基于 TTA 体系结构的嵌入式协处理器的设计与实现*)

赖明澈 戴 葵 陆洪毅 岳 虹 王志英
(国防科学技术大学计算机学院 长沙 410073)

摘 要 本文基于 TTA 结构提出了一种嵌入式协处理器体系结构,并完成了其 VLSI 设计与实现。该协处理器具有双 Cluster 的运算内核,能够高效地支持多媒体应用中的数据密集型计算。为了充分发挥协处理器工作效率,本文还设计了具有流缓冲代理特征的流存储子系统,通过实现数据流存储访问机制以及计算资源与片外存储之间的低耦合结构,提高访存带宽。最后,基于该嵌入式协处理器,本文在 0.18μm CMOS 工艺下实现了一款多核 SoC 芯片,其工作主频为 300MHz,实测功耗为 910mW。

关键词 传输触发体系结构,协处理器,代理缓冲

The Design and Implementation of the Embedded Coprocessor Based on TTA

LAI Ming-Che DAI Kui LU Hong-Yi YUE Hong WANG Zhi-Ying
(School of Computer, National University of Defense Technology, Changsha 410073)

Abstract A novel embedded coprocessor based on the Transport Triggered Architecture is presented in this paper. The coprocessor is consisted of two powerful arithmetic clusters, and good at exploiting the data parallelism in the computation intensive multimedia applications. To improve the efficiency, the coprocessor also designs the stream memory system with the characteristic of the stream buffer proxy, especially uses the decoupled architecture and the stream access mechanism to enhance the access bandwidth. Then, a heterogeneous multiprocessor SoC chip involving the coprocessor is implemented using 0.18μm CMOS process, which can operate at 300MHz and consume about 910mW.

Keywords Transport triggered architecture, Embedded coprocessor, Buffer proxy

1 引言

随着算法复杂性的逐渐提高与计算数据量的不断增大,当前多媒体领域(如 H.264、MPEG4 视频编解码等)迫切需要微处理器具有强大的计算能力,以满足实时处理的需求。根据对大量多媒体算法的研究,尤其是对核心算法的深入分析[1],可以发现多媒体应用呈现出了数据精度低、计算密集以及大规模数据并行等基本特征。针对上述特征,目前典型的嵌入式数字信号处理器,如 Infineon 公司的 Tricore 系列、飞利浦公司的 Trimedia 系列与 TI 公司的 TMS320C64 系列[2],均采用了基于超标量或 VLIW 体系结构的高性能 DSP 内核。然而,这些内核的译码逻辑、指令发射逻辑、数据通路及其旁路机制却比较复杂,占用了相当部分芯片资源,在一定程度上限制了计算密度的提高。特别是,随着应用需求的迅猛增长,其体系结构可扩展性与低功耗等问题[4]也接踵而至。

为此,本文研究并实现了一种基于 TTA 体系结构[3]的嵌入式协处理器,其显著优势主要体现在具有丰富的计算资源、简单的分布式控制以及易扩展的体系结构等方面。该协处理器采取了双 Cluster 的组织方式,每个 Cluster 内部设置了强大的互连网络,使得不同类型的功能部件连接在互连网络上,通过编译器的合理调度,有效地融合数据流和控制流,同时开发指令级并行(ILP)与数据级并行(DLP)。此外,为了充分发挥协处理器工作效率,本文还设计了具有数据流代理特征的流存储子系统,它以有限的资源代价为前提,从不同角度出发,提高了系统的整体性能。在本文中,首先介绍 TTA 计算模型,然后给出协处理器体系结构,其次描述流代理特征的流存储子系统,最后给出具体的实现以及性能评测结果。

2 TTA 计算模型

TTA 的体系结构如图 1 所示。其核心部分 CPU 内核不仅拥有大量的功能单元(FU)、特殊功能单元(SPU)以及寄存器文件(RF),而且配置了强大的互连网络,用于提供足够的通信带宽。与 VLIW、超标量等体系结构中操作触发的原理不同[6],TTA 体系结构的核心思想是利用数据传输触发具体操作,即所有功能单元上的任何操作均以数据传输作为唯一的触发激励,也即任何数据源中的数据经过互连网络的传输后,在写入功能单元数据寄存器的同时会触发一次完整的操作。因此,TTA 体系结构中的每个功能单元均包含一个或者多个操作数寄存器、唯一的触发寄存器和若干个结果寄存器,如图 1 所示。当数据被写入触发寄存器时,它会触发功能单元将操作数寄存器和触发寄存器中的值作为源操作数,来完成相应的操作,并将结果写入结果寄存器。

同时,TTA 体系结构在支持并行性上还有如下优势[7]:

(1)结构简单、计算能力强;

(2)功能单元和互连网络可配置,提供了设计的灵活性和性能的可扩展性;

(3)功能单元包含寄存器,且统一编址,因此旁路了中间结果的保存,不仅有效减少了寄存器文件的读写次数,降低了文件端口压力,同时提高了性能;

(4)互连网络、寄存器文件与功能单元三者间的低耦合互

*)本课题受自然科学基金支持(No. 60173040, No. 90407022)。赖明澈 博士研究生。

连结构更适合于开发三者流水线之间的并行性，从而大幅度提高硬件利用率；

(5)基于一种更加灵活、更加细粒度的传输触发模式，有利于编译器从多个层次开发并行性，不仅可以有效支持指令级并行，同时能够有效支持数据传输级并行。

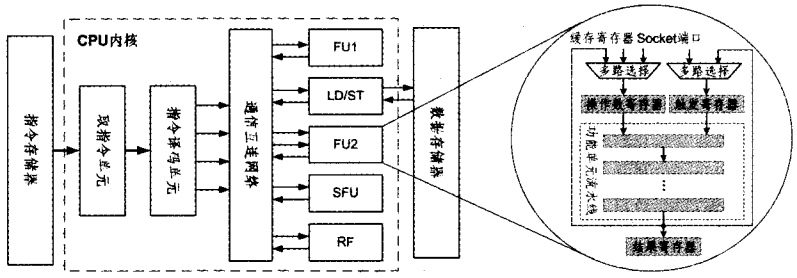

图1 TTA体系结构框图

3 协处理器体系结构

本文设计的嵌入式协处理器结构如图2所示。它包括控制单元(CCU)、指令Cache、流存储子系统以及两个运算Cluster。CCU单元是整个协处理器的控制中枢，它一方面负责解释执行外部(如其他主处理器)向协处理器发出的命令，另一方面负责每个周期从指令Cache中取指令，并完成指令的译码操作。需要说明的是，本文中单个Cluster的指令包含8个传输槽和1个立即数槽，每个传输槽带有三个子域：条件码(C)、源域(S)及目标域(D)，CCU的每次译码均会产生8个从S到D的条件传输操作。另外，协处理器还拥有3个外部接口：控制接口、取指令接口及数据接口。控制接口主要承担与主处理器之间的通信任务，而指令接口与数据接口则分别负责取指令以及与片外存储器之间的数据交互。

双Cluster结构是协处理器完成多媒体数据处理的核心部分，其内部结构如图3所示。协处理器在每个Cluster内部配置了8条32位宽度的数据总线，不同的功能单元通过Socket端口与总线间接互连，形成了协处理器内部以互连网络为中心的体系结构，实现了以数据传输为前提的触发操作机制。同时，在互连网络的设计过程中，通过对数据传输过程的分析，协处理器实现了局部互连网络(如图3所示)，使得每条总线有针对性的服务于特定单元之间的数据传输，有效地删掉了冗余的互连节点，并且在减少总线负载的同时，降低了CCU的硬件复杂性。

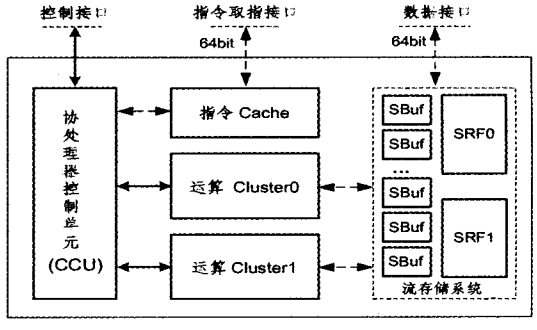

图2 协处理器体系结构框图

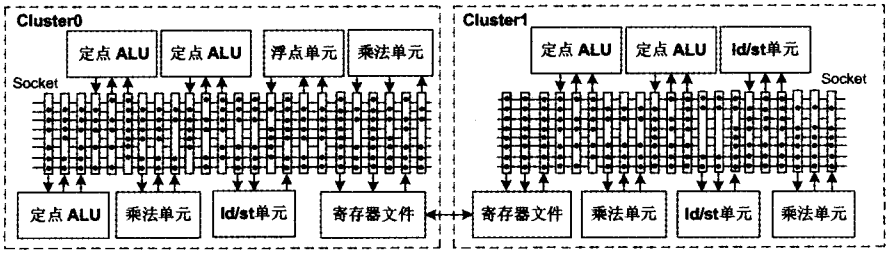

图3 双Cluster的内部结构框图

特别的是，面向数字信号及多媒体应用领域，为了充分发掘协处理器内核的工作效率，本文还选择了TTA的嵌入式ASIP设计流程[5,13]进行了协处理器的功能单元配置。ASIP设计者首先针对目标应用程序集，结合设计约束进行分析，得到指导指令集和体系结构生成的必要信息。其次，依据指导信息以及设计约束，对可能的体系结构进行性能评估与资源优化，选择满足性能、功耗约束以及硬件代价最小的TTA配置，同时生成相应的指令集。上述设计过程[5]简单、灵活，有效地克服了当前ASIP设计的局限性，针对具体的应用背景，能够快速设计出硬件代价小、耗费功率低的高性能嵌入式微处理器。基于上述ASIP设计方法，本文对协处理器内核的硬件资源进行了合理的选择优化，得到的功能单元配置如表1所示。

4 流存储子系统

多媒体应用领域的数据量大、重用性低，传统数据Cache的容量有限并缺少必要的预取机制，数据访问效率低。为了提高视频、图像编解码等多媒体应用的性能，协处理器设计了基于流寄存器文件(SRF)的存储子系统。该系统中，多媒体数据被组织成数据流的形式在流寄存器文件与片外存储器之间传输，较好地满足了多媒体应用对数据带宽的迫切需求。如图4所示，协处理器的存储系统由DMA控制器和流寄存器堆组成，流寄存器堆负责数据流的中间存储，而DMA控制器承担与片外存储器之间的数据交互。协处理器的这种设置有利于实现计算资源与片外存储之间的低耦合构架，硬件开销少，有效地缓解了存储带宽的瓶颈问题。特别地，协处理

表 1　每个 Cluster 下的功能单元描述

单元名称	配置个数	功能描述
定点 ALU 单元	4	支持 8 位、16 位子字并行模式；参考 MMX 指令集实现了多媒体指令扩展。
定点乘法单元	2	支持 8 位、16 位的子字并行操作；流水线建立时间为 2 个时钟周期。
比较单元	1	支持屏蔽方式的条件传输[8]。
跳转单元	1	配合比较单元，支持无条件跳转及有条件跳转。
浮点单元	1	操作数遵循 IEEE-754 标准；除了浮点除法、浮点开方耗费 15 与 24 个周期外，其它操作均支持单周期流水输出。
load/store 单元	3	32 位宽度；支持各种粒度的有符号或无符号数据的存取。
寄存器组	1	4 体结构；每个体包括 48 个寄存器，映射为 4 组，组间共享 4 个寄存器，从而建立了一种环形窗口构架；在函数调用或返回时，利用切换窗口及重叠窗口技术加快程序运转。

器为了实现与片外大容量存储器之间的协同工作，还专门设置了 SRF 资源调度表格。其中，表格项〈locked，working〉作为一种原子信号量，用于记录各 SRF 体的工作状态，支持对 SRF 体的互斥访问。其工作原理是：首先，Host 或协处理器会确认被请求的 SRF 体是否处于〈0，0〉状态，通过确认后，处理器会向 DMA 控制器发出 DMA 读命令，请求加载数据；紧接着，DMA 控制器按照“先入先出”顺序处理请求，待 DMA 数据加载完成后，该 SRF 体的状态修改为〈1，1〉；然后，协处理器才有权对该 SRF 体上数据进行处理，此时运算操作可以与其它 SRF 体上的 DMA 传输并行执行，待运算结束，状态再修改为〈1，0〉；最后，该 SRF 体会等待 Host 或协处理器发出 DMA 写命令，请求将运算结果写回片外存储器，同时返回状态为〈0，0〉。

为了进一步增加流存储子系统的访问带宽，提高内核的工作效率，协处理器还采用了流寄存器文件与流缓冲（SBuf）的两级存储结构，如图 5（c）所示。每个 Cluster 均对应独立的寄存器堆，每个寄存器堆配置了 16kB 寄存器文件与 7 个输入输出流缓冲，两者之间利用 Crossbar 网络互连。其中，基于功耗、面积上的考虑，所有寄存器文件仅仅被组织成了单端口形式，从而在一定程度上增加了访问冲突次数，限制了数据吞吐率。为此，本文将流寄存器文件组织成了四体结构，允许四路并行访问；同时还在 Cluster 与寄存器文件之间设置了若干个流缓冲，它们能够依据各自对应的流式描述算子[11]，分时复用寄存器文件的访问端口，顺序存取数据流，从而提高一个更高层次的存储带宽。同时，随着基于描述算子的数据流访问机制的引入，该模型还可以有效地减少对 load/store 单元操作数寄存器的频繁访问，缓解 TTA 传输总线上的工作负载，从而进一步提高 Cluster 内部可以同时被触发的功能单元数量。针对上述结构，本节选取了 fft 和 mpeg2dec 两个测试程序对存储系统进行了性能评估。通过高速流寄存器文件的组织方式及流缓冲大小，图 6 分别显示了各个程序在不同存储配置下的运行时间百分比。可以看出，后四种配置有不同大小的流缓冲结构 FIFO 明显比不具有流缓冲的 Crossbar 结构占有性能上的优势，并且随着流缓冲大小的增加，性能加速的效果基本趋于饱和。

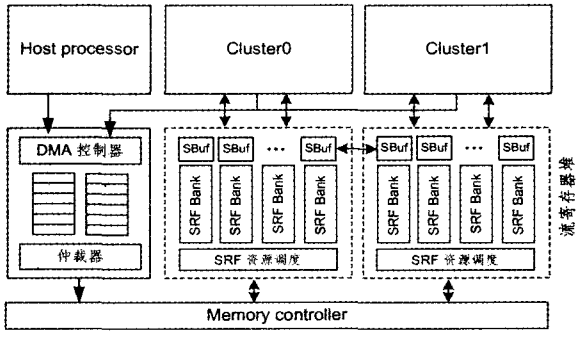

图 4　协处理器存储系统框图

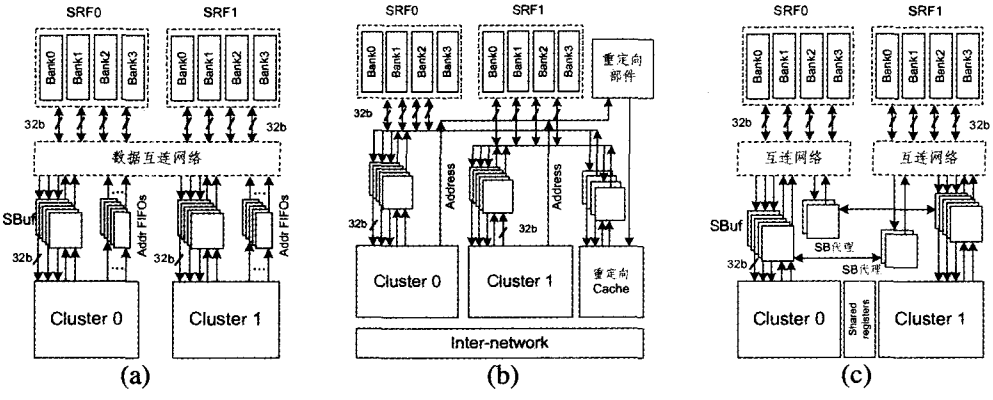

图 5　不同的 SRF 与流缓冲组织结构

此外，在现代数字信号及多媒体应用领域存在大量本地数据流访问的同时，程序本身还存在着大量的 Cluster 间的交叉访问。尤其是对于数据相关的交叉访问，编译总是以最坏方式进行通信资源分配，为此 Imagine[9] 和 MASA[10] 分别采用了 Crossbar 和数据重定向 Cache 两种方案。图 5（a）所示的 Crossbar 方案能够支持 Cluster 间的交叉访问，但扩展性差、访问延迟大，而且存在有大量数据缓冲之间的存储一致性问题[12]。而图 5（b）所示的数据重定向 Cache 在将数据流从一个 SRF 重定向到另一个 SRF 的过程中会消耗大量资源，它必须预留本地的 SRF 存储空间来存取远程的数据副本，同时其逻辑耗时长，往往会阻塞本地 Cluster 执行。

流缓冲代理是针对嵌入式协处理器提出的一种支持多核交叉访问的存储机制。如图 5（c）所示，协处理器在每个 Cluster 中配置了 5 个流缓冲分别服务于本地的 3 个 load/

store 功能单元，而其余流缓冲将扮演本地代理的角色来支持远程数据访问。其工作原理是：首先，请求方 Cluster 发送流算子给本地 load/store 单元；随后，本地 load/store 单元开始分析流算子，一旦解析到该算子访问远程 SRF 体，load/store 单元将申请对方的流缓冲代理服务并进行流算子的重定向；所有本地 load/store 单元相互竞争使用资源，一旦申请成功，load/store 单元将利用远程代理提供的服务开始对远程数据流进行访问，同时使用硬件原语操作将缓冲代理加锁直至本次任务结束。另外，由于流寄存器文件的数据副本仅分布于本地流缓冲以及缓冲代理之中，该方案仅需要较小硬件开销便能维护顺序一致性存储模型，从而为 TTA-C 编译提供一个良好的开发环境。

5 协处理器实现与性能分析

5.1 协处理器实现

基于上述嵌入式协处理器，本文采用 0.18μm 六层金属布线 CMOS 工艺设计实现了一款异构多核 SoC 芯片 ATOM-I，其体系结构如图 6 所示，其中包括嵌入式双核协处理器、存储控制器、外围设备控制器及配置有 4kB 指令 Cache 和 4kB 数据 Cache 的 LEON3 主处理器[12]。该 SoC 芯片主频能稳定在 300MHz，平均功耗 0.91W，芯片的基片大小为 4.9mm×4.9mm。表 2 给出了协处理器的主要性能参数。图 6 给出了 ATOM-I 的版图。

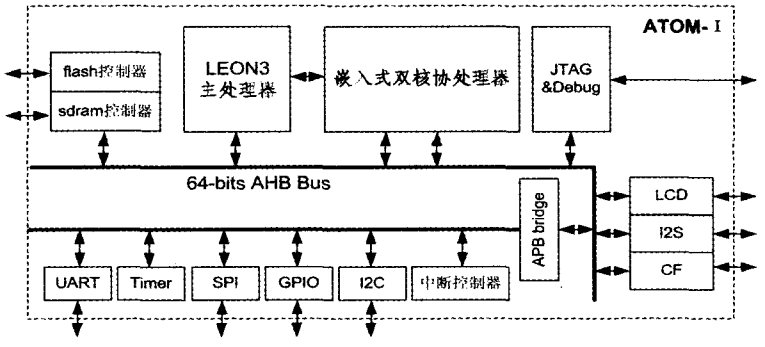

图 6 ATOM-I SoC 芯片的体系结构框图

表 2 嵌入式协处理器的主要性能参数

设计工艺	0.18um CMOS-1P6M
稳定工作主频	300 MHz
片内存储器	I$：12kB SRF：32kB
协处理器面积	≈8.6 mm²
工作电压	1.8V(core)，3.3(external，5V tolerant)

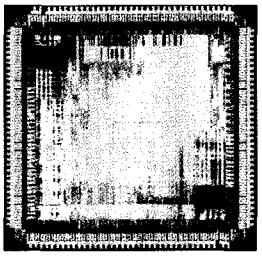

图 7 SoC 芯片 ATOM-I 版图

5.2 性能测试

表 3 核心程序的执行时间(单位：时钟周期)

测试程序	TI-C64	单个 Cluster
Fft	1243	1127
Fir	1519	1577
Maxidx	82	104
Iir	533	562

fft：256 点带数字反转的复数正向 FFT，16 位数据格式
fir：复数 FIR 滤波器，32 个系数和 128 个输出样本，16 位数据格式
maxidx：128 项向量中寻找最大值，16 位数据格式
iir：128 个输出样本，16 位像素点

首先，本文选择了能够体现数字信号以及多媒体应用领域特征的核心程序集合，在理想的外部存储模型下，对协处理器内核单个 Cluster 的性能进行了评测，并与 TI 公司的高性能 DSP TMS320C64 系列处理器[2]进行了比较。为了充分挖掘程序中数据传输级的并行性，下述测试程序经过了汇编级优化，然后在协处理器的模拟器上执行；而 TI-C64 处理器的测试结果通过 TI 的开发环境 CCS2.0 模拟得到。如表 3 所示，单个 Cluster 基本上具有与 TI-C64 相当的性能。

其次，本文对该 SoC 芯片进行了全面的性能测试。在性能测试的过程中，测试程序均选自于目前的图像与信号处理领域(如表 4 所示)。本文采用的测试程序均由统一的 C 语言描述。我们一方面将不同类型的计算任务分配到不同类型的处理器内核上，利用 SPARC-C 编译器与 TTA-C 编译器编译得到的二进制代码对芯片上进行测试，另一方面评估了相同程序在主处理器上的运行性能，得到的时钟周期情况如表 4 所示。从性能测试的结果来看，性能加速比达到了 4.1～5.6。

表 4 在典型测试程序下的性能比较(单位：时钟周期)

测试程序	LEON	ATOM-I	加速比
H264_dec	13.5×10^6	3.21×10^6	4.2
JPEG_enc	13.8×10^6	2.45×10^6	5.6
MPEG4_dec	3.52×10^6	0.85×10^6	4.1

H264_dec：H264 解码，QVGA 240×320 像素
JPEG_enc：JPEG 编码，QVGA 240×320 像素
MPEG4_dec：MPEG4 编码，QCIF 176×144 像素

总结 本文研究并实现了一款基于 TTA 结构的嵌入式协处理器，它采用双 Cluster 的框架，专门应用于面向多媒体应用领域的程序加速。在相应的工作中，本文首先进行了功能单元的配置，其次着重讨论了协处理器中的流存储系统，它以有限的资源代价为前提，从不同角度出发，提高了系统的整体性能。

基于嵌入式协处理器，本文实现了一款异构多核 SoC 芯

片 ATOM-I。实际芯片的性能评测结果表明：该嵌入式协处理器对多媒体应用具有显著的加速效果，并且整个芯片的平均功耗仅有 0.91W。因此，本文研究并设计的基于 TTA 结构的嵌入式协处理器较好地满足了数字信号处理的需求，能够完成多媒体应用领域的关键算法加速。

参 考 文 献

1 Corbal J, et al. DLP+TLP processors for the next generation of media workloads. In: Proc. 7th Intl Symp on HPCA, 2001
2 TMS320C64 CPU and Instruction Set Reference Guide, Texas Instruments, Inc, USA, 2000
3 Corporaal H, Janssen J, Arnold M. Computation in the Context of Transport Triggered Architectures. International Journal of Parallel Programming, August 2000, 28(4): 401～427
4 13 Dr Cichon G. Introduction into Synchronous Transfer Architecture (STA). Feb. 2005
5 岳虹，沈立，戴葵，等. 基于 TTA 的嵌入式 ASIP 设计. 研究与发展，2006，43(4)：752～758
6 Hoogerbrugge J, Corporaal H. Transport-triggering vs operation-triggering. In: Compiler Construction Conference CC-94, 1994
7 Yue Hong, Lai Ming-Che, Dai Kui, et al. Design of a Configurable Embedded Processor Architecture for DSP Functions. In: Proceedings. 11th International Conference on, July 2005, (2): 27～31
8 Kapasi U J, Dally W J, Rixner S. Efficient Conditional Operations for Data-parallel Architectures. In: Proc. Intl Symp on Microarchitecture, Dec. 2000. 159～170
9 Jayasena N, Erez M, Ahn Jung Ho, et al. Stream Register Files with Indexed Access. In: Tenth International Symposium on High Performance Computer Architecture, Madrid, Spain, February 2004
10 Wu Nan, Wen Mei, Li Haiyan, et al. A Stream Architecture Supporting Multiple Stream Execution Models. LNCS 3740. Sep. 2005. 143～156
11 Lopez-Lagunas A, Chai S M. Compiler Manipulation of Stream Descriptors for Data Access Optimization. In: Proceedings of the International Conference Workshops on Parallel Processing, Jan. 2006. 337～344
12 http://www.gaisler.com/cms4_5_3/index.php?option=com_content&task=view&id=115&Itemid=103
13 Pitkanen T, Rantanen T, Cilio A, et al. Hardware Cost Estimation for Application-Specific Processor Design. SAMOS 2005, LNCS 3553, 2005. 212～221

(上接第 252 页)

点；由于 $\exists x(\delta_{x,a} \wedge x=4)$ 为真，因此根结点的右孩子结点不可见，故 $\alpha_{dbs}(T)$ 为图 3(c)所示的标号树。

定理 6.3 已知程序 P 和目标 G，在 LT 的完备化扩展上以下关系成立：

$\bigcup_{i\geqslant 0}\mathcal{O}_{G,P,i}=\alpha_{dbs}(\bigcup_{i\geqslant 0}K_i^G)=\alpha_{dbs}(J_P^{LT}[\![G]\!]F_P^{LT})$。

以上定理表明，LT 语义包含了比本文操作语义更多的信息，通过删除 LT 语义中那些实际上不能被访问但对于获得语义的目标独立性所必须的结点，可以得到与操作语义等价的信息。该定理可视为 LT 语义相对于操作语义的正确性结论。因此 LT 语义可作为目标独立的 Prolog 程序路径依赖分析的基础。

结束语 在抽象解释的框架内进行目标独立的 Prolog 程序路径依赖分析，需要建立能完整描述程序执行路径的目标独立语义。现有 Prolog 语义不能满足该分析的要求。本文给出了一种以标号树序列为语义域、目标独立的标号树语义。由于该语义可利用开放和闭合 cut 为 cut 操作建模，且每一个标号树结点均携带了完整的执行路径信息，因而可用于具有 cut 操作的 Prolog 程序目标独立分析，并利用调用串提高程序分析的精度。我们证明了该语义相对于本文操作语义的正确性。将来的工作包括根据不同需求开发基于该语义的抽象解释工具，并研究其在 Prolog 编译器和部分求值器中的应用。

参 考 文 献

1 Barbuti R, Codish M, Giacobazzi R, et al. Modeling Prolog control. Journal of Logic and Computation, 1993, 3: 579～603
2 Barbuti R, Codish M, Giacobazzi R, et al. Oracle Semantics for Prolog. In: Kirchner H, Levi G, eds. Algebraic and Logic Programming, Proceedings of the Third International Conference, Lecture Notes in Computer Science, vol 632, 1992. 100～114
3 Barbuti R, Giacobazzi R, Levi G. A General Framework for Semantics-based Bottom-up Abstract Interpretation of Logic Programs. ACM Transactions on Programming Languages and Systems, 1993, 15(1): 133～181
4 Bossi A, Bugliesi M, Fabris M. Fixpoint Semantics for Prolog. In: Warren D S, ed. Proceedings of the Tenth International Conference on Logic Programming, Cambridge, MA: MIP Prese, 1993. 374～389
5 Börger E. A Logical Operational Semantics for Full Prolog. In: Börger E, Büning H K, Richter M M, eds. CSL'89. Third Workshop on Computer Science Logic, Lecture Notes in Computer Science, vol 440, 1990. 36～64
6 Codish M, Dams D, Yardeni E. Bottom-up Abstract Interpretation of Logic Programs. Journal of Theoretical Computer Science, 1994, 124: 93～125
7 Cousot P, Cousot R. Abstract Interpretation and Applications to Logic Programs. Journal of Logic Programming, 1992, 13 (23): 103～179
8 Filé G, Rossi S. Static Analysis of Prolog with Cut. LPAR, 1993. 134～145
9 Fitting M. A Deterministic Prolog Fixpoint Semantics. Journal of Logic Programming, 1985, 2(2): 111～118
10 Gabbrielli M, Levi G, Meo M C. Resultants Semantics for Prolog. Journal of Logic and Computation, 1996, 6(4): 491～521
11 Henkin L, Monk J D, Tarski A. Cylindric Algebras (Parts I and II). North-Holland, Amsterdam, 1971
12 Jaffar J, Maher M J. Constraint Logic Programming: A survey. Journal of Logic Programming, 1994. 19～20, 503～581
13 Charlier B L, Rossi S, Van Hentenryck P. An abstract Interpretation Framework Which Accurately Handles Prolog Search Rule and the Cut. In: Bruynooghe M, ed. Proceedings of the 1994 International Symposium on Logic Programming, Cambridge, MA, MIT Press, 1994. 157～171
14 Le Charlier B, Rossi S, Van Hentenryck P. Sequence-based Abstract Interpretation of Prolog. Theory and Practice of Logic Programming, 2002, 2(1): 25～84
15 Levi G, Micciancio D. Analysis of Pure Prolog Programs. In: Sessa M I, Ed. Proceedings GULP-PRODE '95, 1995. 521～532
16 Levi G, Spoto F. Accurate Analysis of Prolog with Cut. In: Lucio P, Martelli M, Navarro M, eds. Proceedings APPIA-GULP-PRODE'96, 1996. 481～492
17 Lu L. Path Dependent Analysis of Logic Programs. In: Proceedings of ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation (PEPM '02), 2002. 63～74
18 Marriott K, SØndergaard H, Jones N D. Denotational abstract interpretation of logic programs. ACM Transactions on Programming Languages and Systems (TOPLAS), 1994, 16(3): 607～648
19 Mellish C. Abstract Interpretation of Prolog Programs. In: Abramsky S, Hankin C, eds. Abstract Interpretation of Declarative Languages, 1987. 181～198
20 Nielson F, Nielson H R, Hankin C. Principles of Program Analysis. Springer-Verlag Heidelberg, 1999
21 Spoto F. Operational and Goal-independent Denotational Semantics for Prolog with Cut. The Journal of Logic Programming, 2000, 42: 1～46
22 Spoto F, Levi G. Abstract Interpretation of Prolog Programs. In: Haeberer A M, ed. Proceedings of the Seventh International Conference on Algebraic Methodology and Software Technology, AMAST'98, Lecture Notes in Computer Science, vol 1548. Amazonia, Manaus, Brazil, January, Springer, Berlin, 1999. 455～470
23 Tarski A. A Lattice Theoretical Fixpoint Theorem and its Applications. Pacific Journal of Mathematics, 1995, 5: 285～310
24 Winsborough H. Path-dependent Reachability Analysis for Multiple Specialization. In: Proceedings of the North American Conference on Logic Programming, 1989. 133～153