微处理器体系结构模拟器 SimpleScalar分析与优化*

金立忠[1]，窦　勇[2]

(1.东北电力学院,吉林 132012; 2.并行与分布处理国家重点实验室，湖南 长沙 410073)

摘　要：Wisconsin-Madison大学发布的 SimpleScalar模拟器为处理器体系结构设计提供了多层次的支持。分析了整个模拟器的结构和工作流程,并通过测试分析得到了优化方法,该方法通过对模拟器编译配置的改进,能够缩短模拟时间 50%左右,大大提高了工作效率。

关键词：模拟器；性能优化；程序配置

中图法分类号：TP332　　**文献标识码**：A　　**文章编号**：1001-3695(2006)08-0197-02

Analysis and Optimization for Processor Architecture Simulator (SimpleScalar)

JIN Li-zhong[1], DOU Yong[2]

(1. Northeast China Institute of Electric Power Engineering, Jilin 132012, China; 2. National Laboratory for Parallel Distributed Processing, Changsha Hunan 410073, China)

Abstract: The SimpleScalar tool set, a collection of publicly-available simulation tools published by University of Wisconsin-Madison, is a great convenience for computer architecture designers. We analyze the structure and work-flow of SimpleScalar. A new method is proposed for shortening the simulation time. During the tests of 6 benchmark programs, the new method decrease the simulation time to nearly 50% by reconfiguring the makefile of the simulator and some parameters.

Key words: Simulator; Performance Optimization; Program Configuration

处理器设计是一项非常耗费资金和时间的复杂工程,特别是在集成度越来越高的今天,面对数千万乃至上亿规模的晶体管,那种设计硬件原型－实现－进行评估－改进－再实现的模式早就被摈弃了;另一方面,数学形式的性能模型在 Cache、乱序执行和前瞻等技术被广泛采用的情况下并不能精确地反映实际情况。处理器设计人员都是通过模拟器进行性能的预先评估和正确性验证,因此处理器体系结构模拟器对芯片设计非常重要。体系结构模拟器均采用软件模拟指令运行的工作方式,虽然其灵活性高,但是效率低。大型测试程序往往需要运行十几个小时甚至几天的时间。因此如何提高模拟器的运行效率至关重要。对于正在开发的模拟器需要采取多种并行措施和检查点保护等方法提高模拟器的执行速度和可靠性。但是如何提高已经使用的模拟器效率,其相关的研究工作不是很多[3,4]。

Wisconsin-Madison大学发布的 SimpleScalar模拟器是一个开放软件[1],源代码是公开的,具有良好的可移植性和可扩展性,能够支持各种不同层次设计人员的需求,因而得到了广泛的应用。现在普遍使用的是 1997年 1月份发布的 SimpleScalar模拟器第二版。本文的主要目的就是分析 SimpleScalar模拟器主程序和两个模拟器 Sim-fast及 Sim-safe的结构,并对其 Makefile中的编译优化参数进行分析。研究 Sim-fast子模拟器 use-jump-table参数的作用,并通过基准程序测试进行了性能比较,给出了优化的方法。

收稿日期：2005-04-15; **修返日期**：2005-09-23

基金项目：国家自然科学基金资助项目(90307001);霍英东教育基金资助项目

1　SimpleScalar模拟器简介

针对不同层次设计人员的需求,SimpleScalar提供了五个不同功能的模拟器:SimFast、SimSafe、SimProfile、SimCache和 SimOutOrder,极大地方便了研究人员的工作需要。同时,SimpleScalar采用了基于 GCC的编译器和相关的工具以产生适合自身运行的目标代码。图 1表示的是 SimpleScalar的框架。

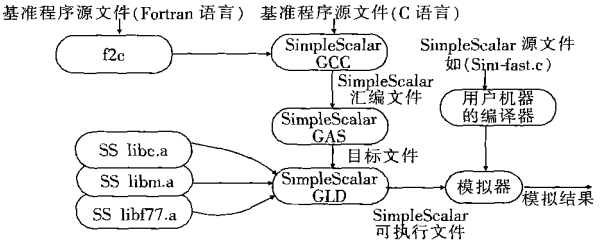

图 1　SimpleScalar 框架

其中 f2c是 AT&T的编译器,用于将 Fortran程序转换成适合 GCC的中间代码;而 SimpleScalar所使用的库是由 GLIBC移植而来的。

2　SimpleScalar模拟器主程序和子模拟器结构、流程分析

2.1　模拟器主程序基本流程

SimpleScalar主程序的工作过程如图 2所示。

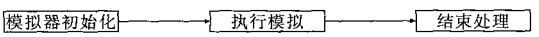

图 2　模拟器工作流程

(1)模拟器执行的入口点是 Main.c文件中定义的 Main()

主函数。当模拟器开始执行时,首先执行模拟器初始化工作,它主要包括:① 显示版本信息。②处理用户对模拟器的配置选项。SimpleScalar模拟器提供了许多选项供用户设置,如配置 Cache的大小及结构、前瞻执行策略的选择等,从而方便用户得到关心的信息。③存储器的初始化。SimpleScalar模拟器提供了 2GB的虚存空间,用来存放被模拟程序的正文段和数据段等。执行初始化后,被模拟程序的目标代码全部载入模拟器的内存中。④寄存器初始化。SimpleScalar提供了 32个整数、32个单精度浮点 /16个双精度浮点寄存器,另有六个控制寄存器。⑤相应子模拟器的初始化。五个子模拟器的初始化复杂程度不一,实现的功能也不尽相同。

(2)执行模拟。运行时分别调用各个子模拟器的 sim_main()函数:首先将被模拟程序的全部正文段译码,然后从程序的入口点处开始执行模拟。在下面详细介绍 Sim-fast和 Sim-safe模拟器的实现。

(3)执行完毕后进行结束处理,并将模拟过程中依据用户的选项配置记录的执行信息,如执行时间、各种类型指令的分类统计、前瞻执行的命中 /不命中情况等结果输出。

2.2 Sim-fast的基本流程

Sim-fast是执行速度最快,最不关心模拟过程细节信息的子模拟器程序。它采用顺序执行指令的方式,没有指令并行;不支持 Cache的使用,也不进行指令正确性检查,由程序员保证每条指令的正确性;不支持模拟器本身内嵌的 Dlite! 调试器 (类似于 gdb调试器)。

为了模拟器的速度优化,在缺省情况下,Sim-fast模拟器不进行时间统计,不对指令的有关信息 (如指令总数及访存指令数目)进行统计。当然,可以修改模拟器源程序,通过改变其设置,使模拟器更加符合设计人员的需求。

为保证模拟器的正常执行,能够加快执行速度的 jump_table参数被忽略。此参数仅在 Sim-fast子模拟器中才有,它用一数组存储跳转指令的目的地址以节省时间、提高速度。在文章的下一部分有 jump_table参数作用的详细讨论。

Sim_main()程序是每个子模拟器运行的核心部分。Sim-fast中 Sim-main()函数的主要工作是:①从模拟器的内存空间中取一条在前面主程序中存入的指令;②分析指令代码;③执行指令,并在执行过程中,依据用户的配置,将用户关心的信息进行统计;④循环执行步骤① ~步骤③,直到被模拟的程序执行结束;⑤跳转返回到模拟器的主程序。

2.3 Sim-safe的基本流程

Sim-safe是 Sim-fast的孪生兄弟,实现基本与 Sim-fast一致,但它们又是相互独立的。Sim-main()函数流程与 Sim-fast中的 Sim-main()函数大体相同,主要的区别在于在执行指令时,Sim-safe进行指令的齐整性 (Alignment)检查,对所有的访存指令首先检查是否合法,而且 Sim-safe增加了访存指令的分析,并支持 Dlite! 调试器。

3 SimpleScalar模拟器配置参数的优化

为了保证模拟器在所有运行平台上的正常执行,SimpleScalar模拟器的缺省配置均比较保守,这样的代价是运行速度很慢。表 1给出了 SPEC95基准程序的概要说明[2],它是测试所采用的 SPEC95基准程序功能和测试输入集的说明。

表 1 SPEC95基准程序

	基准程序功能说明	测试输入集
applu	抛物线 /椭圆偏微分方程求解 (CFP95)	applu. in
compress	在内存中压缩 /解压缩文件 (CINT95)	bigtest. in
fpppp	量子化学方程求解 (CFP95)	natoms. in
ijpeg	图像压缩 /解压缩 (CINT95)	penguin. ppm
li	LISP解释器 (CINT95)	ctaks. lsp
wave5	等离子体物理学:电磁粒子模拟 (CFP95)	wave5. in

SimpleScalar模拟器的优化主要有两部分:① Makefile中优化参数的设置,这些设置对所有的模拟器均有效;②在各个子模拟器的源程序中优化参数的修改,因此优化的效果仅作用于相应的子模拟器。

Makefile中优化参数缺省设置为 OFLAGS= -gWall即支持调试和对一些事件输出警告信息;优化后的设置为 OFLAGS= -O3-Wall-funroll-loops-m486,即允许为了优化而多花费一些编译时间,支持循环展开,面向 486而不是 386的体系结构进行编译优化。

在 Sim-fast中的 USE_JUMP_TABLE参数缺省是忽略的,为了说明该参数的作用,我们对忽略和允许此参数时的运行时间都进行了测试。测试平台为

CPU: AMD Duron 600;主板: Slotek SL-75KV (SOCKRT A),支持 DMA 66;

MEM: KingMax-133 128MB;硬盘:希捷 酷鱼 30GB 7200转;

OS: RED HAT Linux 7.0。

由于 Sim-fast是顺序执行的,为保证测试数据的可靠性和有效性,在单用户、单任务的环境下进行测试,所以最终的模拟执行时间可以认为是被测程序实际所耗费的时间。图 3是测试所得数据的比较 (时间单位是 s)。

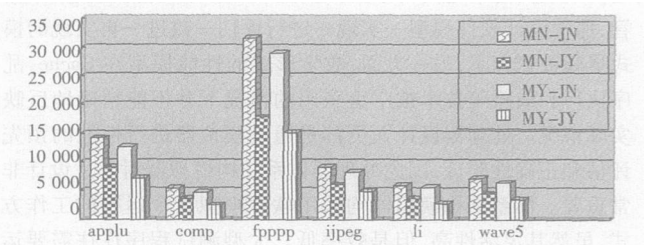

图 3 四种策略运行时间比较(s)

表中缩写的意义是:

MY——在 Makefile中采用优化编译参数;

MN——在 Makefile中采用缺省编译参数;

JY——在 Sim-fast中使用 USE_JUMP_TABLE参数;

JN——在 Sim-fast中忽略 USE_JUMP_TABLE参数。

由图 3可以看出,当使用 SimpleScalar的缺省配置时,全部基准程序耗费的时间均比其他情况多;在 Makefile中使用优化方法后,取得的效果比仅改动子模拟器的优化参数时要好。虽然是对 Sim-fast子模拟器进行的测试,但 Makefile中的优化参数对其余几个子模拟器的速度提升也同样起作用。当两种优化策略都使用时,获得的收益更加明显,能够缩短一半的测试时间。因此在操作系统支持下,应尽量使用优化设置,从而可以降低时间开销,加快设计开发的进度。

4 小结

SimpleScalar模拟器为处理器设计人员提供了强大的支持,但是由于模拟执行,其运行速度较慢,导致设计 (下转第 202页)

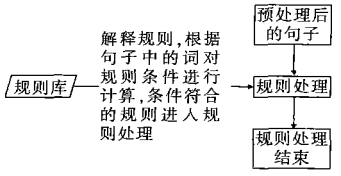

图1 单字动词规则处理流程图

表1 上述六条规则测试结果

规 则	句 数	正确率(%)
规则一	42	92.9
规则二	11	66.7
规则三	22	95.5
规则四	27	96.3
规则五	13	100
规则六	9	55.6
合计	124	89.5

从表1可以看出,在所测试的语句中基本能够将所能出现的情况正确处理,但也出现了错误的结果。存在错误是因为未能引入其他相关的知识,当把单字处理规则与整个句类分析过程融汇在一起时就能将错误消除。

对错误发生的原因进行分析研究,将错误主要分为三类:

(1)对不符合的情况却用单字动词处理规则进行了处理。

例如对规则三进行测试,下面这一句不符合单字动词组合处理规则:

范志毅飞脚踹向小将引发嘘声一片。

程序将它作为符合规则三来理解,实际上并不能这样简单地处理。在这个句子中,"飞"、"脚"、"踹"等均是单字词,"飞脚"一般并不把它作为一个双字词收录在知识库中,因此在该句中应该首先优先组合"飞脚",这属于单字词的研究,而本文目前只是对单字动词进行研究,这种情况将会在对单字词(包括单字非动词)进行研究的时候解决。这也正是笔者下一步的工作。

(2)对符合的情况却没有应用单字动词处理规则。

例如还是对规则三进行测试,下面这一句是应该用规则来进行处理:

这伙歹徒手持马鞭刀、铁棍……

程序并没有理解它。这是因为在进行初步切分时,"徒手"是一个双字词,这样初步分段处理字串后,"持"的前面并不是一个单字"手",而是双字词"徒手",其本质就是因为字串"歹徒手"有切分模糊。不过这个模糊在整个句类分析系统中是可以排除的。在进行句类假设验证时,能够消解"歹徒手"这一字串的切分模糊,排除"徒手",然后再运用单字动词处理规则,则可以得到正确的结果。

(3)组合后的结构含有的特定的意义

例如对规则一进行测试,下面例句中的"染红"包含了特定的意义。

杨璞再次染红。

该例句中的"染红"特指足球运动员在足球比赛中得到红牌。对于这种情况必须需要一些常识性的知识才能让计算机理解。单从概念基元符号体系与句类体系并不能理解其真正含义。

5 结束语

单字词的意义复杂,义项众多。对于单字动词来说,这势必造成其句类代码也存在多种情况。单字词的组合能力很强,这为单字词的处理提供了重要的线索。当单字动词与其他词组合出现时,该语句的特征语义块一般不能再单纯地由单字动词来感知,而一般由组合后的结构来感知,这时该语句的句类不同于由单纯的一个单字动词担任特征语义块的情况。本文对汉语中单字动词动态概念组合的情况,在HNC的概念基元体系与句类体系的基础上进行了探索,总结形成了单字动词组合处理规则;并在BNF范式和产生式规则的基础上,对规则进行了形式化,使规则能够被计算机理解与执行。在目前的实验结果中,出现了约10%的错误结果,错误产生的原因是因为我们没有将单字动词处理模块与其他句类分析模块结合起来。当单字动词的处理融汇在句类分析系统当中时,目前存在的一些错误将逐一被消除。

参考文献:

[1] 苗传江. HNC(概念层次网络)理论导论[M]. 北京:清华大学出版社, 2005.

[2] 黄曾阳. HNC(概念层次网络)[M]. 北京:清华大学出版社, 1998.

[3] 张克亮. 面向机器翻译的汉英句类及句式转换研究[D]. 北京:中科院声学所, 2004.

[4] 马希文. 与动结式动词有关的句式[J]. 中国语文, 1987, (6): 424-441.

[5] 杨丽君. 动词"搞"在现代汉语中的语用考察[J]. 语言文字应用, 2002, (2): 59-66.

[6] 晋耀红. 基于HNC理论的句类分析系统的设计与实现[D]. 北京:中国科学院声学研究所, 1998.

[7] 刘群,俞士汶. 汉英机器翻译的难点分析[C]. 1998中文信息处理国际会议论文集,北京:清华大学出版社, 1998. 507-514.

作者简介:

孙雄勇(1978-),男,湖南人,博士研究生,研究方向为自然语言理解、机器翻译、HNC(概念层次网络)理论及技术;张全(1968-),男,陕西人,研究员,博导,研究方向为HNC自然语言处理理论及相关技术、语言知识的表示与获取及处理、机器翻译、信息检索与信息抽取等。

(上接第198页)测试时间很长,效率不高。本文针对该模拟器的配置与优化方法进行了测试,通过对多个测试程序不同优化配置的分析比较,提出了模拟器配置的优化方法(MY-JY),能够提高模拟器执行速度近一倍,显著提高了处理器的设计效率。

参考文献:

[1] D C Burger, T M Austin. The SimpleScalar Tool Set(version 2.0)[R]. Technical Report CS-TR-97-1342, University of Wisconsin-Madison, 1997.

[2] A KleinOsowski, J Flynn, et al. Adapting the SPEC 2000 Benchmark Suite for Simulation-based Computer Architecture Research[C]. Proceedings of the International Conference on Computer Design, 2000.

[3] T Lafage, A Seznec. Choosing Representative Slices of Program Execution for Microarchitecture Simulations: A Preliminary Application to the Data Stream[Z]. Workload Characterization of Emerging Applications. Kluwer Academic Publishers, 2000.

[4] M Oskin, F T Chong, M Farrens. HLS: Combining Statistical and Symbolic Simulation to Guide Microprocessor Designs[C]. The 27th Annual International Symposium on Computer Architecture, 2000. 71-82.

作者简介:

金立忠(1967-),男,吉林人,环境与轻化工程系主任,讲师,学士,研究方向为环境与轻化工程;窦勇(1966-),男,吉林人,实验室主任,博导,博士,研究方向为计算机体系结构。