SimpleScalar 模拟器内核分析及应用

陈剑龙，傅忠传，崔 刚

(哈尔滨工业大学 计算机科学与技术学院，黑龙江 哈尔滨 150001，E-mail：fuzhongchuan@hit.edu.cn)

摘 要：为了探讨以 SimpleScalar 作为基础框架的修改和扩展空间，对 SimpleScalar 整体结构进行了描述，详细分析了其最复杂的 Out－of－Order(乱序)模拟器的内核结构，通过基于 SimpleScalar 的应用实例，提出了改进意见.

关键词：性能模拟；计算机系统建模；计算机体系结构

中图分类号：TP391.9 **文献标识码**：A **文章编号**：0367－6234(2004)05－0652－03

Kernel analysis and application of Simplescalar simulator

CHEN Jian-long，FU Zhong-chuan，CUI Gang

(School of Computer Science and Technology，Harbin Institute of Technology，Harbin 150001，China，E-mail：fuzhongchuan@hit.edu.cn)

Abstract：SimpleScalar is a kind of computer system simulation and modeling tool which that has been used widely in the computer architecture research field. The source code of SimpleScalar is open，which makes it easy to modify and port. This article first gives a general description to SimpleScalar；then gives a detailed analysis to the kernel structure of the complicated Out-of-Order simulator and an example based SimpleScalar and finally there is some advice to improve it.

Key words：performance simulation；computer system modeling；computer architecture

计算机系统的软件建模可分为功能模拟和性能模拟两种.建立性能模拟器的目的是为处理器设计和体系结构研究提供一个灵活有效的设计研究平台，其主要功能是在硬件实现前评测各种设计方案的性能及功耗，找出设计瓶颈并加以改进.现代微处理器的设计流程是一个反复迭代的过程，性能模型为后续设计步骤提供了一个规范，后续设计的变更都必须通过性能模型的验证，处理器的性能建模是整个设计的关键.本文对是计算机系统的性能建模进行了探讨.

1 SimpleScalar 工具集介绍

计算体系结构研究者和处理器设计者需要在真正的硬件实现前对其设计进行性能及功耗的验证评测.SimpleScalar 工具集的出现满足了这种需求，为计算机系统性能及功耗分析、处理器微体系结构建模、软硬件协同验证提供了有效的支持.SimpleScalar 既提供了简单的功能模拟器，也提供了模拟超标量处理器微体系结构的乱序(Out-of-Order)性能模拟器，其乱序模拟器支持动态指令调度、指令乱序执行、指令预测执行、分支预测等现代微处理器的特性，而且还提供了一系列的工具，包括编译器、Benchmark、调试工具、流水线跟踪器等，为计算机体系结构的研究提供了全面的支持，目前它已经支持 PISA、ARM、X86 等指令集.由于 SimpleSalar 的开放性，许多研究项目都采用它作为研究平台并扩展其功能[1].

2 SimpleScalar 模拟器软件架构

图 1 为 SimpleScalar 模拟器的软件架构图，模拟器采用分层模块化的组成结构.SimpleScalar 通过采用指令驱动技术(即由目标指令仿真器对每一条指令进行解释，通过解释的结果驱动性能模拟器)将功能模拟和性能模拟结合起来协同工作，加快了处理器误预测状态的恢复.对于 Syscall 系统调用 I/O 指令，SimpleScalar 将其转换成对操作系统的系统调用，并将结果返回给相应指令.Sim-

收稿日期：2003－03－14.
作者简介：陈剑龙(1978－)，男，硕士研究生；
崔 刚(1947－)，男，教授，博士生导师.

pleScalar 提供了从简单的功能模拟器 Sim-safe 到复杂的 Sim-outorder 性能模拟器. Simulator core 模块定义并实现了各个模拟器的行为方式，其余各个模块由各个模拟器共享，各个模拟器可根据自己的需要调用各个模块. 这意味着如果要设计自己的模拟器，主要工作就是修改 Simulator core 模块. SimpleScalar 提供的是一个建模框架，具有一定灵活性. 每个模块具体由. h 和. c 文件构成，其中. h 文件定义了模块所需的数据结构和可共调用的函数声明，. c 文件包含函数功能的实现. 其模块一类是计算机系统的组成模块；一类是辅助模块. 以下是模拟器主程序部分主要源代码：

```
main(int argc, char **argv, char **envp)
{sim-check-options(sim-odb, argc, argv);//检验命令行参数，并对每个模块初始化，sim-*.c 定义
    mem-init();//程序加载前 Memory 初始化，mem.c 定义
    ld-load-prog();//程序加载，loader.c 定义
    regs-init();//寄存器初始化，regs.c 定义
    mem-init1();//程序加载后 Memory 初始化，mem.c 定义
    sim-init();//模拟器初始化，sim-*.c 定义
    sim-main();//模拟器执行入口函数，sim-*.c 定义
}.
```

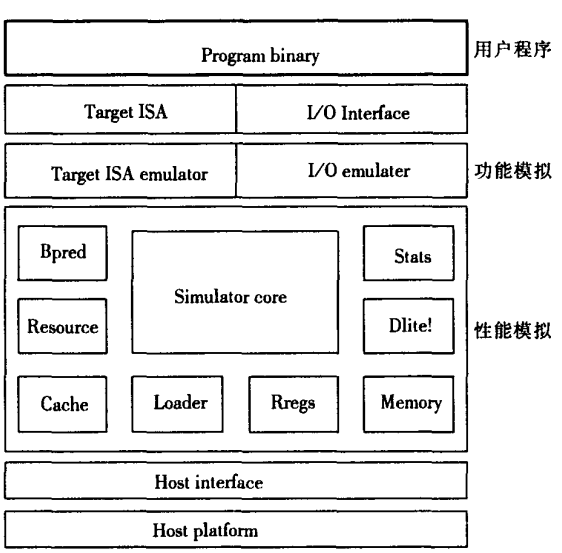

图 1　SimpleScalar 软件架构图

3　Out-of-Order 模拟器内核分析

Out-of-Order 模拟器是提供了足够微体系结构细节的性能模拟器，是支持指令动态调度乱序执行的超标量模拟器，在 2.0 版的 SimpleScalar 中只为其提供了 PISA 一种指令集[2]（4.0 版还支持 ARM、X86、Alpha 指令集）. Out-of-Order 模拟器主函数 sim-main()的程序结构表示如下：

```
void sim-main(void)
{
    for (;;) {ruu-commit();//提交
    ruu-writeback();//写回
    lsq-refresh();//LSQ 队列刷新
    ruu-issue();//执行
    ruu-dispatch();//发射
    ruu-fetch();//取指
    sim-cycle++;//模拟器周期，用于统计程序执行周期数
    }
    }.
```

for 循环执行一次代表流水线执行一个机器周期，每个 ruu-*()函数代表流水线的一段(stage)，采取反向执行顺序是为正确实现流水线各段之间的互锁同步. 图 2 为 Out-of-Order 模拟器流水线示意图.

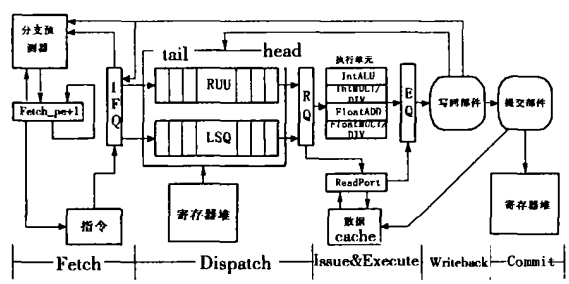

图 2　Out-of-Order 模拟器流水线框图

3.1　模拟器主要数据结构

RUU(Register Update Unit)：此结构将传统的保留站和排序缓冲站合并，实现了指令的乱序执行按序提交，用于支持处理器精确中断和误预测的状态恢复. RUU 将这两个部件合并，可以减少硬件实现的费用. RUU 被组织为循环队列，在指令发射阶段，按取指先后顺序为每条指令分配 RUU 单元，指令在 RUU 中连续存放并且排列顺序与取指顺序一致，这使执行完成的指令能够按序提交[3].

LSQ (Load/Store Queue)：用于处理 Store/Load 指令，其结构与 RUU 一致，一条 Store/Load 指令将被分成两个操作，地址计算和存取操作，地址计算操作作为加法运算放入 RUU 中，而存取操作放入 LSQ 中，RUU 和 LSQ 需要同步.

IFQ (Instruction Fetch Queue)：用于存放取指(Fetch)段取出的指令队列.

RQ (Ready Queue)：如果指令执行所需所有操作数已准备好，则将指令放入此队列准备执行.

EQ (Event Queue)：该队列记录已发射(Issue)指令何时执行完毕，wirteback 段以此确定何时将指令执行结果写回.

3.2 Out-of-Order 模拟器流水线分析

3.2.1 Fetch 段

此段的操作由 ruu-fetch()函数实现，这个段的任务是从指令 Cache 取指令并放入 IFQ 中，对于分支指令，还要访问分支预测器以确定下条指令内存地址.具体过程如下：

while(IFQ 未满 && 已取指令数小于取指宽度){

1) 根据 Fetch-pc 从指令 cache 取出一条指令；

2) 如指令 cache 发生缺失，则取指阶段结束，否则将取出指令放入 IFQ 中；

3) 确定下条指令的地址(对分支指令，访问分支预测器)；

}.

3.2.2 Dispatch 段

由函数 ruu-dispatch()实现，这个阶段的任务是指令解码仿真，寄存器重命名，RUU/LSQ 分配，将所有源操作数已准备好的指令放入 RQ 中.由于采用了指令驱动技术，模拟器可以判断指令是否处于误预测状态，对处于误预测状态的指令，仿真时其访存和写寄存器操作会映射到预测 buffer 中，而不会访问真正的物理寄存器和存储器，其具体过程表示如下：

while(RUU/LSQ 未满 &&IFQ 不空 && 已分配指令数小于解码宽度){

1)从 IFQ 中取出一条指令；

2) 解码该指令并仿真(此时模拟器可能处在误预测状态)；

3)为指令分配 RUU 单元，进行寄存器重命名(如果指令为 Store/Load 指令，则将指令分为计算地址操作和访存操作，分别分配 RUU 和 LSQ 单元)；

4)如果该指令所需源操作数指令已准备好，指令放入 RQ；

5)根据 2)指令仿真结果判断模拟器是否进入误预测状态；

}.

3.2.3 Issue&Execute 阶段：

由 ruu-issue()函数实现，为处在 RQ 队列中的指令分配执行所需资源并执行指令(具体过程略).

3.2.4 Writeback 段

由 ruu-writeback()和 lsq-refresh()函数实现.ruu-writeback()任务包括误预测状态的恢复和解决数据冒险(其执行过程略).lsq-refresh()函数的任务是将 LSQ 中满足下列条件的 Load 指令放入 RQ(过程略).

3.2.5 Commit 段

由函数 ruu-commit()实现，将执行完成的指令按序提交并升级处理器状态，(具体过程略.)

4 基于 SimpleScalar 的应用实例

随着嵌入式和便携式计算机系统的发展，处理器的功耗也成为处理器设计必须要考虑的关键问题.现有的功耗分析工具大都在电路级进行分析，这意味着功耗分析工作只能在整个设计实现流程的末端进行.而人们需要在设计阶段就对未来产品的功耗进行评估，以便进行设计折衷选择，而且，处理器的性能与功耗往往是矛盾的，处理器设计必须综合考虑性能与功耗，这对于嵌入式处理器尤其重要.高层体系结构级(Architectural-level)的功耗分析成为目前一个研究热点，本文介绍的 Wattch[4] 是一个基于 SimpleScalar 可进行体系结构级功耗分析和优化的软件工具.

图 3 为 Wattch 的总体结构图.Wattch 的设计思想是，首先建立超标量处理器的各个组成模块的功耗模型，组成模块包括 Cache、寄存器堆、分支预测器、重排序缓冲、各功能单元、控制电路、时钟等等.然后将功耗模型集成到性能模拟器中，每个时钟周期性能模拟器都将跟踪纪录处理器各组成模块的使用情况，并提供给功耗模型进行分析处理，从而既可评估每个组成模块的功耗也可评估整个处理器的功耗.功耗模型建立见文献[4].

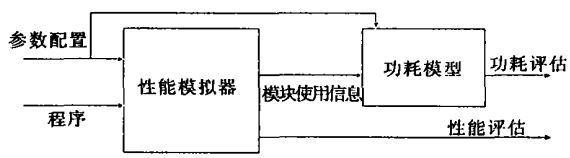

图 3 Wattch 总体结构图

实现 Wattch 需要一个具备以下特点的性能模拟器：具有高细节度的微体系结构；便于修改，如流水线的段数；各组成模块可参数化，如 Cache 的大小、相联度等.SimpleScalar 具备以上特点，使其较易集成到 Wattch 中，开发者可以将主要精力放在功耗模型的建立上.通过真实电路和处理器

(下转第 696 页)

[3] KANNINEN M F, POPELAR C H. Advanced Fracture Mechanics[M]. Oxford:Oxford Univ Press, 1985.

[4] FAN Tianyou. Moving defect with irreversible deformation zone and criterion of nonlinear dynamic fracture of solids [J]. Chinese Physics Letters, 1986(3):405—408.

[5] FAN T Y. Moving Dugdale model[J]. Zeitschrift Fur Angewandte Mathematik and Physik, 1987, 38: 630—641.

[6] KNAUSS W G. Dynamic fracture[J]. Int J Frac, 1987, 25: 35—91.

[7] CHAREPANOV G P. Mechanics of Brittle Fracture[M]. Nauka Moscow:[s.n.], 1973.

[8] CHAREPANOV G P, AFANASOV E F, Some dynamic problems of the theory of elasticity—A review[J]. Int J Engng Sci, 1970,12: 665—690.

[9] ATKINSON C. The propagation of a brittle crack in anisotropic material[J]. Int J Engng Sci, 1987(3): 77—91.

[10] MUSKHELISHVILI N I. Singular Integral Equations [M]. Nauka Moscow:[s.n.], 1968.

[11] MUSKHELISHVILI N I. Some Fundamental Problems in the Mathamatical Theory of Elasticity [M]. Nauka Moscow:[s.n.],1968.

[12] 程 靳. 某些正交异性体弹性动力学问题[J]. 哈尔滨工业大学学报,1985(增刊):8—21.

[13] DUGDALE D S. Yielding of steel sheets containing slits [J]. J Mech Phy Soli, 1960,18: 100—104.

[14] BILLY B A, COTTREL A H, SWINDEN K H. The spread of plastic yield from a notch[J]. Proc Roy Soc Series A, 1963,272: 304—314.

(**编辑** 刘 彤)

(**上接第 654 页**)

功耗的对比验证(过程见文献[4]),Wattch 的功耗分析结果与真实功耗相差在 10%以内.

5 结 语

SimpleScalar 提供的是一个建模框架,一个基线(baseline)模型,其设计目的在于为微处理器设计提供低层支持,这充分考虑到了设计人员的需要,因为处理器的微结构设计多种多样,一个专门为某处理器设计的模拟器有很大的局限性,所以其设计处处体现的都是可重用、模块化的思想.大部分研究项目也是将 SimpleScalar 作为基础框架(framework),然后对其进行修改和扩展,如本文介绍的 Wattch 及 MASE[5].基于此,本文认为其在以下几个方面有改进的空间:

1)微体系结构应符合现代处理器的设计思想,如其指令窗口目前采用 RUU,而现在几乎没有微处理器采用 RUU,所以可以改为通用的结构,如重排序缓冲和保留站.

2) 目前 SimpleScalar 采用 trace 驱动,指令并未在功能单元执行,这使得它无法模拟 time-dependent 计算,如值预测[6](Value Prediction).

3) 目前的存储系统,包括 memory 和 cache 过于简化,与真实的存储系统差异较大.

参考文献:

[1] AUSTIN T, LARSON E, ERNST D. SimpleScalar: An infrastructure for computer system modeling[J]. IEEE Computer, 2002,35(2):59—67.

[2] BURGER D, AUSTIN T. The SimpleScalar tool set, version 2.0[R]. [s.l.]:University of Wisconsin Computer Sciences Technical Report, 1997.

[3] SOHI G. Instruction issue logic for high-performance, interruptible, multiple functional unit, pipelined computers[J]. IEEE Transactions on Computer, 1990,39(3):349—359.

[4] BROOKS D, TIWARI V, MARTONOSI M. Wattch: A framework for architectural-evelPower analysis and optimizations[A]. Proc 27th Ann Int'l Symp Computer Architecture [C]. Los Alamitos:IEEE CS Press, 2000. 83—94.

[5] LARSON E, CHATTERJEE S, AUSTIN T. MASE: A novel infrastructure for detailed microarchitectural modeling[A]. Proc. IEEE Int'l Symp. Performance Analysis of Systems and Software[C]. Los Alamitos: IEEE CS Press, 2001. 1—9.

[6] CALDER B, REINMAN G, TULLSEN D. Selective value prediction[A]. 26th International Symposium of Computer Architecture[C]. [s.l.]:[s.n.], 1999.1—10.

(**编辑** 王小唯,杨 波)