第 32 卷　第 14 期　　计　算　机　工　程　　2006 年 7 月
Vol.32　№ 14　　Computer Engineering　　July 2006
·软件技术与数据库·　文章编号：1000—3428(2006)14—0054—03　文献标识码：A　中图分类号：TP277

一种基于抽象解释的 WCET 自动分析工具

姬孟洛[1]，李　军[2]，王　馨[1]，齐治昌[1]
(1. 国防科学技术大学计算机科学学院，长沙 410073；2. 北京跟踪与通信技术研究所，北京 100094)

摘　要：利用基于抽象解释的变量值范围传播技术，提出了一种自动分析高级语言程序流信息的方法；并在白盒测试工具 NPCA 的基础上，利用该方法实现了 WCET 分析工具 NPCA-WCET。
关键词：实时程序；WCET 分析工具；抽象解释

An Automatic WCET Analysis Tool Based on Abstract Interpretation

JI Mengluo[1], LI Jun[2], WANG Xin[1], QI Zhichang[1]
(1. School of Computer Science, National University of Defense Technology, Changsha 410073;
2. Beijing Institute of Trace and Telecommunication Technology, Beijing 100094)

【Abstract】 This paper presents a worst-case execution time (WCET) analysis method where its automatic program flow analysis method is based on value range propagation method supported by abstract interpretation. The WCET analysis method is for modern RISC processor including pipelining and caching. Based on this method, the paper implements a WCET analysis tool——NPCA-WCET.
【Key words】 Real-time programs; Worst-case execution time (WCET) analysis tool; Abstract interpretation

与非实时系统不同，实时系统的正确性不仅取决于它所产生的输出，而且还取决于产生输出的时间。实时系统的计算结果只有在规定的时间范围内到达才是有效的。

分析实时系统时间行为的经典方法是调度分析。调度分析程序根据每个任务的执行时间按照一定的调度策略确定每个任务能否在规定的时间完成。这里每个任务的执行时间指的是最差情况下的执行时间(Worst-Case Execution Time，WCET)，也就是说，不管输入数据如何，调度都按照这个时间考虑。这个时间要求必须在调度之前事先获知。

WCET 分析不仅用于调度(如最小时间优先)和调度检测，还用于确定周期性任务是否满足其性能目标，以发现系统的性能瓶颈。WCET 分析是实时系统的一个重要研究领域。

获得 WCET 估值的常用方法是静态分析。静态分析方法计算应用程序代码片段的执行时间上界，这里代码片段的执行时间定义为执行代码片段占用的处理器时间。

WCET 分析值必须安全(safety)和精确(tightness)，安全保证不能低估最差执行时间，精确要求提供可接受的高估值。精确的 WCET 估算值能够节省资源。

本文利用基于抽象解释[1]的变量值范围传播技术，提出了一种自动分析高级语言(比如C语言)程序流信息的方法；在该技术的基础上，针对现代RISC处理器结构通常具有的流水线和高速缓存特征计算程序的WCET。

利用上述 WCET 分析技术，我们扩展了白盒测试工具 NPCA 的功能，使其支持实时程序的 WCET 分析，并将该工具命名为 NPCA-WCET。NPCA 是针对 C++语言开发的白盒测试工具，在解放军总装备部内具有广泛的应用。

1 WCET 分析简述

1.1 程序流分析

WCET 分析通常包括 3 部分：程序流分析，处理器特征分析和计算，如图 1 所示。

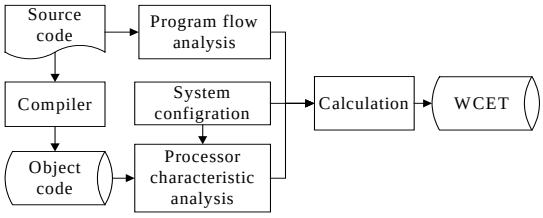

图 1　WCET 分析的组成

为了得到精确的 WCET 分析值，需要获得程序流信息。流信息通常包括循环的迭代次数范围、不可行路径(infeasible path)、递归调用的最大深度等程序流约束信息。

获得程序流信息的方法包括手工标注和自动分析。手工标注是用户通过和 WCET 工具交互，给源程序代码添加标注(annotation)，或者在程序代码外面给出需要的信息。手工标注给程序员增加了额外的工作负担，且容易出错。自动流分析则根据程序语法和语义信息自动获取程序的流信息。

1.2 处理器特征分析

要获取程序的WCET，需要根据处理器的特征对程序的目标代码进行分析，以获得每一条指令、每一个基本块实际的时间行为。对于不带cache、没有流水线的简单CISC指令，此分析就非常简单：一个代码段的执行时间就是其所对应的每条指令执行时间的累加。但对于带有高速缓存和流水线等特征的现代处理器，代码段执行时间的估算[2,3]就复杂得多。

1.3 计算

计算的目的是，在给定程序流和低层分析结果的情况下，为程序估算 WCET。WCET 计算方法分为 3 类：基于路径，基于树和隐含路径枚举 IPET。在基于树(tree-base)的计算中，

基金项目：国家自然科学基金资助项目(60303013)
作者简介：姬孟洛(1963—)，男，博士、高工，主研方向：实时系统分析，面向对象设计；李　军，高工；王　馨，博士；齐治昌，博导
收稿日期：2005-07-31　**E-mail：**justin.ruan@163.com

使用为每种类型的复合程序语句定义的规则确定语句的WCET，然后通过自底向上遍历对应于程序的语法分析树产生 WCET 估值。IPET(Implicit Path Enumeration Technique)方法根据程序的语法和语义约束，利用整数线性规划求整个程序 WCET。

在基于路径(path-based)的计算中[2,3]，通过计算程序中不同路径的时间，查找其中具有最长执行时间的路径，然后计算WCET估值。其对应于循环的计算公式可表示为

$$WCET\_loop\_time = WCET\_path\_time * n \quad (1)$$

其中，WCET_*path_time* 是循环中最长路径的执行时间，*n* 为循环的迭代次数。

2 基于抽象解释的程序流分析

在抽象解释[1]和基于通用的单调数据流框架[4]的基础上，给出自动获取变量的值范围的方法。根据变量的值范围，提出了一种自动计算循环最大迭代次数及标识循环中的不可行路径的计算方法。

2.1 基于抽象解释的值范围分析

称程序中变量值改变的位置为控制点，把该点所有变量的可能取值称为环境，即

$$\sigma_i^h = \{a_1 \text{ a } v_1, ..., a_m \text{ a } v_m\}$$

h 表示处于循环之中第 *h* 次迭代时的环境。

具体语义给出的是程序产生环境的规则，也就是赋值和条件改变环境的规则。抽象环境 σ 存放的是变量与其抽象值的结合，这里的抽象值是变量的值范围。比如说 $b \mapsto 2..4$ 的意思是 b 在 2 到 4 之间。变量的初始值范围可以通过输入规格说明获得。

完备格描述一种具有偏序关系集合的代数系统。我们把抽象环境定义为完备格 $State_{\text{Int}} = \left((Var \to Interval)_\perp, \dot{\sigma}\right)$，其中 *Var* 表示程序变量集合，*Interval* 表示值范围集合。值范围 $\left(Interval, \dot{\sigma}\right)$ 也构成完备格。完备格 $State_{\text{Int}} = (Var \to Interval)_\perp$ 上的偏序 $\dot{\sigma}$ 定义为

$$\forall \sigma \in (Var \to Interval)_\perp : \perp \dot{\sigma} \sigma$$

$$\forall \sigma_1, \sigma_2 \in Var \to Interval :$$

$$\sigma_1 \dot{\sigma} \sigma_2 \text{ iff } \forall x : \sigma_1(x) \dot{\sigma} \sigma_2(x)$$

相应地，定义其上确界和下确界操作为

$$\forall \sigma \in (Var \to Interval)_\perp : \perp \cup \sigma = \sigma$$

$$\forall \sigma_1, \sigma_2 \in Var \to Interval :$$

$$\forall x : (\sigma_1 \cup \sigma_2)(x) = \sigma_1(x) \cup \sigma_2(x)$$

$$\forall \sigma_1, \sigma_2 \in Var \to Interval :$$

$$\forall x : (\sigma_1 \cap \sigma_2)(x) = \sigma_1(x) \cap \sigma_2(x)$$

通过定义语句或条件表达式的迁移函数 f^{Int} 传播变量的值范围，比如对赋值、复合和条件表达式，有

$$[x := a] : f^{\text{Int}}(\sigma) = \begin{cases} \perp & \text{if } \sigma = \perp \\ \sigma(x \text{ a } A_{\text{Int}} a \sigma) & otherwise \end{cases}$$

$$[S1; S2] : f^{\text{Int}}(\sigma) = f_{S2}^{\text{Int}}(f_{S1}^{\text{Int}}(\sigma))$$

$[b]^{T,F} : f_T^{\text{Int}}(\sigma) = \sigma \cap \sigma_b$ 和 $f_F^{\text{Int}}(\sigma) = \sigma \cap \sigma_{\neg b}$

其中，括号“ ”中包含的是语法成分，f_{S1}^{Int}、f_{S2}^{Int} 分别为对应于 S1 和 S2 的迁移函数；f_T^{Int}、f_F^{Int} 分别为对应于布尔表达式[b]TRUE 分支和 FALSE 的迁移函数。σ_b 和 $\sigma_{\neg b}$ 是由布尔表达式[b]生成的值范围状态，满足 $B_{\text{Int}} b \hat{\sigma}_b = true$ 和 $B_{\text{Int}} b \hat{\sigma}_{\neg b} = false$。而

$$\sigma(x \text{ a } A_{\text{Int}} a \sigma)(y) = \begin{cases} \sigma(y) & \text{if } y \neq x \\ A_{\text{Int}} a \sigma & \text{if } y = x \end{cases}$$

这里迁移函数使用了函数 $A_{\text{Int}} : AExp \to (State_{Int} \to Interval)$ 来分析算术表达式，使用了函数 $B_{\text{Int}} : BExp \to (State_{Int} \to Bool)$ 来分析布尔表达式，其中 *AExp* 和 *BExp* 分别是算术表达式和布尔表达式集合，$State_{Int}$ 是抽象环境集合，*Interval* 和 *Bool* 分别是值范围和布尔值集合。

利用通用的单调数据流框架实例[4]，根据程序的语法结构可以建立每个控制点的等式公式。如对于带标号程序例 1：

[x:=0][1]; while ([x<100][2,3]) {

if ([x>100][4,5]) [y:=1][6]; else [y:=2][7];

[x:=x+2][8]};

标号和方括号是为了标出程序的控制点。而对于控制点 8，有

$$\sigma_\circ(8) = \sigma_\bullet(6) \sqcup \sigma_\bullet(7)$$

$$\sigma_\bullet(8) = f_8(\sigma_\circ(8)) = \sigma_\circ(8)[x \text{ a } A[x+1]\sigma_\circ(8)]$$

其中，$\sigma_\circ$ 和 $\sigma_\bullet$ 分别表示控制点在实施迁移之前和之后的抽象环境。

根据每个控制点的等式公式，通过迭代过程[4]，求出每个控制点稳定的抽象环境(不动点)，即得到每个变量在每个控制点的值范围。比如稳定之后控制点 2 的变量值范围为：

$$\sigma_\circ(2) = \{x \text{ a } [0,100]; y \text{ a } [1,2]\}$$

$$\sigma_\bullet(2) = \{x \text{ a } [0,99]; y \text{ a } [1,2]\}$$

2.2 循环迭代次数的计算

在循环 *L* 中，若变量 *i* 值的每次改变都增加或减少某个固定的常量，那么 *i* 称为循环 *L* 的归纳变量。归纳变量 *i* 唯一，形如：$i := i \pm c$ 的赋值，其中 *c* 是常量。这里称常量 $\pm c$ 为归纳变量 *i* 的跨度。这里假定所有的循环都有归纳变量。

根据归纳变量的值范围并不总能够求出循环的界限。举例来说，假定 *i* 是循环 while (*i* <*100*) do S 的循环变量，如果 *i* 的初值为 0，*i* 的跨度 *c* 为 3，则无法根据 *i* 的值范围求出循环的界限。因此，这里还假定根据归纳变量的值范围总能够求出循环的界限。

需要说明的是，一般的实时系统开发语言都强制要求循环具有归纳变量，且根据归纳变量能够求出循环的界限。

当迭代过程稳定之后，可以根据归纳变量的值范围和跨度计算循环的界限。比如对 while 语句，假定其 TRUE 分支的值范围为 $\sigma \in (Var \to Interval)_\perp$。根据归纳变量的值范围及其跨度，可知 while 循环的迭代次数为

$$loopCout := \min_{j \in InducVar} \left(\left\lfloor \frac{\sup(j) - \inf(j)}{|stride(j)|} \right\rfloor + 1 \right) \quad (2)$$

其中 *InducVar* 为循环的归纳变量集合。

对例子 1 的程序，while 循环的迭代次数为 $\left(\lfloor (99-0)/2 \rfloor + 1\right) = 50$。

2.3 标识不可行路径

一条路径 *p* 是一个控制点序列 $p = [l_1, ..., l_n]$，这里 l_i 为控制点，$i = 1, ..., n$。路径 *p* 的迁移函数 f_p^{Int} 定义为

$$f_p^{\text{Int}}(\hat{\sigma}) = \begin{cases} \hat{\sigma} & \text{if } p = \varepsilon \\ f_{p'}^{\text{Int}}(\hat{\sigma}') & \text{if } p = [l_1, ..., l_n] \end{cases}$$

其中，$p' = [l_2, ..., l_n]$，$\hat{\sigma}' = f_{l1}^{\text{Int}}(\hat{\sigma})$。

循环路径是循环中由控制流迁移连接的基本块序列，该

路径由循环头开始。比如对 while $[b]^{T,F}$ do S 语句中的循环路径 $p=[l_1,...,l_n]$，则有 l_1 对应于控制点 T，l_n 对应 S 语句的最后一个标号。对于循环路径 $p=[l_1,...,l_n]$，当迭代过程完成之后，则显然有下列命题成立：

假定 $\hat{\sigma}$ 为对应于 l_1 的抽象环境，$\hat{\sigma}_p = f_p^{\mathrm{Int}}(\hat{\sigma})$。如果 $\exists v \in Var$ 使得 $\hat{\sigma}_p(v) = \emptyset$，则 p 为不可行路径。比如，对于路径 p=[2,4,6,8]，有

$$f_p^{\mathrm{Int}}(\hat{\sigma}_\circ(2)) = f_8(f_6(f_4(f_2(\hat{\sigma}_\circ(2))))) = \{x \text{ a } \emptyset; y \text{ a } [1,1]\}$$

说明 p 为不可行路径。

3 NPCA-WCET 的设计

在测试工具 NPCA 的基础上，我们实现了基于抽象解释的 WCET 自动分析工具 NPCA-WCET。NPCA 具有词法和语法分析功能，能够获取程序的语法结构和调用关系。NPCA 的主要目标是测量程序的函数、语句和分支覆盖。NPCA-WCET 的逻辑结构如图 2 所示。

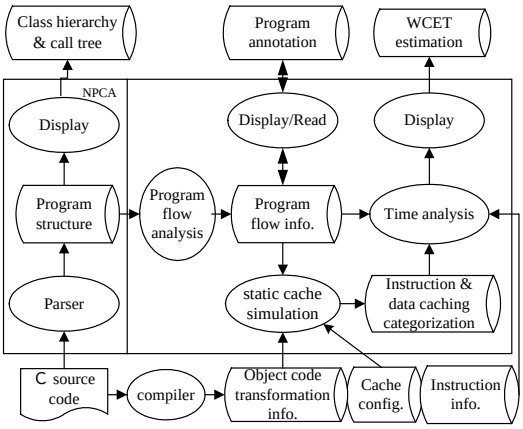

图 2　NPCA-WCET 的逻辑结构

3.1 词法、语法分析

NPCA 对 C 源程序进行词法和语法分析，得到程序的结构信息，包括语句间的关系、函数的调用关系。对于 C++还能够得到类层次图。这些信息通过 GUI 显示。

3.2 循环标记

程序流分析采用第 2 节的方法对程序结构信息进行分析。NPCA-WCET 还允许用户通过 GUI 输入程序流信息，主要包括循环界限和不可行路径，如图 3 所示。

图 3 是NPCA-WCET的一个显示界面示意图，显示的是程序Sumnegpos[4]的控制流。把Sumnegpos的循环改为不能界定时，NPCA-WCET将在对应该循环体的分支上显示“?”，比如在图 3 节点 1 之TRUE分支的位置显示“?”。此时允许在该位置输入该循环体的界限。

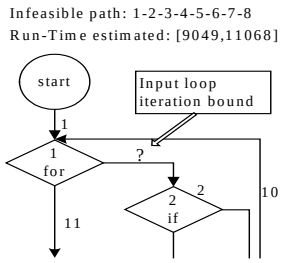

图 3　NPCA-WCET 显示界面示意

3.3 时间估算

按照和文献[2,3]类似的方法，静态缓存模拟根据目标代码和缓存的配置，利用程序流信息以及目标代码和源程序的对应关系，对每一条指令和数据访问进行分类。时间分析程序利用厂商提供的指令特征信息产生指令的流水信息，然后根据指令和数据访问的缓存类别[2]，考虑到路径的第 1 次执行与后续执行的差异，使用式(1)计算循环和函数的WCET。

举例来说，对于 SPARC 结构，其缓存配置为 8 行，每行 16 字节直接映射指令缓存，则程序 Sumnegpos 运行时间估值为[9049,11068]，如图 3 所示。

4 结语及未来工作

本文提出了一种 WCET 分析技术，描述了基于该技术的分析工具 NPCA-WCET。NPCA-WCET 针对的是高级语言和现代 RISC 体系结构(如流水线和高速缓存)，其特点在于利用基于抽象解释的变量值范围传播技术，自动获取程序的流信息，同时还允许用户通过 GUI 标注程序的流信息。

参考文献

1 Cousot P, Cousot R. Abstract Interpretation: A Unified Model for Static Analysis of Programs by Construction or Approximation of Fixpoints[C]. Proceedings of the 4th ACM Symposium on Principles of Programming Languages, Los Angeles, 1977: 238-252.

2 Arnold R D, Mueller F, Whalley D, et al. Bounding Worst-case Instruction Cache Performance[C]. Proceedings of the 15th Real-time Systems Symposium, Massachusetts, 1994: 172-181.

3 Healy C A, Arnold R D, Mueller F, et al. Bounding Pipeline and Instruction Cache Performance[J]. IEEE Transactions on Computers, 1999, 48(1): 53-70.

4 Nielson F, Nielson H R, Hankin C. Principles of Program Analysis[M]. Berlin: Springer-Verlag, 1999.

（上接第 32 页）

参考文献

1 Cech E. Topological Spaces[M]. New York: Wiley, 1966.

2 Galton A. A Generalized Topological View of Motion in Discrete Space[J]. Theoretical Computer Science, 2003, 305(1-3): 111-134.

3 Smith M B. Semi-metrics, Closure Space and Digital Topology[J]. Theoretical Computer Science, 1995, 151(1): 257-276.

4 Kelly J L. General Topology[M]. Springer-Verlag, 1975.

5 程家兴，陈万里，宋　杰. 相容关系与相容核[J]. 安徽大学学报(自然科学版), 2004, 28(4): 1-4.

6 陈万里，程家兴，张持健. 基于相容关系推广的粗糙集模型[J]. 计算机工程与应用, 2004, 40(4): 26-28.