编译基础设施研究进展初谈*)

Advance in Compiler Infrastructures

戴桂兰　田金兰　张素琴　蒋维杜

(清华大学计算机科学与技术系　北京100084)

Abstract Based on main components of the compiler infrastructure, the paper focuses on discussing some key techniques for compiler back ends, reviews the recent representative common compiler infrastructures and their used individual techniques, and outlines some current issues of compiler back ends and development directions for the further researches.

Keywords Retargetable compiler, Compiler infrastructure, Machine description, Code generation, Intermediate representation

1 引言

为便于开展对计算机体系结构、编程语言和编程环境的研究,必须能够快速地产生高质量的编译器。编译器的构造是一个繁琐的工作,经常是一个瓶颈。在通常情况下,尽管必须修改的成份仅占整个编译器的一小部分,但却要重新构造整个编译器。在编译器成份和算法方面虽已积累了丰富的研究成果,尚未能为广泛编译器开发者所共享。随着嵌入式系统的迅速发展和高性能体系结构的推陈出新,对编译器的开发速度和质量提出了新的挑战和需求[5]。为此,人们在编译基础设施方面做了大量卓有成效的工作,研制了一些在理论上、实践上均有重要价值的编译基础设施,以便于编译器成份的复用和编译技术的共享,进一步实现高质量编译器的快速开发。

2 编译基础设施

编译基础设施不是一个具体的编译器,而是编译器的开发环境,它提供一系列开发编译器的策略和工具,支持多种源语言、多目标机编译技术,以便于人们在具有较高抽象层次的平台上进行编译器开发和研究工作。

编译器成份一般包括词法分析器、语法分析器、语义检查、代码生成器、代码优化器、机器描述工具和目标机模拟器。这些成份在操作功能上比较独立,但在语义功能上密切相关。编译基础设施为开发各种编译器成份提供相应的生成工具。

对编译前端(如:词法分析器和语法分析器)的研究已有比较成熟的理论基础和实用的开发工具,而对编译器后端研究离应用需求仍存在很大的差距,而它又是编译器开发的关键,尤其是对嵌入式系统的开发。随着嵌入式系统应用的市场迅猛扩大,对嵌入式软件在正确性、灵活性、规模和快速高效方面提出了极高的要求,从而对编译器的开发技术提出了新的挑战[9]。因此,本文针对嵌入式系统的开发,着重讨论编译基础设施中支持多目标的编译构造技术。

3 多目标编译器构造技术

由于编译基础设施用于构造编译器,因此对于多目标编译后端构造,中间表示、代码优化和代码生成等传统的编译技术,应该为编译基础设施提供依据。编译基础设施中的代码优化技术与传统的没有什么本质区别,但对构造中间表示技术提出了更高的要求[14]。另外,由于不同的源语言之间、不同的处理器之间和不同的操作系统之间存在着相当的差异性,也存在着很多的共同性,因此对于支持多源语言多目标机的编译系统而言,它还需要目标机及目标操作系统描述技术、代码生成器的构造技术以及表达两者间相关性的接口技术为之提供有力的支持。

3.1 机器描述技术

在传统的多目标编译器的构造中,常常假定一个相当固定的体系结构抽象模型[15]。由于不同体系结构的指令系统、寄存器方式、标量、流水线以及嵌入式系统的信号量各不相同,因此这种方法并不能产生高质量的目标代码。随着计算机体系结构的不断推陈出新,固定的体系结构模型已难以满足应用需求,因而嵌入式系统越来越多地采用的是混合型的风格。为便于产生满足各种条件(如:能量消耗、代码尺寸等)的目标代码,提供一种对目标机进行恰当描述的机制,精确描述处理器资源及处理器的指令集如何利用这些资源等显得越来越重要。这种机制既应能描述系统和硬件的共性,又应能描述它们各自的个性,同时还应适于编译程序的处理。研究表明,基于体系结构描述语言详细地指定体系结构是产生高质量机器级工具的关键技术[1]。

3.2 代码生成器的构造技术

一般来说,代码生成器的构造器的输入是机器描述,输出是代码生成器。由于汇编语言的泛味、易错、耗时且程序难以维护等缺点,在大型的嵌入式系统应用的开发中,它正逐步为高级程序设计语言(如C、C++等)所替代。这些语言能够提供方便的抽象层次以及较高的模拟速度,从而可简化软件开发

*)基金项目:国家自然科学基金(名称:嵌入式软件开发环境研究,编号:60083004),清华大学'985'重点项目(名称:嵌入式操作系统及其开发环境创新技术与方法的研究)。戴桂兰　博士后,主要从事编译技术、对象技术、软件工程等方面的研究工作。蒋维杜　教授,主要从事编译技术、对象技术、软件工程等方面的教学科研工作。张素琴　教授,主要从事编译技术,系统软件,软件开发环境等方面的教学与科研工作。田金兰　副教授,主要从事软件工程、软件开发环境方面的教学与科研工作。

周期。然而，高级程序设计语言与其支撑的机器指令集合体系结构之间存在着灵活的、多变的、复杂的对应关系，利用代码生成器将高级语言程序(通过中间表示)转换为等价的目标程序是不可缺少的。处理器的日益更新，自然对代码生成器的自动构造提出了更高的需求。在评估一个特殊体系结构与目标代码在尺寸和执行时间上的有效性间的关系方面，代码生成器的构造器似乎是一个大有作为的工具[14]。为简化开发过程、提高代码生成器的可靠性，支持代码生成器的自动构造是编译基础设施中的关键技术之一，也是主要难点之一。

3.3 目标机描述与目标代码生成的接口技术

目标机描述与目标代码生成的接口，指定了与目标机无关的前端及与目标机有关的后端之间的相互联系。为简化编译器的开发、提高编译器效率，在编译基础设施的研究过程中，抽取一系列函数和数据结构作为目标代码生成与机器描述之间的接口是很有必要的[4]，而好的接口设计有较大的难度。如果接口太小，它或许不能编码足够的信息以充分利用新机器的特征；如果接口太大，会使接口具有不必要的复杂性。由此可见，如何设计和组织这些数据结构和函数也是一个关键问题。

3.4 中间表示技术

中间表示是在编译程序将高级语言程序翻译为汇编语言或机器代码的过程中产生的。对于书写支持单个源语言、运行在单个目标机上的编译器而言，程序代码的中间表示主要用于实现优化。传统的中间表示(如：三元式、四元式、树和DAG等)虽为循环优化、数据控制流分析等编译任务提供了适当的模型，但它们并不能满足编译器的重定目标的需要。中间表示对提高编译器的可移植性以及代码生成的有效性起到关键作用[14]。因此，在编译基础设施的研究中，应提供一种结构良好的中间表示。

4 代表性工作

为便于吸取现有编译基础设施中的好的设计思想和技术，同时，为编译器的开发选择理想的工具提供指导，这里简单介绍几个比较成熟的、有代表性的、广泛应用在工业上或编译技术研究中的一些编译基础设施，并着重比较和分析其中采用的较有特色的中间表示技术及后端构造技术。

4.1 GCC

GCC(GNU Compiler Collection)是FSF启动的GNU工程的一部分，其开发目标在于提高GNU系统中编译器的质量。GCC目前已支持的源语言有C、C++、Objective-C、FORTRAN、Ada，已移植的平台有一百多种，涉及三十多种处理器、六十多种系统[3]。GCC属于自由软件，在全世界范围内得到了广泛使用。

GCC采用两层中间表示，它们分别为语法树和RTL(Register Transfer Language)表达的中间代码。GCC的RTL代码则由这样一条一条的指令组成，每条RTL指令实际上都抽象地表达了目标机的一条指令，但对同一条指令的操作意义而言，不同的目标机完成这一操作的指令条数、操作数的形式以及汇编格式各不相同，从而使其机器描述几乎不具有可复用性。由于前端产生的语法树设有保留，且机器信息从不同的编译阶段进入，从而使其编译成份间具有较高的耦合度。对于任何目标机，由于insn集合不发生变化，使语法树到RTL中间表示的转换比较简单，从而使其编译后端可以自动产生。

4.2 Zephyr

Zephyr编译基础设施是美国启动的NCI工程的主要部分之一。它是由弗吉尼亚大学和普林斯顿大学联合开发的。Zephyr以VPO(Very Portable Optimizer)为核心，支持机器指令级的优化，目前已用于开发了支持java、C++等多种源语言，Alpha、Mips、Motola 88100等二十多种处理器的编译器[2]。

Zephyr基础设施也提供了两层中间表示，对其高级中间表示没有明显的限制，它可为前端输出的任意形式，机器信息通过代码扩充器(Code expander)一次性地进入编译主体，使编译成份间的耦合度较低。低级中间表示是具有强类型树形结构的RTL(Register Transfer List)，其语义描述与任何机器无关。Zephyr提供一个语言簇CSDL以进行机器规格说明，并提供了一些标准操作码，对于一个新的目标机而言，只定义其特有的部分指令就可以了，从而使机器描述比较简洁且可复用性好。另外由于用户可以定义新的RTL操作符，这虽提高了机器描述的灵活性，但不利于从高级中间表示到RTL表示的转换，因此其代码扩充器需要手工编写。

Zephyr的目标在于支持编译器研究，而不是工业使用，将其直接改编为高质量的产品编译器比较困难。

4.3 SUIF

SUIF编译基础设施是美国启动的NCI工程的另一个主要部分[16]，它是由斯坦福大学开发的。SUIF以SUIF中间格式(Very Portable Optimizer)为核心，支持高级的优化，如：自动并行优化和面向对象优化。SUIF编译基础设施目前提供了C和Fortran前端、高级并行和局域优化器、一个优化MIPS后端和一系列编译器开发工具。SUIF中间格式是用C++实现的一种典型的高级中间表示，它提供了表达语言程序的共同格式，其核心部分采用了基于类库的面向对象层次结构。SUIF中间格式的主要设计目标在于支持过程间分析和高级优化，且方便于优化和分析的扩展。它是相当灵活的，允许编译遍的扩充和重新排序，但这种灵活性以性能为代价，因此不适用于开发编译器，但在教学和编译技术研究中得到广泛使用。

4.4 SGI Pro64

SGI Pro64编译基础设施是由SGI公司开发的。它为运行在Linux操作系统上的Intel IA-64体系结构提供了一套优化编译器工具。它支持的源语言有C、C++、Fortran90/95和OpenMP。尽管SGI Pro64以SGI现有的编译器系列为基础，但由于IA-64自身还不够成熟，到目前为止至少有两个用其编译的SPEC2000基准程序不能够正确执行[13]。Pro64提供了可分为五个层次的中间表示WHIRL。在编译过程中，中间代码依次降低。多数优化算法绑定在特定层次的中间表示上。多层次中间表示的使用，实现了多源语言为多目标产生代码的编译器的集成。

目前，目标为Linux操作系统和Intel-64处理器的编译器的源代码已开放。SGI Pro64为较先进的开放源代码的编译基础设施，它将可能会替代GCC。

4.5 IMPACT

IMPACT (Illinois Microarchitecture Project utilizing Advanced Compiler Technology)编译基础设施是由伊利诺斯大学开发的。它具有谓词编译、指令级并行优化、编译器工程推测、基于侧面优化、高级的机器描述设施、基于指针的依赖分析和追踪设施等高级特征。通过分析和论证硬件的层次和

编译器支持，以便于理解体系结构升级的代价和有效性，从而为微处理器工业提供关键性、体系结构专门知识和编译器原型的研究。IMPACT 的主要重点在于展示、提高和利用指令级并行[6]。

IMPACT 提供了具有不同形式的两种机器描述语言：HMDES 和 LMDES[8]。其中 HMDES 的语法形式与人的直觉相符，具有较好的可读性和可修改性。利用转换器，将 HMDES 转换为 LMDES，为用户提供友好的语法检查器。利用 LMDES 文件，实现编译器对机器描述的快速装载。编译接口函数与其支撑的体系结构无关。基本接口函数不使用编译器内部的数据结构。利用这些基本函数，可构造功能更强大的编译器内部函数。对于美国重要的公司和学术研究者而言，IMPACT 已成为主要的编译基础设施。

4.6 Trimaran

Trimaran 系统是编译和性能监测的集成基础设施[7]。它是由惠普公司和伊利诺斯大学联合开发的，主要面向 EPIC (Explicitly Parallel Instruction Computing)体系结构，支持指令级并行体系结构的编译器后端技术研究，易于实现分析和优化模块的添加、删除和越级。Trimaran 系统也提供了详细的模拟环境和灵活的性能监测环境。

Trimaran 提供了基于图的中间表示，实现了数据依赖图、控制流图的可视化。Trimaran 也是利用一种基于关系数据库的机器描述语言的特殊格式 HMDES 进行机器规格说明。利用转换器，HMDES 描述被转换为 LMDES 文件装入到编译器并建成 MDS 数据库的内部结构。MDS 数据库可供 Trimaran 设施中的各个模块利用 mQs 接口进行调用。Trimaran 编译基础设施目前提供给非商业应用。

5 存在问题及进一步研究方向

编译基础设施的研究已取得了相当大的进展，有大量的先进技术和工具为之提供支持。编译基础设施的发展不仅仅受到编译技术因素的影响，许多其它因素也会对其产生不容忽视的影响。在一些重要因素的影响下，编译基础设施的研究与应用存在着一些问题。

5.1 传统的编译技术自身有待于进一步提高

编译基础设施的发展以传统的编译技术的发展为基础。由于计算机体系结构的日益复杂化，尤其是异构体系结构的使用，而在嵌入式系统的开发中，资源和限制的相对重要性各不相同，因此，现有的传统编译技术，如：代码生成、代码优化、寄存器分配和指令调度等技术已很难满足需求，有待于进一步提高和发展。

5.2 机器描述工具急需完善和规范化

如前所述，基于 ADL 的方法已成为描述计算机体系结构及产生高质量的机器级的工具的关键。由于机器描述和机器描述工具的重要性，上述的编译基础设施分别提供了自己的机器描述工具，如 GCC 的 RTL[3]，Zephyr 的 CSDL[1]以及 IMPACT 和 Trimaran 的 MDES[10]等等。这些机器描述工具虽在理论和应用等方面均具有重要价值，但仍存在着一些问题。这主要体现在以下几个方面：其一，机器描述语言的通用性较差，一般只适用于某一特定的体系结构；其二，未规范化，对机器描述的抽象层次、描述内容都没有统一的标准，因而很难保证机器描述的正确性和完备性；其三，一些机器描述工具在可扩展性和可复用性等方面存在着一定的不足，如 GCC 的 RTL；其四，不易于使用，对于一个新的体系结构，往往觉得无从下手。由此可见，为提高机器描述语言的适用性和机器规格说明的质量，有必要建立有效的且易于扩展的机器描述语言，提供友好的机器描述界面。

5.3 后端自动构造工具尚需有突破性的进展

中间代码的语义是内部的且是很明确的。由于用目标机的性质来解释这些语义，将中间代码语义的形式化描述与目标机器的形式化描述相结合存在一定的困难，因此，编译后端自动构造工具的出现比词法和语法分析器的构造器的出现要晚得多。在二十世纪七、八十年代，出现了许多编译后端的自动构造技术，它们大致可分为解释性代码生成、模式匹配代码生成和表驱动代码生成三大类[14]。开发了一系列代码生成器的构造器，如：BEG[11]、iburg[9,12]和 MBURG[10]等。上述一些编译基础设施也提供了相应的代码生成的自动构造技术，如：GCC 等。这种利用声明性描述构造代码生成器要比手工编写容易，而且需要的工作量少，但其效率往往还不足手工开发的十分之一，从而很难为广大用户所接受。为提高代码生成器的效率，一些编译基础设施中的代码生成器仍是手工编写的，如：Zephyr 等。为迎合嵌入式系统开发的需要，提高自动构造的代码生成器的适用性，其技术尚需有突破性的进展。

5.4 编译基础设施的使用状况尚需改善

编译基础设施研究现状中另一个明显不足是理论研究和工程应用脱节。首先，使新编译器软件由一些即插即用的对象组成是多数编译基础设施的主要设计目标之一。而实际上，编译器成份间存在一定的依赖关系，创建可操作的、有效的编译器对象集合是很困难的，例如：若中间表示发生变化，流图的创建对象是否仍然能正常工作就是一个问题。由此可见，对于开发者而言，复用其中的某些成份或许比较容易，但对一般的用户来讲，复用是很困难的。其次，熟练掌握一个编译基础设施的使用比较困难，从而使之很难为广大编译器的开发者所接受。最后，多数编译基础设施是针对研究性应用的，它们常常以牺牲性能和尺寸以换取灵活性，从而使用其产生高质量的编译器是不切实际的。改善编译器的开发效率和质量，很重要的一点是探索有效的解决途径，简化编译基础设施的使用，且使开发出来的编译器满足应用领域的需要。

参 考 文 献

1 Norman R, Jack W D. Machine Description to Build Tools for Embedded Systems. In ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES'98). Springer Verlag, 1998, 1474: 172～188

2 Andrew A, Jack D, Norman R. The Zephyr Compiler Infrastructure. http://www.cs.virginia.edu/zephyr/

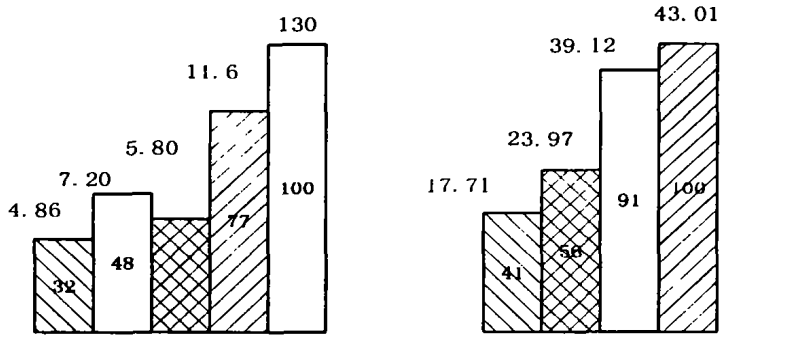
3 Stallman R M, Richard M. Using and Porting GNU CC (for version 2.95). Free Software Foundation, Inc. 1999

4 Fraser C. W, Hanson D R. The lcc 4.x Code-generation Interface, Microsoft Research: [Technical Report]. July 2001

5 Nikil D, Alex N, Hiroyuki T, Ashok H. New Directions in Compiler Technology for Embedded Systems. In: Proc. of the Conf. on Asia South Pacific Design Automation Conf. Yokohama, Japan. 2001

6 Chang P P, et al. IMPACT: An Architectural Framework for Multiple-Instruction-Issue Processors. In: Proc. of the 18th Annual Int'l Symposium on Computer Architecture. Toronto, Canada. 1991, 28(5): 266～275

（下转第28页）



A级BT的jacz过程(4节点)　　B级BT的jacz过程(8节点)

图5 (时间单位:秒)

结论及将来的工作 本文讨论了通过发掘节点间流水,有效提高程序并行性和降低通信开销的技术,它对于解决规则的数组引用问题,具有普遍的积极意义。

流水循环中粒度的选取是一个重要课题,我们现在的方法是采用静态工作集估算的方法,粒度的选取不一定是最佳的。将来的工作包含:采用运行时流水粒度选择,处理更复杂的数据分布形式、支持更复杂的循环分割形式及如何使目标代码更为简洁。

参 考 文 献

1 Hiranandani S, et al. Evaluation of Compiler Optimization for FortranD on MIMD. In: 1992 Intl. Conf. on Supercomputing, Washington, D.C, July 1992

2 Banerjee P,et al. An Overview of the PARADIGM Compiler for Distributed Memory Message-Passing Multicomputers. IEEE Computer, Oct. 1995. 37～47

3 Lowenthal D K, James M. Run-Time Selection of Block Size in Pipelined Parallel Programs. Second Merged IPPS/SPDP Symposium,1999. 82～87

4 Van der Wijngaart R F, et al. Analysis and Optimization of Software Pipeline Performance on MIMD Parallel Computers. Journal of Parallel and Distributed Computing, 1996

5 Wolfe M. High Performance Compilers for Parallel Computing. Addison-Wesley Reading, MA, 1996

6 Callahan C D. A Global Approach to the Detection of Parallelism: [PhD thesis]. Dept. of Computer Science, Rice Univ., Mar. 1987

7 McKinley K S,Carr S,Tseng C-W. Improving Data Locality With Loop Transformations. ACM Transactions on Programming Languages and Systems,1996,18(4)

8 Singhai S K,McKinley K S. An Algorithm for Improving Parallelism and Cache Locality. The Computer Journal,1997,40(7)

9 Manjikian N,Abdelrahman T S. Fusion of Loops for Parallelism and Locality. IEEE Transaction on Parallel and Distributed Systems,1997,8(2)

10 Abdelrahman T, et al. Locality Enhancement for Large-Scale Shared-Memory Multiprocessor. Languages, Compilers, and Run-Time Systems for Scalable Computers 4th International Workshop, LCR'98,LNCS 1511,Springer-Verlag, 1998

11 Lim A W, et al. Maximizing Parallelism and Minimizing Synchronization with Affine Partitions. Parallel Computing, 1998, 24(3-4):445～468

(上接第11页)

7 ReaCT-ILP laboratory. Trimaran: An Infrastructure for Research in Instuction-level Parallelism. http://www.trimaran.org

8 Gyllenhaal J C. A Machine Description Language for Compilation: [Master Thesis]. University of Illinois at Urbana-Champaign, 1994

9 Fraser C W,Hanson D R. A Retargetable C Compiler: Design and Implementation. Benjamin/ Cummings Pub Co, Redwood City, CA, USA, 1995

10 Gough J. Bottom up Tree Rewriting with MBURG: The MBURG Reference Manual. ftp.fit.qut.edu.au. 1995

11 Emmelmann H, Schroer F W, Landwehr R. BEG－A Generator for Efficient Back Ends. In:Proc. of the SIGPLAN'89 Conf. on Programming Language Design and Implementation, SIGPLAN Notices, 1989, 24(7): 227～237

12 Fraser C W,Hanson D R,Proebsting T A. Engineering a Simple, Efficient Code Generator Generator. ACM Letters on Programming Languages and Systems, 1992, 1(3): 213～226

13 Gao G R, Amaral J N, Dehnert J,Towle R. The SGI Pro64 Compiler Infrastructure: A tutorial. The International Conference on Parallel Architecture and Compilation Techniques (PACT2000), Oct. 2000

14 Ganapathi M,et al. Affix Grammar Driven Code Generation. ACM Transactions of Programming Languages and Systems, 1985, 7(4): 560～599

15 Moona R. Processor Models for Retargetable Tools. In: Proc. of 11th Intl. Workshop on Rapid System Prototyping, IEEE, 2000. 34～39

16 Wilson R P,et al. SUIF: An Infrastructure for Research on Parallelizing and Optimizing Compilers. ACM SIGPLAN Notices, 1994, 29(10): 31～37