象语言也得到大量的研究，近几年部分计值技术还被应用于 C# 和 Aspect-

Oriented 程序设计语言。

3.5 分布式部分计值

随着工作站集群和计算机网络的迅猛发展，分布式计算技术也得到了巨大发展，程序在网络环境中并行执行效率更高，因此把分布式计算技术与部分计值技术结合起来也成为了一个新的研究方向。

Tubingen University 的 Michael Sperber 等人利用部分计值过程中固有的并行性加速程序的例化过程，在一个松耦合的工作站集群中使用 Farm 模式实现了一个 Scheme 语言的分布式部分计值系统，在拥有6个处理器的环境中得到了2～3倍的加速比。

进一步的工作也已经在 MPI、PVM、RPC 以及 RMI 的分布式部分计值方面展开。

4 应用

部分计值已经被应用于许多领域，包括编译与编译器自动生成，字符串与模式匹配，计算机图形学，数学计算，电路模拟和操作系统等。

4.1 编译与编译器自动生成

编译与编译器自动生成是部分计值应用的最主要领域。在这里首先介绍 Futamura 三映射。

4.1.1 Futamura 三映射 把部分计值应用于解释器就是通常所说的 Futamura 映射。Futamura 三映射是：

Futamura 第一映射：编译

$[\![peval]\!][interpreter, source]=target$

也就是说，通过把部分计值器作用到源程序 *source* 和它的解释器 *interpreter*，可以把源程序从一种语言编译到另一种语言，也就是完成了编译工作。

Futamura 第二映射：编译器生成

$[\![peval]\!][peval, interpreter]=compiler$

$[\![compiler]\!] source=target$

也就是说，把部分计值器作用到它自身，可以从解释器得到编译器，也就是完成了编译器的生成工作。

Futamura 第三映射：编译器生成器的生成

$[\![peval]\!][peval, peval]=compilergenerator$

$[\![compilergenerator]\!] interpreter=compiler$

本映射是说通过部分计值器的自作用，可以得到一个语言的编译器生成器，把这个编译器生成器作用到解释器就可以得到编译器，从而完成了编译器的自动生成。

4.1.2 应用实例 Sperber 和 Thiemann 把一个离线的一阶函数语言部分计值器应用到 Schemem 语言的编译过程，得到了一个 Scheme-to-C 编译器。

Jorgensen 描述了部分计值的一个大型应用，生成了一个强类型函数语言 BAWL 的编译器，并讨论了其中用到的部分计值技术，特别是为了得到短小高效的目标程序的约束时间分析技术。

编译与编译器生成是部分计值最主要的一个应用，得到了广泛的研究，但正如 Jorgensen 所说的，由于大部分程序设计语言的语义的复杂性，现在真正生成的编译器主要是针对小型语言的。

4.2 计算机图形学

光线跟踪也是研究得较多的一个应用，光线跟踪从屏幕上不同的像素沿着不同的方向反复计算光线穿过一个给定场景时的路径信息。在一个典型的光线跟踪应用中存在着数以百万的像素（当然就有百万计的光线），所以对于一个固定的场景来例化光线跟踪程序即使是渲染一个单一的图片也能达到一个可观的加速比。Mogensen 提出对于一个简单的光线跟踪程序加速比可以超过6倍。对于一些实用性的光线跟踪程序，加速比一般在1.5到3之间。

前面提到的 Jung Gyu Park 和 Myong-Soon Park[6] 已经将他们提出的分布式环境下的 Java 语言部分计值技术应用到一个可视化服装设计系统中，并得到了很好的效果。

4.3 数学计算

部分计值也被用于一些数学计算中。例如在一些模拟程序运行过程中，模型的一部分是固定的而另一部分是变化的，针对固定部分对程序进行例化就可以加速程序的运行。一个典型的例子是 N 体问题，就是模拟一些移动物体在重力作用下的相互作用，在模拟过程中，大部分物体的质量是固定的，而它们的位置和速度是变化的，针对物体的质量进行例化就可以加速模拟的速度。Berlin 就指出对于这个问题可以达到超过30倍的加速比。

在神经网络的训练过程中，要在一些测试事例上进行大量的计算，而许多参数是固定的，比如网络的拓扑结构，学习率。所以根据这些参数进行例化也可达一定的加速比，Jacobsen 指出可以得到25%到50%的加速比。

4.4 其他应用

本课题组将部分计值理论与信息个人化相结合，以 PIPE (Personalization is Partial Evaluation)理论为基础，结合机器学习技术并采用智能人机交互的 Agent 构架，以辅助人们进行信息检索。

另外，部分计值还被应用于软件体系结构、面向对象的软件工程以及硬件驱动程序的编写等领域。

结论 部分计值是一种程序转换技术，在给定原程序的部分输入的情况下，可使用该技术对程序进行例化，完成程序中只依赖于给定的那部分输入的计算，最终得到高效的剩余程序。由于其思想非常简单，得到了广泛的研究，并被应用于许多领域，如今随着网络的迅猛发展，部分计值技术已经和分布式计算技术相结合，对并行语言的分布式研究也已经开始进行，部分计值技术将会得到更广泛的应用。

参考文献

1 Bondorf A, Danvy O. Automatic autoprojection of recursive equations with global variables and abstract data types. Science of Computer Programming, 1991, 16: 151～195

2 Mogensen T, Bondorf A. Logimix: A self-applicable partial evaluator for Prolog, In LOPSTR'92, Workshops in Computing, K. K. Lau and T. Clement, Springer-Verlag, Berlin, Jan. 1993

3 Anderson L O. C program specialization: [Master's Thesis]. DIKU, University of Copenhagen, Denmark, DIKU Student Project 91-12-17, 1991

4 Meyer U. Correctness of on-line partial evaluation for a Pascal-like language. Science of Computer Programming, 1999, 34: 55～73

5 Masuhara H, Yonezawa A. Generating Optimized Residual Code in Run-Time Specialization: [TR-CS-1999]. University of Tokyo, 1999

6 Schultz U P, Lawall J L, Consel C, Muller G. Towards Automatic Specialization of Java Programs, IRISA CAMPUS UNIVERSITY DE BEAULIEU, 35042 RENNES CEDEX, FRANCE ISSN: 1166-8687

7 Park J G, Park M-S. Specializing Java Programs in a Distributed Environment. Journal of Information Science and Engineering, 2002, 18: 787～801