计算机科学 1996 Vol. 23 №. 4

IPM——一种增量计算的自动生成模型*)

廖湖声
(北京工业大学计算机学院　北京100044)

摘　要　IPM provides a general incremental execution framework for application programs. In the framework, non-incremental programs can be executed incrementally. Some techniques including the partial evaluation, program mergence and function catching, which are exploited for handling any input change to a program, are introduced in the paper.

关键词　Incremental Computation, Partial Evaluation, Program Synthesise.

一、引　言

增量计算在排版系统、编译系统和各种软件工具中有着广泛的应用，而实现一个增量式软件系统往往需要采用复杂的算法设计和艰苦的程序调试。近年来，随着部分求值技术的发展，人们开始研究用于增量计算的通用的程序生成方法，以求从根本上消除使用增量计算实现算法的需求。

有关增量计算自动生成的研究工作，大致可分为两类。一类以耶鲁大学的 Hudak 研究组为代表[1]，力图建立一种完成增量计算的程序执行环境。在这种环境中，应用程序的每次执行将能够利用上次执行的部分计算结果。另一类以康乃尔大学的 Teitelbaum 研究组为代表，试图将应用程序变换成采用增量式计算的程序[2]。两种方法的共同特点是采用了部分求值和函数收集(function catching)等技术，以求提高增量处理的效果。

本文为增量计算的自动生成提出了一种新的通用模型——IPM 模型，为建立一种函数式程序的增量式执行环境提供了基本框架。在这种框架中，应用程序能够按照输入的各种变化调整计算结果，进行增量计算。我们已经实现了一个基于 IPM 模型的增量计算执行环境的原型系统。

二、函数式语言 FL

IPM 模型的处理对象 FL 语言是一种一阶函数式语言，该语言采用积极求值方式，并具有流式的输入输出功能[3]，其语法如下：

语法域：

prog∈Prog	FL 程序
exprs∈Exps	表达式组
expr∈Exp	程序表达式
decl∈Decl	定义表达式
iden∈Idn	标识符
const∈Bas	常数

语法规则：

prog ::＝exprs 表达式组 'exprs WHEREREC decl…函数定义

decl ::＝iden…' ＝' exprs 定义表达式

exprs ::＝expr 简单表达式|READ iden;exprs 输入表达式|WRITE expr 输出表达式'IF expr exprs exprs 条件表达式

expr ::＝iden 变元标识符|const 常数'OP expr 单目运算|expr OP expr 双目运算|iden expr … 函数调用式'(expr)

其中，输入表达式用于从输入流读一数据后进行后续表达式的计算。

在 IPM 模型中，FL 程序的首次执行要求读取所有输入数据，而再次执行时仅要求指明输入的变化，如输入数据的更新、增加和减少等。IPM 模型能够按照输入变化，产生更新后的输出结果。

三、部分求值与程序归并

建立增量计算执行环境的基本设想是力图保存每次程序执行的中间结果，程序再次执行时可直接加以利用，从而避免重复计算，实现增量式处理。这种方法的关键在于选择中间结果的表示形式及其收集与利用的方法。

3.1　IPM 模型中的程序部分求值

鉴于部分求值等相关技术的发展，有人[1]提出以部分求值的结果——滞留程序(residual program)

*)本课题得到国家自然科学基金及北京市自然科学基金的资助。

的形式保存中间结果。程序的部分求值是在给定部分输入的条件下完成与给定输入相关的部分计算，已完成计算的结果自然包含于滞留程序中。

IPM 模型中采用了这种方法来处理输入数据值的变化，要求将所有输入划分为 N 个组(N>1)，在程序首次执行时针对每个输入组分别进行程序的部分求值，得到 N 个滞留程序。在某输入发生变化时，程序再次执行仅按照该输入所在的输入组进行一次程序部分求值，更新相应的滞留程序。随后，对 N 个滞留程序进行 N−1次归并(program mergence)，得到更新后的计算结果。在这种计算过程中，保存在其余 N−1个滞留程序中的部分计算结果得到直接使用，从而实现了增量计算。

一个用于计算一组整数的阶乘之和的 FL 程序如下：

```
read x (SumFac (Fac x))
whererec
  SumFac x=
    read o;
    if o==',' then
      read y;
      SumFac x+(Fac y)
    else
      write x
  Fac x=
    if x<2 then
      1
    else
      x * (Fac x−1)
```

该程序执行要求输入一组整数，以逗号分割，以分号表示输入结束，如果将输入分为前三个一组和其余输入为另一组，则程序首次执行将对于输入"2，3，4；"进行程序部分求值产生两个滞留程序，相对于输入2、','、3的滞留程序为：

```
if (input 4)==',' then
  write 8+(Fac (input 5))
else
  write 8
```

其中，输入式 Input i 表示第 i 个输入数据，该表达式是在输入表达式的部分求值中未得到输入值的情况下产生的。程序中的整数8是相对于输入组的部分计算结果2!+3!。

相对于其他输入的滞留程序为：

```
if (input 2)==',' then
  write (Fac (input 1))
    +(Fac (input 3))+24
else
  write (Fac (input 1))
```

程序中也包含了4!的计算结果。两个滞留程序的归并将得到程序：

```
write 32;
```

程序归并中直接利用了包含在各滞留程序中的部分计算结果。所有滞留程序归并后，都会得到这种完全由输出表达式组成的程序。执行该程序产生程序最终的计算结果。

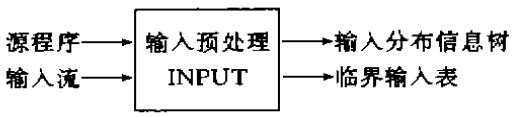
如果程序再次执行时某输入发生变化，将只有该输入相关的滞留程序被重新生成。随后，经过程序归并得到结果的输出程序，从而使保留在另一个滞留程序中的部分计算结果得到再次利用。

这种部分求值和程序归并机制是 IPM 模型响应输入数据值变化的基础。然而，IPM 模型使用的部分求值技术和传统的部分求值有相当大的差别。这种差别来自归并滞留程序的需求和程序设计语言的差别。

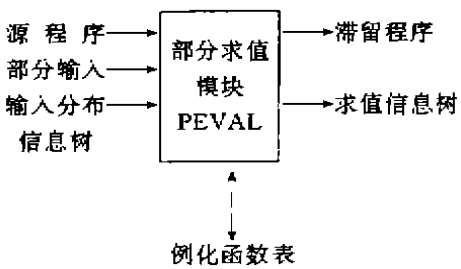
3.2 IPM 模型的输入分布信息

IPM 模型中 FL 语言具有的输入输出流功能增加了部分求值的复杂性。例如，在部分求值过程中，对于作为滞留程序组成部分保留的某表达式，如果其计算中包含输入，则无法确定该表达式的计算中将使用多少个输入数据，从而无法确定后续的其他表达式实际使用哪些输入。

为此，IPM 模型中设置了一个输入预处理模块，用于统计程序执行中各表达式读输入流的次数，形成一个输入分布信息树，并且找出所有临界输入。所谓临界输入是指其数值变化会改变输入分布的输入数据。如程序例中的第2、4、6输入。这些输入变化时将另行处理。



IPM 模型的程序部分求值就是针对部分输入，参照输入分布信息树，对源程序进行部分求值。在部分求值过程中，对于程序中的函数调用表达式，在实在参数给定时，进行求值；在实在参数部分给定时，则进行部分求值，形成函数的例化(specialization)。例化生成的新函数中，包含了该函数调用的部分结果。这种例化函数及其函数的部分求值模式被保存在例化函数表中，在每个函数调用表达式的部分求值之前，将检查该表中有无相同模式的可利用的例化函数，这种安排不仅避免了部分求值的不终止问题，并且为增量计算提供了更多的可直接使用的中间结果。

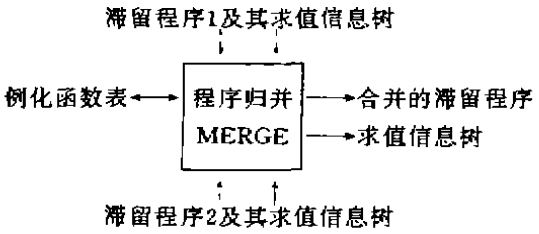


在 IPM 模型中，部分求值的结果将用于程序归并的处理对象，而程序归并的完成要求跟踪滞留程

序的构造过程。为此，模型中采用信息树的形式表示了整个部分求值过程。每个程序部分求值的同时，均构造了相应的求值信息树。

3.3 IPM 模型中的滞留程序归并

在 IPM 模型中，程序归并模块用于将两个滞留程序合并为一个滞留程序。对于分别对应于两个输入组的滞留程序，归并得到的滞留程序相当于合并两组输入进行部分求值的结果。如以上例题所示，在所有输入组相关的滞留程序被合并后，将生成完全由输出表达式组成的程序。



在程序归并中，也将产生新的例化函数，为增量计算的实现提供新的部分结果。

总之，IPM 模型采用的部分求值和程序归并方法要求程序的首次执行调用输入预处理模块获取输入分布信息，并对程序进行 N 次部分求值后进行 N－1次程序归并；对于在输入数据值变化时程序的再次执行，将使用一次部分求值和 N－1次程序归并。因此，程序执行的效率将主要取决于部分求值和程序归并的效率。

四、IPM 模型

上述部分求值和程序归并方法能够有效地响应输入数据值的变化，但是对于一般的应用系统，输入变化不仅包括输入数据值的变化，也包括输入数据的增加和减少。一个完整的增量计算生成模型应该能够响应输入数据的任意插入和删除，应该允许应用程序动态地确定输入个数。

IPM 模型设计中考虑了对用户插入或删除输入数据时的响应，并采用了和程序首次执行基本相同的处理方法。完整的 IPM 模型如图1所示。

在图1中，在输入增加和输入减少的情况下，处理过程与首次执行完全相同。但是，由于在例化函数表中保存了过去的部分计算结果，本次的部分求值和程序归并仍可以有选择地加以利用，从而达到增量式处理的效果。

例如，对于上节的程序部分求值，在例化函数表中除了滞留函数以外，保存了如下函数及其部分求值模式：

```
SumFac(x)[DD,4=]=
  if (input 2)==',' then
    write x+(Fac (input 3))+24
  else
    write x          (I-1)
SumFac(x)[,4=]=
    write x+24       (I-2)
Fac(2)[ ] =2         (I-3)
Fac(3)[ ] =6         (I-4)
Fac(4)[ ] =24        (I-5)
```

例化函数表中，圆括号中表示部分求值时参数的取值，方括号中表示了函数调用中使用的各输入值。部分求值中未确定的输入用符号 D 表示。例化函数表以这种形式保存了函数调用的部分求值模式，说明了函数在哪些参数和输入被给定，哪些参数和输入未给定的情况下，被处理到何种形式，从而为再次利用提供了依据。例如，例化函数(I-2)表示了函数 SumFac 在参数 x 未给定，其求值使用了三

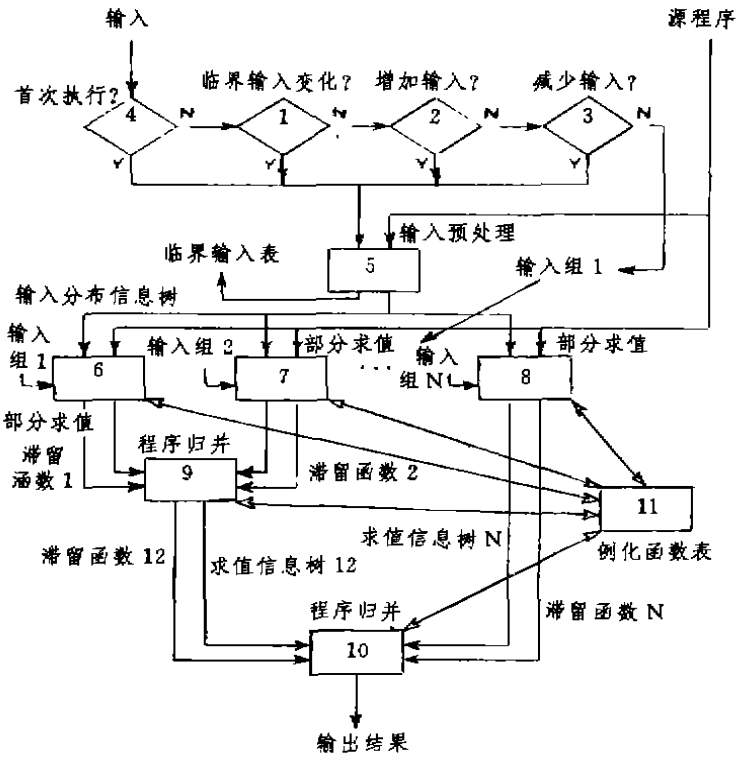


图1 IPM 模型

个给定输入值时的计算结果。在增量计算中，如果函数 SumFac 被用于输入流中三个输入，只要输入值相符，就可以利用该例化函数。

在输入数据改变时，例如输入数据2被改为5，则在随后的部分求值和程序归并中，(I-1)、(I-4)等例化函数将得到直接使用。在输入数据增加时，例如在第一个输入之前加入两个输入(7和逗号)，则部分求值中仍会出现与(I-1)、(I-3)、(I-4)相同的函数部分求值模式，可以直接利用例化函数表中的结果。IPM 模型正是利用例化函数表的这种函数收集作用(catching)去支持增量计算的完成。

另一方面，IPM 模型对于临界输入的变化，也采用了与首次执行相同的处理方法。因为这种变化将改变输入信息树的分布，其处理同样利用例化函数表完成增量计算。

五、问题与发展方向

IPM 模型存在与文[1]相同的输入分组问题。由于部分求值是按照输入组进行的，输入分组的方法直接影响增量计算的指标——增量度(incrementality)。如何根据程序结构进行输入分组，能否实现输入分组的自动生成，如何根据输入变化调整输入分组，都需要进行研究。

增量度的提高同样需要改进程序的部分求值技术。滞留程序归并为程序的部分求值提出了新的要求。发展满足这种需求的联编时间分析(*binding time analysis*)和静态解释技术是提高部分求值的效率，发展 IPM 模型的需要。

在 IPM 模型中，程序归并的实现方法是提高增量度的关键。由于每次执行都使用 N－1次程序归并，效率低的程序归并方法可能完全抵消了直接利用部分计算结果所带来的好处。另一方面，求值信息树也可能占用过多的内存空间。因此，优化程序归并实现算法和简化求值信息树都是研究课题。

对于增量计算的自动生成，程序设计语言的选择在相当大的程度上决定了问题的复杂性。功能强的语言将增加部分求值和程序归并的实现难度、执行效率和系统规模。然而，实用化要求使用通用性强的语言，IPM 模型选择具有输入输出流的语言也是向这个方向的努力。

IPM 模型发展为成熟的软件技术乃至实用的软件工具，还需要研究输入分组的方法，进一步提高部分求值和程序归并的效率，并且发展通用性较强的程序设计语言的部分求值和程序归并技术。

主要参考文献

[1]Sundaresh R. S.，Building incremental programs using partial evaluation，Proc. of the Symp. on PEPM，Yale University，1991.

[2]Liu Y. A. et al.，Systematic derivation of incremental programs，Science of Computer Programming，24(1)，1995.

[3]廖湖声等，函数式程序的离散式延迟输入输出流，18(7)，1995

(上接第80页)

$$\mu'_{min}=\frac{(\mu-1)T_p-2|\triangle^m_{max}-\triangle^m_{min}|}{T_p}+1$$

$$\mu-\mu'=\frac{2(\triangle^m_{max}-\triangle^m_{min})}{T_p} \quad (8)$$

四、结束语

多媒体低层同步问题是多媒体播放系统中必须研究的问题，然而又是一个相当复杂的课题，而在分布式多媒体数据库系统中，由于要考虑网络环境的延时，更增加了问题的难度。本文在分析网络延迟的基础上，对连续媒体的平滑播放进行了研究，给出了播放算法中预取媒体块数的计算方法，以及播放过程中所需缓冲区的计算方法。对于不同媒体的同步问题，我们将在以后做进一步的探讨。

参考文献

[1]Thomas D. C. Little 等，Interval-Based Conceptual Models for Time-Dependent Multimedia Data，*IEEE Trans. on Know. and Data Eng.*，Vol. 5，No. 4，1993

[2]S. Ramanathan 等，Feedback Techniques for Intra-Media Continuity and Inter-Media Synchronization in Distributed Multimedia Systems，The Computer J.，Vol. 36，No. 1. 1993

[3]G. F. Zhao & K. M. Lye，Synchronization protocol for high quality and Single-point playout multimedia applications，Computer Comm.，Vol. 17，No. 9，1994

[4]G. J. Lu 等，Temporal synchronization support for distributed multimedia information systems，同[3]，Vol. 17，No. 12，1994

[5]孙铭、胡修林，多媒体对象同步模型的分析和应用，第十二届全国数据库学本会议论文集，1994年10月