

# 基于 SimpleScalar 的龙芯 CPU 模拟器 Sim-Godson

张福新 章隆兵 胡伟武

(中国科学院计算技术研究所系统结构重点实验室 北京 100080)

**摘 要** 现代高性能通用处理器的设计越来越复杂,模拟器在处理器设计中所起的作用越来越大.龙芯2号是中国科学院计算技术研究所研制的高性能通用处理器.最早开发的龙芯2号的模拟器 ICT-Godson 是信号级模拟器,它模拟了处理器的所有细节,十分准确,但速度和灵活性有较大限制.文章基于 SimpleScalar 工具集,设计并实现了龙芯2号的模拟器 Sim-Godson. Sim-Godson 具有高速度和高灵活性的优点,且准确性也很高.在 3.0GHz 的 Pentium4 微机上,Sim-Godson 速度约为 500K 指令/s.大部份测试程序在 Sim-Godson 上的 IPC(Instruction Per Cycle)与 ICT-Godson 相差不到 5%,达到了很高的准确性. Sim-Godson 在龙芯2号的性能分析工作中发挥了重要作用.

**关键词** 模拟器;龙芯2号处理器;SimpleScalar;通用处理器;高性能处理器

中图法分类号 TP302

## Sim-Godson: A Godson Processor Simulator Based on SimpleScalar

ZHANG Fu-Xin ZHANG Long-Bing HU Wei-Wu

(Key Laboratory of Computer System and Architecture, Institute of Computing Technology,

Chinese Academy of Sciences, Beijing 100080)

**Abstract** As the increase in complexity of modern processor design, simulator is becoming more and more important in processor design. Godson-2 is a high performance general purpose processor developed by the Institute of Computing Technology, Chinese Academy of Sciences. ICT-Godson, the initial simulator of Godson-2, is a signal level simulator, which simulates all hardware details of Godson-2 and is very accurate. But the speed of ICT-Godson is not high enough and the flexibility is not good. Based on SimpleScalar tool sets, Sim-Godson, one processor simulator of Godson-2, is designed and implemented. Comparing with the ICT-Godson, Sim-Godson has high speed and high flexibility and the accuracy is also very good. On PC of Pentium4 3.0GHz, the speed of Sim-Godson is about 500K instructions per second. IPC(Instruction Per Cycle) difference between Sim-Godson and ICT-Godson is less than 5 percent for the most applications. Sim-Godson plays an important role in the performance analysis of God-son-2.

**Keywords** simulator; Godson-2 processor; SimpleScalar; general purpose processor; high performance processor

收稿日期:2005-06-11;修改稿收到日期:2006-10-18. 本课题得到国家杰出青年基金(60325205)、国家自然科学基金(60673146)、中国科学院计算技术研究所基础研究基金(20056020)、中国科学院计算技术研究所知识创新课题(20056240)、国家“八六三”高技术研究发展计划项目基金(2002AA110010,2005AA110010,2005AA119020)、国家“九七三”重点基础研究发展规划项目基金(2005CB321600)资助.  
张福新,男,1976年生,博士,副研究员,主要研究方向为微处理器系统结构设计、性能评估和机群计算. E-mail: fxzhang@ict.ac.cn.  
章隆兵,男,1974年生,博士,副研究员,主要研究方向为微处理器设计、机群计算. 胡伟武,男,1968年生,博士,研究员,博士生导师,主要研究领域为高性能计算机体系结构、并行处理、VLSI设计.

## 1 引言

现代高性能通用处理器的设计越来越复杂,体系结构的探索空间越来越大,由专家根据直觉和经验来决定处理器结构已不可能,处理器体系结构的确定通常是一个团队进行性能分析、不断改进的结果<sup>[1]</sup>. 而性能分析的基础是处理器的模拟器. 龙芯 2 号是中国科学院计算技术研究所研制的高性能通用处理器<sup>[2]</sup>. 在龙芯 2 号的研制过程中,最早开发的模拟器是信号级的 ICT-Godson 模拟器. ICT-Godson 模拟了处理器所有细节,可以非常准确地给出处理器的行为数据,但是这同时也造成它的速度和灵活性受到比较大的限制. 对于性能分析的一些重要任务,例如设计空间探索、硅前性能预测等,使用 ICT-Godson 都显得很困难. 为了解决这些问题,我们基于著名的 SimpleScalar 工具集<sup>[3]</sup>①设计并开发了 Sim-Godson 模拟器. Sim-Godson 与 ICT-Godson 相比具有高速度和高灵活性的优点,且准确性高,在龙芯 2 号的性能分析工作中发挥了重要作用.

本文第 2 节是背景介绍,包括 SimpleScalar 和龙芯 2 号微体系结构简介;第 3 节是 Sim-Godson 的设计与实现;第 4 节是 Sim-Godson 的验证;第 5 节是评测;最后是结论和未来的工作.

## 2 背景介绍

### 2.1 SimpleScalar 简介

SimpleScalar 工具集是一个用于构建各种模拟程序的系统软件框架,可用于构建各种体系结构模拟器. 它提供了一套参考模拟器,包括快速的功能模拟器,用于分支预测评估的模拟器,用于 Cache 层次评估的模拟器以及一个详细的、动态调度的、多级存储层次的微体系结构模拟器. SimpleScalar 还包含一个机器定义框架,允许绝大部分体系结构细节和模拟器的具体实现分离. SimpleScalar 模拟器采用执行驱动,它的指令解释器支持多个流行的指令集,包括 Alpha、PowerPC、X86 和 ARM. 除了模拟器,SimpleScalar 工具集还提供统计分析、调试、验证和可视化支持. SimpleScalar 工具集被广泛用于计算机体系结构的研究和教学. 例如在 2000 年,顶级体系结构会议上超过三分之一的文章使用 SimpleScalar 来评价设计.

研究人员基于 SimpleScalar 开发了许多扩展工具和功能,包括功耗模拟器 WATTCH<sup>[4]</sup>、多线程模拟器 SIMCA、支持统计采样的 SMARTS 系统<sup>[5]</sup>、用于选择模拟样本的 Simpoint<sup>[6]</sup> 以及一个经 Alpha EV6 工作站验证过的模拟器 Sim-alpha<sup>[7]</sup> 等.

### 2.2 龙芯 2 号微体系结构简介

龙芯 2 号是实现 64 位 MIPS-like 指令集的 RISC 处理器. 每个时钟周期取 4 条指令进行译码,并且动态地发射到 5 个全流水的功能部件中. 四发射超标量的结构使得指令相关和数据相关问题更加突出,龙芯 2 号采用乱序执行和激进的 Cache 设计来提高流水线的效率. 乱序执行技术包括寄存器重命名、动态调度和转移预测技术. 龙芯 2 号有两个定点功能部件,两个浮点功能部件和一个访存部件. 浮点部件通过浮点指令的 fmt 域的扩展可以执行 32 位和 64 位的定点指令以及 8 位和 16 位的用于媒体加速的 SIMD 指令. 龙芯 2 号的基本流水级包括取指、预译码、译码、寄存器重命名、调度、发射、读寄存器、执行、提交等 9 级.

## 3 Sim-Godson 的设计与实现

Sim-Godson 的主要设计目标包括高速度和高灵活性. 为了达到这两个目标,Sim-Godson 采用功能模拟和时序模拟分离的执行驱动方式,使用独立的指令执行引擎来解释和执行指令,性能模型部分只维护确定指令流动时序所需要的微体系结构,不关心实际的数据. 例如,访问 Cache 时,性能模型只需要知道 tag 是否命中以计算访存指令的延迟,Cache 里不用存储实际的数据. 采用这种执行驱动方式的好处是:一方面使得 Sim-Godson 基本上只有控制通路,没有数据通路,可以减少大量的数据流动和处理操作,提高运行速度;另一方面,使得 Sim-Godson 易于实现和修改,为进行体系结构评价提供了较高灵活性.

另外,我们还采用一些软件实现上的优化技术来提供速度,包括①使用软件来缓存常用的数据,例如最近访问的 TLB 项、Cache 项等,降低查找开销;②指令在流水级间的流动用指针传递来完成,最小化数据复制;③模拟器中每条运行中的指令所有信息用一个数据结构表示. 系统初始化时分配足够的

① Burger D, Austin T M. The SimpleScalar tool set. 1997. <http://www.simplescalar.com>

指令信息结构,用一个链表保存.运行中该结构的分配和释放都只是对这个链表操作,既加快了速度,还可能提高 CACHE 利用率(同一块空间被重复利用).类似的结构缓冲方法被多处使用.就灵活性而言,我们还充分利用 SimpleScalar 工具集提供的框架结构,以较好的模块化形式提供了参数化部件、调试、统计、可视化等功能.

用 SimpleScalar 工具集实现龙芯 2 号性能模型的基础工作分为两部分:实现 Linux/MIPS 体系结构和实现龙芯 2 号微体系结构.

### 3.1 MIPS 体系结构支持

在 SimpleScalar 的架构下,实现对 MIPS 体系结构的支持主要包括 3 项工作:

(1) 实现目标机器定义. SimpleScalar 中机器定义包括目标结构的寄存器定义、指令译码和功能实现等. SimpleScalar 中现有的 PISA 指令集和 MIPS 指令比较相似,在 PISA 机器定义的基础上实现 MIPS 机器定义可以节省不少工作量.主要的工作包括实现 MIPS 指令的译码和定义、处理 MIPS 的转移指令延迟槽、处理 likely 类的转移指令等.

(2) 实现目标二进制文件装载. Linux/MIPS 的可执行文件是标准的 ELF 文件,装载过程和其它体系结构没有很大差别,只需要少量改动.

(3) 实现对目标操作系统调用的代理. SimpleScalar 采用一种被称为“代理(proxy)”的方法来处理应用程序中的系统调用.它首先从被模拟系统中得到目标系统调用的相关参数,然后调用主机操作系统的系统调用实现该功能,最后根据主机系统调用的结果设置被模拟系统的寄存器和内存,使得它看起来像刚完成一个系统调用. SimpleScalar 原来所“代理”的系统调用和 Linux/MIPS 系统调用有较大区别,需要大量的重新实现.主要的工作包括识别系统调用的类型和参数以及转换目标操作系统与主机操作系统的数据结构.

### 3.2 龙芯 2 号微体系结构实现

SimpleScalar 所提供的 Sim-outorder 模拟器实现了一个执行驱动的超标量处理器模型, Sim-alpha 模拟器也实现了一个 Alpha 21264 的详细性能模型.但是两者的处理器微体系结构和龙芯 2 号差别都比较大,无法直接采用. Sim-Godson 实现了龙芯 2 号处理器的微体系结构,只借鉴了 Sim-alpha 的组织方式和 SimpleScalar 的一些基础模块. Sim-Godson 的关键实现技术包括:

(1) 逆向调用流水级. 在 Sim-Godson 里,不需要维持模块接口和硬件的一致性,因此模块间的相互依赖可以通过移动处理逻辑来消除.例如,模块 A 要输出数据给模块 B,而是否被接受依赖于模块 B 送给 A 的允许信号,模块 A 要根据输出是否被接受来更新自己的寄存器.在 ICT-Godson 里,如果只调用一遍,模块 A 和 B 无论怎么安排都不能正确工作.在 Sim-Godson 里,只需要把 B 中允许信号的生成逻辑放到 A 中就能以先 A 后 B 的次序解决问题.消除这种向后的控制之后,流水级只向前传递数据,那么以和流水进行方向相反的次序调用流水级,就能保证流水级之间生成、使用数据次序正确.这个技术也为 Sim-outorder 和 Turandot 所采用,它避免了流水级管理的开销,能够显著地提高速度.

(2) 多重误预测支持. 为了模拟误预测路径上的指令执行,指令执行引擎需要做相应的配合. Sim-outorder 的做法是,一旦发现分支预测错误,指令执行引擎进入猜测模式,此时所有对体系结构状态(寄存器和内存)的改变都不直接生效,而是暂存在其它结构里(采用 Copy-on-write 方式).正确的分支结果出来后,猜测模式的结果被抛弃,指令执行引擎重新从正确路径开始执行.为了正确执行猜测路径上的指令,在猜测模式下,指令执行引擎读寄存器和访存操作需要相应改变:先查找猜测模式下的暂存数据,没有找到才去访问实际体系结构状态. Sim-outorder 只能支持一个猜测层次,在猜测模式下再发生猜测错误不会被更正.而实际硬件不可能知道目前是否在猜测模式下,只能一旦发现猜测错误就立即更正,这就要求我们支持多重误预测.为此 Sim-Godson 引入了错误层次,增强了指令执行引擎的体系结构状态读写接口,使它能支持多重误预测.

(3) 寄存器重命名. 龙芯 2 号处理器内部使用基于全相联查找的寄存器重命名方式,为定点和浮点寄存器各使用一个 64 项寄存器重命名表,每项对应一个物理寄存器.按照硬件方式实现要经常进行表的全相联查询,算法复杂度比较高.由于 Sim-Godson 的性能模型不使用实际的寄存器值,它不需要进行实际的重命名,而只关心寄存器值什么时候可用.这可以通过维护指令间的依赖关系来做到. Sim-outorder 对重命名寄存器的总数没有限制,相当于物理寄存器堆无限大.在 Sim-Godson 中我们按实际硬件情况维护空闲物理寄存器数目,在空闲

数小于 4 时停止重命名。

(4) 功能部件. 功能部件采用类似 Sim-out-order 的事件调度方式. 这样既有利于参数化功能部件个数、延迟等, 效率也比较高. 针对龙芯 2 号功能部件的行为, 我们做了一些修改, 使它能够在模拟可变延迟的功能部件、端口竞争等情况。

(5) 发射. Sim-Godson 精确模拟了龙芯 2 号的发射策略, 但采用复杂度比较低的软件实现方法。

(6) 访存. 我们对 Load/Store 队列的功能做了实现上的优化, 使它的复杂度大大低于硬件逻辑. 为了准确起见, 我们详细模拟了访存部件的各流水级, 而没有采用类似 Sim-outorder 和 Sim-alpha 的事件调度机制。

## 4 Sim-Godson 的验证

在 Sim-Godson 的验证中, 我们采用准确的 ICT-Godson 模拟器作为基准, 不断校准 Sim-Godson, 使得准确性高. 一般来说, 使用模拟器作为基准有几个好处. 首先这样可以进行更详细的比较: 除了运行时间或者 IPC, 还可以比较任意的内部统计数据, 如分支预测命中率、CACHE 命中率等. 其次, 使用模拟器作为基准可以避免在实际硬件中的各种随机因素, 便于准确测量. 再者, 未来 Sim-Godson 所模拟的处理器模型, 很可能还没有实际硬件存在。

另外, 在模拟器的验证中, 发现误差之后如何有效地定位模型实现问题是一个关键问题. 误差可能是由于忽略了一些重要的细节, 也可能是实现错误. 利用模拟器提供的数据收集功能, 我们可以从事件计数统计、流水级吞吐统计以及按指令的统计数据等多种角度进行分析. 由于我们使用一个准确的模拟器 ICT-Godson 作为参考, 所有这些数据都能够进行比较, 使得我们能够比较快地定位问题。

### 4.1 验证流程

我们采用两种类型的负载来对 Sim-Godson 进行验证, 包括微基准测试程序<sup>[8]</sup>和完整应用程序. 微基准程序一般是一个比较简单的程序(一般是一个循环), 专门设计来考察特定的微体系结构特性. 利用微基准测试程序进行验证, 可以考察每个微体系结构特性是否被正确地模拟, 从而完成对模拟器的一个系统验证. 另外, 由于微基准程序主要考察各部件的建模正确性, 实际程序的性能和各部件之间的交互密切相关, 因此仅靠它们不足以充分验证整个模拟器的正确性和准确性. 在微基准程序的误差得

到控制之后, 需要使用真实的完整应用程序(如 SPEC CPU2000)来进行更进一步的验证。

我们采用的 Sim-Godson 的验证流程如下:

1. 根据目标微体系结构特性设计微基准程序.
2. 用微基准程序验证模拟器.
3. 根据误差分析结果修正模拟器. 回到步 2.
4. 采用更实际的工作负载验证, 目前采用 SPEC CPU2000.
5. 分析验证误差:
  - 5.1. 如果迹象显示误差原因可能是某些特性模拟偏差, 回到步 1, 增加针对该特性的微基准程序.
  - 5.2. 根据误差分析结果修正模拟器. 回到步 2.

步 1 关键是, 需要尽可能全面地覆盖目标微体系结构特性. 经验表明, 很多由 SPEC CPU2000 发现的模型问题, 实际上都可以由一些简单的微基准程序发现. 而分析 SPEC CPU2000 的误差要比分析微基准程序困难得多. 步 3 和 5 的反馈是非常重要的. 我们不止一次发现, 修正了一个问题后, 某些程序误差很小了, 但另一些原先结果看似正确的测试程序却出现较大的偏差. 这是因为几个实现问题有时会互相掩盖. 进行了任何修改都应该重新验证所有的程序. 步 3 和 5 的误差分析工作是这个流程中最关键的部分. 面对验证的结果数据, 我们一方面要能作出正确的结论, 另一方面还需要从中有效地发现误差根源. 我们发现, 为了得到正确的结论, 要尽可能全面地分析程序运行的结果. 仅仅比较运行时间或者 IPC, 有时会得出错误的结论. 例如, 未经验证的 Sim-Godson 运行 SPEC CPU2000, 很多个程序的 IPC 误差小于 10%, 只有 *bzip2*, *crafty*, *twolf* 等几个程序误差较大. 但是, 该版本模拟器的浮点部件延迟、发射策略、分支预测等都还存在严重的问题. 如果我们检查其它统计数据, 如分支预测率等, 就会发现很多异常了。

Sim-Godson 验证所使用的微基准程序一共包括 19 个程序. 根据其功能, 这些程序可以大致地分为三类: 控制类、执行类、访存类. 除了访存类以外, 所有程序都驻留在 ICACHE, DCACHE 和 TLB 里, 以排除存储访问部件的影响. 控制类考察分支预测器的设计, 包括 6 个程序 (C-cond、C-recur、C-switch1、C-switch2、C-Switch3、C-Complex). 执行类考察功能部件和流水线设计, 包括 9 个程序 (E-I、E-F、E-D{1-6}、E-DM1). 访存类考察存储层次设计, 包括 4 个程序 (M-I、M-F、M-D、M-M). 例如: C-cond 是一个简单的 if-then-else 语句循环, 其分支指令按一次跳转、一次不跳转的规律变化. E-I 是一

系列不相关的整数加指令集合, E-F 是一系列不相关的浮点加指令集合, 它们用来考察功能部件组织和流水模拟的正确性. M-I 包含一系列不相关的 CACHE 命中的 load 操作. 它可以测试 DCACHE 的带宽.

由于龙芯 2 号主要针对桌面应用和低端服务器应用. 采用 SPEC CPU2000 基准测试程序集作为完整应用程序来验证 Sim-Godson 是合适的, 能够比较全面地考察处理器模型.

## 5 评 测

### 5.1 准确性测试

我们的测试平台是 3.0GHz Pentium 4, 内存 2GB, 硬盘 4GB. 我们采用 SPEC CPU2000 来测试模拟器的准确度, 主要以 IPC 作为指标. 处理器微体系结构配置如表 1 所示. SPEC CPU2000 程序运行时使用 test 输入集, 每个程序最多运行 10 亿条指令. 测试结果如表 2 所示. 从表 2 中可以看出, 除了 ammp(18%) 和 Twolf(6%), 大部分程序在 Sim-Godson 上的 IPC 与 ICT-Godson 相差不到 5%. 由于 ICT-Godson 与处理器的硬件行为完全一样, 并且龙芯 2 号处理器的逻辑设计的 RTL 代码就是按照 ICT-Godson 来写的, 因此 ICT-Godson 的测试结果从理论上来说与真实的处理器是一致的, 这就说明 Sim-Godson 达到了很高的准确性.

表 1 Sim-Godson 模拟器验证使用的微体系结构配置

特征	配置
Fetch; decode; issue; commit; Width	4; 4; 5; 4
功能部件	2 定点, 1 个访存, 2 浮点
Roqueue	32
定点发射队列	16
浮点发射队列	16
L1-ICACHE	64KB, 4 路组相联, 随机替换
分支预测器	Gshare, 9 位 ghr, 4096 项 pht, 128 项 BTB, 两路组相联, 随机替换
主流水级	7 级
功能部件 load-use 延迟	定点 ALU 2, 乘 4, 除可变 <sup>1</sup> , 浮点加 4, 乘 5, 访存(CACHE 命中)5
转移队列	8
存储访问队列	16
失效访问队列	2
L1-DCACHE	64KB, 4 路组相联, 随机替换
访存延迟	第一个子块返回延迟 50 拍, 子块间隔 2 拍

注 1: 定点乘法和除法内部被分成两个子操作. 除法的延迟和操作相关.

表 2 SPEC CPU2000 程序验证结果

(a)			
Prog(SPECfp)	ipc-ict	ipc-sim	ipc-err
Ampmp	0.24	0.28	0.18
Applu	0.61	0.62	0.02
Apsi	0.51	0.52	0.02
Art	0.15	0.15	0.03
Swim	0.45	0.46	0.01
Equake	0.82	0.83	0.01
Mesa	1.09	1.12	0.03
Mgrid	0.52	0.54	0.04
Wupwise	1.17	1.14	-0.03
Sixtrack	1.04	1.03	-0.01
(b)			
Prog(SPECint)	ipc-ict	ipc-sim	ipc-err
Mcf	0.41	0.43	0.03
Parser	0.75	0.76	0.01
Perlbmk	0.57	0.55	-0.04
Crafty	1.05	1.06	0.01
Twolf	0.84	0.89	0.06
Vortex	0.66	0.64	-0.03
Vpr1	0.97	0.97	0.01
Vpr2	0.68	0.69	0.01
Gap	0.83	0.84	0.02
Gcc	0.71	0.71	0.00
Gzip	0.81	0.82	0.01
Eon1	0.97	0.98	0.01
Eon2	0.97	0.98	0.01
Eon3	0.92	0.93	0.01

### 5.2 性能评测

对于模拟器的速度而言, 应用程序不同和所模拟的处理器微体系结构不同对模拟器的速度有一定影响. 但是对模拟器使用者而言, 模拟器的速度量级对其最重要, 影响其对模拟器的选择. 我们用 SPEC CPU2000 程序在 Pentium 4(3.0GHz) 上做了性能评测. 评测结果表明: ICT-Godson 的速度约为 50K 指令/s; Sim-Godson 的速度约为 500K 指令/s; SimpleScalar 中的 Sim-outorder 约为 700K 指令/s. 这个评测结果是易于理解的, ICT-Godson 与硬件行为完全一致, 模拟最多的细节, 具有最高准确性, 因此模拟速度慢; Sim-Godson 抽象层次较高, 模拟细节较小, 准确性较高, 速度较快; 而 Sim-outorder 模拟细节最少, 速度最快, 但准确性要比 Sim-Godson 差.

## 6 结论和未来的工作

本文在 SimpleScalar 和信号级模拟器 ICT-Godson 的基础上, 设计和开发了 Sim-Godson 模拟器. Sim-Godson 的速度比 ICT-Godson 快约一个数量级, 具有更高的灵活性, 且准确性几乎差不多. 在龙芯 2 号的性能分析工作中, 我们配合使用 ICT-Godson 和 Sim-Godson 模拟器, 发现了多个龙芯 2 号性能瓶颈, 并提出了解决方案. 例如: 定位了早期

版本的龙芯 2 号的存储系统瓶颈, 并提出相应解决方案, 获得很好的效果.

从目前测试来看, Sim-Godson 的速度比 Sim-outorder 约慢 30%, 这一方面是由于 Sim-Godson 比 Sim-outorder 模拟更多的细节; 另一方面是在 Sim-Godson 中为了与 ICT-Godson 校准, 存在许多冗余代码. 我们计划进一步优化 Sim-Godson 的代码, 提高速度及可读性. 最终的目标是完善 Sim-Godson 的文档, 并在互联网上公布. 另外计划将 Sim-Godson 与多处理器模拟器 SimOS 结合起来, 构成一个单片多处理器 (Chip Multi-Processor) 的模拟器.

### 参 考 文 献

[1] Pardip B, Thomas M C. Performance analysis and its impact on design. *IEEE Computer*, 1998, 31(5): 41-45

[2] Hu Wei-Wu, Zhang Fu-Xin, Li Zu-Song. Microarchitecture of the Godson-2 processor. *Journal of Computer Science and Technology*, 2005, 20(2): 243-249

[3] Austin T, Larson E, Ernst D. SimpleScalar: An infrastructure system modeling. *IEEE Computer*, 2002, 35(2): 59-67

[4] Brooks D, Tiwari V, Martonosi M. Wattch: A framework for architectural-level power analysis and optimizations//*Proceedings of the 27th Annual International Symposium on Computer Architecture*. Vancouver, BC, Canada, 2000: 83-94

[5] Wunderlich R E, Wenisch T F, Falsafi B, Hoe J C. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling//*Proceedings of the 30th Annual International Symposium on Computer Architecture*. San Diego, 2003: 84-95

[6] Sherwood T, Perelman E, Hamerly G, Calder B. Automatically characterizing large scale program behavior//*Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating System*. San Jose, 2002: 45-57

[7] Desikan R, Burger D. Sim-alpha: A validated, execution-driven Alpha 21264 simulator. Department of Computer Sciences, University of Texas at Austin, Technical Report; TR-01-23, 2001

[8] Desikan R, Burger D, Keckler S W. Measuring experimental error in microprocessor simulation//*Proceedings of the 28th Annual International Symposium on Computer Architecture*. Göteborg, Sweden, 2001: 266-277



**ZHANG Fu-Xin**, born in 1976, Ph. D. candidate, associate researcher. His research interests include microprocessor design, high performance computer architecture and operating system.

**ZHANG Long-Bing**, born in 1974, Ph. D., postdoctoral, associate researcher. His research interests include microprocessor design, system architecture, cluster computing.

**HU Wei-Wu**, born in 1968, Ph. D., professor, Ph. D. supervisor. His research interest includes high performance computer architecture, parallel processing and VLSI design.

### Background

As the increase in complexity of modern processor design, performance analysis of processor is becoming more and more important. Performance simulator of processor is essential to performance analysis. General purpose processor Godson-2 is developed by the group in the Institute of Computing Technology, Chinese Academy of Sciences. ICT-Godson, the initial simulator of Godson-2, is a signal level simulator, which simulates the all hardware details of Godson-2 and is very accurate. But the speed of ICT-Godson is not high enough and the flexibility is not good. Based on SimpleScalar tool sets, Sim-Godson, one performance simulator of Godson-2, is designed and implemented. Based on Sim-Godson, performance analysis helps a lot on improving the

performance of Godson-2 processor. The authors' work is supported by the National Natural Science Foundation for Distinguished Youth Scholar(60325205), the National Natural Science Foundation of China (60673146), the Basic Research Foundation of the Institute of Computing Technology, Chinese Academy of Sciences under Grant No.20056020; Knowledge Innovation Project of the Institute of Computing Technology, Chinese Academy of Sciences under Grant No.20056240; the National High Technology Research and Development Program (863 Program) of China (2002AA110010,2005AA110010,2005AA119020); the National Basic Research Program (973 Program) of China under Grant No.2005CB321600.