分类号 TP3

UDC 编号

中国科学院研究生院
博士学位论文

微处理器性能分析与优化

张福新

指导教师 胡伟武 研究员

中国科学院计算技术研究所

申请学位级别 工学博士 学科专业名称 计算机系统结构

论文提交日期 2005 年 4 月 论文答辩日期 2005 年 6 月

培养单位 中国科学院计算技术研究所

学位授予单位 中国科学院研究生院

答辩委员会主席

声　　明

我声明本论文是我本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，本论文中不包含其他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

作者签名：　　日期：

论文版权使用授权书

本人授权中国科学院计算技术研究所可以保留并向国家有关部门或机构送交本论文的复印件和电子文档，允许本论文被查阅和借阅，可以将本论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编本论文。

（保密论文在解密后适用本授权书。）

作者签名：　　导师签名：　　日期：

摘 要

面对日益复杂的设计和有限的设计时间，如何有效地进行性能分析和优化，是摆在每一个处理器设计者面前的问题。本文紧密结合一个高性能微处理器（龙芯 2 号[HZL05]）的设计，从性能分析环境设计和性能分析方法着手，系统地研究了微处理器性能分析和优化问题，并根据性能分析结果提出了若干龙芯 2 号结构优化方法。

性能分析环境的核心是目标处理器和工作负载的模型，这些模型的准确性、速度和灵活性是其关键指标。处理器的 RTL 模型是一种准确的模型，但是它的速度和灵活性不能满足性能分析的要求。本文提出了一种使用高级语言对硬件建模的方法，并使用它建立了龙芯 2 号的信号级模拟器 ICT-godson。ICT-godson 和 RTL 的逻辑行为相同，但速度可以比 RTL 仿真高一个数量级以上。

ICT-godson 部分解决了运行速度的问题，但由于它还是模拟了所有硬件信号和行为，其速度和灵活性依然受到比较大的限制。为此本文进一步建立了一种更高层的模拟器 Sim-godson。Sim-godson 是一个用 C 语言实现的龙芯 2 号处理器性能模型。执行驱动的组织方式、高效的数据结构和算法以及模块化实现保证了 Sim-godson 的速度和灵活性。Sim-godson 每秒能够模拟约 500K 个处理器周期，和目前公开的最快的详细模拟器相当。Sim-godson 支持大程序快速评估，能够在一个小时内准确地给出 SPEC CPU2000 程序的性能数据。Sim-godson 同时支持用户级模拟和全系统模拟，既能满足不同场合的要求，还能方便地隔离操作系统和 I/O 对程序性能的影响。本文还为 Sim-godson 建立了一个完整的验证流程，它能够有效地控制模拟器的误差。

为了充分满足不同设计阶段和设计目的的需求，性能分析环境应该由一组工具组成。除了 ICT-godson 和 Sim-godson 两种高级语言性能模型，本文建立的性能分析环境还包括 RTL 和 FPGA 仿真环境以及一些辅助的软件工具。RTL 和 FPGA 主要用于验证高级语言模型，各种辅助工具可以用于工作负载的分析和优化。

本文为 ICT-godson 和 Sim-godson 模拟器实现了多角度的数据收集。这种方法能够从各个侧面体现目标处理器的行为，帮助设计人员发现性能瓶颈。为了更深入地理解高性能微处理器的行为，本文进一步考察了一些瓶颈分析方法。这些方法能够用来系统地确定处理器内部各类瓶颈因素的重要程度以及它们之间的关系。

最后，本文详细地讨论了龙芯 2 号性能分析和优化工作。我们展示了多角度的数据收集方法的实际应用，分析了龙芯 2 号的瓶颈，并讨论了基于性能分析的一些优化，包括 Load 猜测优化、存储系统优化等。本文还讨论了对 SPEC CPU2000 工作负载的分析，以及编译器和操作系统对性能的影响。实践证明，本文所建立的性能分析环境和分析方法能很好地满足实际高性能微处理器性能分析和优化的需求。

关键词：微处理器；工作负载分析；性能分析；微体系结构；性能模型；瓶颈分析

Performance analysis and optimizations of microprocessors

Zhang Fuxin （Computer Architecture）

Directed By Hu Weiwu

With the ever-increasing design complexity and limited design time, it is a question for each designer: how to efficiently perform analysis and optimizations of the target system? This paper tries to answer this question via studies of the performance analysis environment and the methodology needed by a realistic high performance microprocessor (the godson-2 processor [HZL05]) design.

At the core of performance analysis environment are the models of target processors and workloads. Accuracy, speed and flexibility are the key requirements of these models. RTL models dictate the logic behavior of microprocessors, but they are too slow, and baring too much details to be flexible enough. This thesis proposes a way to model hardware in high level programming languages and successfully uses it to build a signal-level simulator, which is called ICT-godson. ICT-godson has the same logic behavior with the Godson-2 RTL model, but it can run 10x faster.

ICT-godson partially solves the speed problem, but since it bears all the details it is still limited in flexibility. We go further to build a more high-level model, which is called Sim-godson. Sim-godson is a performance model of the godson-2 processor written in C. The adoption of execution-driven organization, highly efficient data structures and algorithms, and modular design ensures its speed and flexibility. Sim-godson can simulate around 500K cycles per second and this is comparable to the fastest public detailed simulators. Sim-godson supports fast evaluation of large workloads. It can accurately predict the performance of SPEC CPU2000 in an hour. Sim-godson supports both user mode simulation and full-system simulation so that we can choose a conventional mode for different purposes. And this is also useful for isolating the effect of the operating system and extern I/O. We establish a systematical validation flow for Sim-godson, which is effective in controlling the model error.

To best meet the requirements of different design stages and design purposes, a performance analysis environment should be composed of a set of tools. Besides ICT-godson and Sim-godson, the environment established by this thesis includes the RTL model and FPGA emulation platform, and some other software tools. RTL and FPGA are used to validate high-level models while software tools are used for tasks such as workload analysis.

This thesis implements a multi-facet data collection scheme in both ICT-godson and Sim-godson. They can present different views of target processor’s behaviors and help designer to find bottlenecks. To better understanding the behavior of high-performance microprocessors, we further investigate several methods for bottleneck analysis. These methods can systematically decide the effects each bottleneck and their interactions have on the performance.

Finally, we discuss the performance analysis and optimization of the godson-2 processor in detail. We show how multi-facet data collection is used, we analysis the bottlenecks, and we discuss some optimizations based on performance analysis, including load speculation and optimizations of memory system. We also discuss the SPEC CPU2000 workload, and the performance impact of compilers and operating systems. It is shown that the performance analysis environment and methods proposed by this thesis can well meet the real world requirements.

Keywords: microprocessor; workload analysis; performance analysis; micro-architecture; performance model; bottleneck analysis

目 录

图目录

表目录

第一章 引言

由一两个专家根据直觉、经验和一些基础规则来确定一个处理器设计的时代已经一去不复返了。一方面，高性能微处理器设计变得越来越复杂，例如，Intel 的 Itanium2 处理器[McS03]使用了 2.2 亿个晶体管，包括 11 个功能部件，能够同时发射和提交 6 条指令。另一方面，目标工作负载也在不停地变化，从早期以科学计算为主到今天个人桌面应用、服务器事务处理、电子商务应用和各种嵌入式应用并存。为了保持市场竞争力，处理器设计者们需要采用高度系统化的流程来保证其设计能充分利用技术的进步并尽可能地适应目标工作负载的特点。这个流程的核心是目标处理器及其工作负载的模型。设计和验证这些模型的过程称为性能分析[Boc98]。

1.1 处理器性能分析技术

处理器设计可以分为若干阶段，性能分析技术在不同的阶段表现为不同的形式。在设计的早期，焦点是基本设计定义和设计空间探索。设计者根据设计经验、工艺发展情况和工作负载特性确定期望的设计目标。此时需要一些快速的模型和分析方法来辅助决策。下一阶段，人们常常使用详细的性能模型来指导设计。性能分析技术表现为这种模型的设计和验证工作。芯片流片成功后进入硅后（post-silicon）优化阶段。此时我们一方面可以根据实际芯片的表现验证和改进硅前（pre-silicon）性能分析技术，另一方面可以继续进行更多的性能优化工作，如机器相关的编译器优化等。

1.1.1 工作负载表示和分析

一些工作负载的行为特性和所运行的处理器细节无关，例如同一个程序运行时执行的指令序列。提取这种独立于硬件的行为特性能够帮助性能评估。程序执行时的指令序列（也可以包括更多信息，如指令地址，数据地址等）被称为‘踪迹（trace）’。一种性能模拟的方法是用独立的工具得到目标工作负载的踪迹，然后用这些踪迹作为模拟器的输入。这种模拟器只需维持用于控制指令时序的数据，而不需要考虑指令的解释，因此它们可以运行得更快，设计起来也相对容易。这种方法被称为踪迹驱动模拟。

绝大部分踪迹收集工具都使用基于软件指令修改（instrumentation）的方法，即在目标代码中插入额外的指令来生成踪迹。也有一些踪迹收集工具使用硬件实现。例如，Intel 公司就使用硬件来收集用于其微处理器性能分析的踪迹[Int04]。

比较流行的踪迹收集工具包括 Alpha 工作站上的 Atom[SrE94]和 Sun Sparc 机器上的 Shade[Cmk94]。

通过一些专用的踪迹分析程序，人们可以不用模拟整个微体系结构就快速获得目标工作负载的分支预测失效率、CACHE 失效率以及指令相关程度等特征值。这对设计早期的分析和取舍非常有用，也可以用于验证后期的详细性能模型。

标准性能评估公司（Standard Performance Evaluation Corporation，SPEC）开发的 SPEC CPU2000 基准程序集[Hen00]是目前体系结构研究使用最多的工作负载。SPEC CPU2000 是一种计算密集型工作负载，可以用来比较不同的计算机系统。根据实际应用负载变化的情况，SPEC 公司已经推出了 SPEC CPU92, SPEC CPU95, SPEC CPU2000 三代 CPU 基准程序，正在开发 SPEC CPU2005 版本。

有多项研究分析了 SPEC 程序的各种特性。Gee 等人[GHP93]分析了 SPEC CPU92 的存储行为，Charney 等人[Chp97]分析了 SPEC CPU95 的预取和存储行为，Cantin 等人[CaH00]和 Sair 等人[SaC00]分析了 SPEC CPU2000 的存储行为，Kandiraju 等[KaS02]则研究了 SPEC CPU2000 的数据 TLB 行为。

1.1.2 性能模型设计技术

设计一款新的高性能通用处理器时，设计者面临许多选择，例如超标量的组织方式、猜测执行、流水线划分、分支预测器设计以及存储层次设计等等。设计者不仅要考虑每个部分的微体系结构，还需要考虑个部分之间的交互影响，以及结构和工作负载的配合。完成这些任务需要准确、快速、灵活的性能模型。

我们从组织形式、加速技术和误差分析与验证三个方面来总结主要的性能模型[1]设计技术。

1.1.2.1 组织形式

组织形式指性能模型建模和运行的方法，它常常对模型的准确性、速度和灵活性都有很大的影响。建立性能模型最直接的方法就是完全模拟所有硬件单元（寄存器、连线、触发器和存储器等）及其行为，这样性能模型就能真实体现目标处理器行为。我们把这种模型称为信号级模型，它的优点是准确，但是其速度和灵活性通常比较有限。例如，龙芯 2 号处理器的 RTL 模型是一种信号级模型，它每秒大约只能模拟几千个处理器周期，而且修改比较困难，也不容易和其它分析工具接口。一般说来，信号级模拟器难以全面满足性能分析的需求。

上节我们提到了踪迹驱动（trace-driven）的模拟器。这是一种流行的性能模型组织形式。在这种组织方式中，指令解释与模型实现互相独立。指令解释和执行由独立的踪迹生成器完成，而模型实现中只需要考虑指令时序控制。这样能够降低模型实现难度，也能提高运行速度。工作负载的踪迹可以通过各种踪迹收集

[1] 一个能够执行的性能模型称为模拟器。本文有时不加区别地使用这两个名词。

工具快速获得。收集到的工作负载踪迹存在磁盘上，可以重复使用。踪迹驱动方式的主要问题是：(1)、由于微体系结构不再直接影响程序的正确运行，建模时引入的错误可能很难被发现，需要辅以系统的验证手段；(2)、大工作负载的踪迹需要大量的磁盘空间。

执行驱动（execution-driven）可以看作一种特殊的踪迹驱动。把踪迹驱动模型中的踪迹生成器和踪迹分析器链接在一个程序里，使这个程序运行时一边产生目标负载的踪迹一边分析所生成的踪迹，就成了执行驱动。实践上采用执行驱动时，生成踪迹的方法可能不再是基于软件指令修改，而是采用快速的功能级指令模拟器。采用指令模拟器生成踪迹的优点是它可以跨平台执行（除非它采用直接执行技术或者平台相关的二进制翻译），而基于软件指令修改的方法只能在目标软件能运行的机器上使用。功能级的指令模拟通常远远快于详细的性能模拟，因此它所花的时间不会对整体模拟速度有大的影响。近年来执行驱动的模拟器也很常见，甚至有超过踪迹驱动的趋势。

踪迹驱动和执行驱动都逐拍地模拟处理器内部状态，速度和灵活性都受到一定限制。此外，它们只建立处理器模型，没有建立工作负载的模型。另一种思路是建立某种分析性的模型，其中比较成功的尝试是一些基于统计的模型。它们先通过某种方法得到程序执行的一些统计信息，然后利用这些统计信息建立统计模型，再通过这个统计模型来进行更多的分析。这种模型综合了处理器和工作负载的信息，建立之后能够很快地评估新的处理器结构或负载。而且，这种模拟器可以用来做一些 cycle-by-cycle 模拟器中难以实现的事情，例如构造一些特殊的工作负载以研究处理器行为随工作负载特性变化情况。

与统计模型相关的研究可以分为两大类。第一类[Jou89, Nos94, NOS97]把性能看作处理器并行性和程序并行性的交互结果，通过分别建立处理器模型和程序模型，并把程序模型作为处理器模型的输入来进行性能分析。程序的模型参数（指令相关概率等）通过详细模拟运行一遍该程序而得到。这种方法的处理器模型是一种概率模型，它根据输入和概率分布在不同的处理器状态间转换，运行时间只和状态数目相关。但是，这种处理器模型的建立比较困难，目前只在一些简单的例子上取得成功。第二类[Cas98, OcF00, NuS01, EJS04, JEJ04]通过实际程序详细模拟得到统计信息（指令组合、指令频率、指令间相关信息、cache 命中率等），并利用这些统计信息合成符号程序，然后通过处理器模型符号执行这种合成程序得到结果。这种符号程序实际上是目标工作负载的一个统计模型，它包含的指令少，信息精炼。对这种程序的模拟可以很快收敛到一个结果。同时，这种方法也能够达到比较高的准确性。例如，Eeckhout 等人[JEJ04]改进后的 HLS 模型[OcF00]，对 SPEC CPU95 程序的平均 IPC 预测误差只有 4.1%。

1.1.2.2 加速技术

这里所说的加速技术是针对踪迹驱动或者执行驱动的模拟器。踪迹驱动或者执行驱动的模拟器逐拍地模拟目标程序的运行。通常目标机器的一个处理器周期需要成千上万的主机周期来模拟，因此模拟速度和实际主机运行速度相比相差很远。而且随着技术进步，处理器变得越来越复杂，典型工作负载也不停增大，模拟速度的问题日益突出。因此，模拟器加速技术始终是一个研究热点。

常见的模拟器加速技术包括以下几种。

- 采样（sampling）

统计采样在许多领域都有广泛的应用，例如人口普查、产品质量检查等。使用采样技术，人们可以随机地或者周期性地采样和测量性能，然后以样本的平均性能来作为整个工作负载性能的估计。假设一定的分布概率后，可以用采样原理来估计采样的误差。踪迹驱动的模拟器可以直接选择样本片断来进行详细模拟。执行驱动的模拟器常常使用功能模拟和详细模拟结合的方法来实现采样，用前者快速执行未选中的样本，用后者执行和测量选中的样本。

已经有多个研究[KHW91, MGA93, CHM96]表明采样方法能够带来显著的加速，而不损失过多的精度，但这些研究没有给出易于操作的方法来控制精度损失。事实上常常加速越多精度损失越大，影响了研究者对采样方法的信心。最近，Wunderlich 等人[WWF03]提出一种严格的统计采样方法（SMARTS），它利用统计原理和实际程序运行表现出来的统计特性实现了对误差的估计和控制。对给定的误差范围和置信度，如果采样结果不能满足要求，它能自动推荐很可能满足要求的采样频率。

减小采样的误差需要解决微体系结构状态的预热（warm up）问题，即保证在开始测量一个样本时，模拟器的微体系结构状态处于正确或者基本正确的状态。一种预热方法是在开始测量前先详细模拟一段指令，如何决定这段指令的长短和预热误差是关键问题[CHM96, HaS03]。采用执行驱动的模拟器可以在快速模拟中保持对关键微体系结构状态的更新，以减少预热时间和预热误差。

- 选择有代表性的工作负载片断

这种方法的一个例子是 Simpoint[SPH02]。Simpoint 先完整地运行程序（用踪迹生成工具或快速模拟器），得到每个样本中所有基本块执行次数所构成的基本块矢量（BBV, basic block vector[SPC01]），并根据 BBV 的相似性通过聚类算法把所有样本分成若干类，然后为每类挑选一个样本来做为代表，最后得到一组作为代表的样本。整体的性能由所有代表样本的性能及其权重决定。从本质上说，这种方法也可以看作一种采样方法。

- 更快的算法

一个典型的例子是利用所谓“单遍多配置”算法[Mat70]，运行一次模拟器

就可以获得多个 CACHE 配置（不同大小，不同相联度）的失效率数据。

- 预处理

对踪迹或者目标可执行程序进行预处理，常常可以减少模拟器运行时的开销。

- 软件缓存

把模拟器重复做的事情缓存起来，加速其运行。

- 并行

最后，可以通过同时模拟多个踪迹或者同一个踪迹的多个样本来加速模拟[NMN97, HaS01, GMC03]。和采样技术类似，这种技术的关键是要解决预热和误差控制问题。

1.1.2.3 误差分析和验证

模拟器的误差分析和验证是一个非常重要的工作，缺少这一步工作，使用模拟器得到的结果常常是不准确甚至是错误的。同时，这也是一件困难的工作。多位研究者的工作都验证了这个论点。

Black 等人[BlS98]给出了一些性能模拟器的验证经验。他们通过比较性能模型和实际机器的运行周期数，对性能模型进行了反复的改进。他们发现，即使经过长时间的分析和调试后，建模和抽象处理的误差仍然存在;而且某些误差只能通过和实际硬件比较来发现。他们的工作表明，在采信一个性能模型的结果之前，需要对它进行深入的、反复的验证。

Desikan 等人[DeB01, DBK01]讨论了 Sim-alpha 模拟器和 Alpha 21264 处理器的验证工作。他们发现，不针对特定处理器的模拟器，如 Simplescalar，倾向于高估性能。这通常是因为它没有反映实际硬件中的一些限制条件。有意思的是，针对特定结构设计的模拟器在未经验证之前却常常低估性能。这大概是因为目标处理器的设计都经过非常精细的调整，不容易正确实现。

Gibson 等人[GKO00]比较了 FLASH 模拟器和实际处理器的执行时间。他们发现对有些程序两者的差别高达 30%以上，不过如果模拟器的主要部件模型正确的话，模型器通常还是能够准确预测体系结构级的变化趋势。

Cain[CLS02]等人测量了操作系统和 I/O 对模拟器精确度的影响。他们的结果表明，忽略操作系统的影响可能导致高达 100%的误差。

1.1.3 硅后优化技术

芯片流片成功之后，性能分析工作进入另一阶段。一方面，我们可以比较硅前（pre-silicon）预测性能和实际性能，根据实际情况改进硅前性能预测流程。另一方面，性能分析工作的主要任务转向系统精调，例如机器相关的编译器优化、操作系统优化等。

实际系统中无法象模拟器一样任意观察处理器的内部状态。为了有效地进行优化，现代商用处理器几乎都实现了硬件性能计数器，例如 Pentium 4[Int03-1]，MIPS R10000[Yea96]，Alpha 21264[Kes99]等。硬件性能计数器是一组处理器内部寄存器，用于统计某些事件的发生次数，如 CACHE 不命中次数，分支预测失败次数，功能部件使用次数等。它们可以提供某个工作负载全速运行时的一些特性。配合一些采样技术，可以利用硬件性能计数器做程序优化，如 DCPI（Digital Continuous Profiling Infrastructure）[ABD97]。

传统的硬件性能计数器对单发射、按序执行的处理器很有效，因为这种处理器中每个事件很容易对应到执行时间。对于复杂的超标量乱序执行的处理器，解释硬件性能计数器的值常常很困难：流水线同时有很多并发的指令，每个事件不容易准确归结给特定的指令；而且孤立地考虑每类事件无法体现事件之间的相互影响，不能提供足够的信息。ProfileMe[DHW97]通过采样某条指令的详细数据（流水延迟，所遇事件等）来解决事件准确定位问题；通过同时采样一对临近指令的详细数据，它能够获得一些和并行度相关的信息。Fields 等人提出的 Shotgun profiling[FBH04]也采样指令相关数据，并用这些数据建立依赖图模型[FBR01]，再通过依赖图分析获得事件交互代价和执行时间切分图。

许多高性能处理器厂家都提供了功能强大的硅后性能分析工具，例如 SGI 的 Speedshop[SGI]和 Intel 的 Vtune[IntV]等。它们可以利用按时间采样或者按事件采样的技术，把性能瓶颈归结到应用程序的特定位置，以帮助用户找到和优化引起问题的代码。

1.2 典型性能分析环境介绍

本节介绍一些工业界和学术界的典型性能分析环境。

1.2.1 Alpha 微处理器性能模拟

1998 年前后，DEC 公司的设计队伍开发了一个处理器性能模型来指导处理器的设计。他们的设计目标是：

（1） 模型必须相当灵活，模型必须能探索各种设计空间，各种参数能灵活配置，代码要便于修改。

（2） 模型必须要快，在中等配置的 Alpha 工作站上能达到大约每秒 10,000~50,000 条指令的速度（这样一般的测试程序运行时间为 3 个小时左右）。

（3） 模型必须反映真实的硬件实现情况。例如 CACHE 大小有限、访问存储需要花时间、寄存器端口有限并且代价很大等等。在项目的后期，性能模型要尽可能地和 RTL 模型的行为一致。

设计的结果是一个基于 Atom[SrE94]/Aint[Pai94]的踪迹驱动性能模型。目标程序的踪迹使用 Atom 或者指令集解释器 Aint 生成。Atom 通过修改目标程序插入代码来生成踪迹，而 Aint 通过解释执行目标程序来生成踪迹。

Atom 只能送实际执行的指令给模型，而现代处理器中存在大量推测执行的指令。误预测路径上的指令执行会引起各种问题：这些指令会占用一些资源，可能会延迟其它指令的执行，而且误预测路径上的 Load 指令会引起对指令和数据 CACHE 的污染。如果不考虑误预测路径上的指令执行，模拟的精确性会受到影响。

Aint 模式可以解决这个问题。Aint 模式下，性能模型中的分支预测器能指导 Aint 产生误预测路径上的指令流。这样性能模型就能得到误预测路径上的指令所产生的处理器状态。当误预测分支被修正时，Aint 能消除误预测路径上的执行所带来的影响。

这个模型能够输出的数据包括：程序执行的拍数、每拍平均执行了多少条指令以及功能部件的使用、每种指令类型花在每个主要流水级的拍数、频繁执行的指令序列、分支预测和其它预测的报告、很难预测的指令等等。

设计者在加快模拟速度方面也做了许多工作，包括采用踪迹驱动模型、采用事件驱动方法以及支持快速掠过模式和随机采样模式。

设计者讨论了该模型设计的一些经验。他们认为，灵活性是模型开发的首要目标。早期开发一个灵活的框架，随着项目不断推进，会获得很多的回报。模型的灵活性，不但很重要而且能做到。其它的一些结论包括：在测试分支预测器的想法时，应该写一个专用的模拟器来单独追踪分支指令；事件驱动的模拟不是很有效，大约只有 35%的模拟时间是事件驱动；图形化的输出在理解程序的行为时很有效，但是在模型调试的时候不是很有用；基于文本的事件日志很有用等等。

1.2.2 PowerPC 微体系结构探索的环境

IBM 的微处理器研发部门开发了 MET（Microarchitecture Environment Toolsets）[MoW99]用于 PowerPC 处理器微体系结构的探索。MET 是一个基于 PowerPC/AIX 的环境，它由一组工具组成，包括处理器性能模型 Turandot[MoB99]，踪迹生成器 Aria，踪迹读入器以及一些验证工具。

MET 采用踪迹驱动建模的方法，使用单独的踪迹生成器，把踪迹产生过程和处理器模型分离。MET 有两个工作模式，一种是踪迹文件已经存在了，使用踪迹读入器把踪迹读入送给 Turandot；另外一种是使用 Aria 和 Turandot 配合的方式，由 Turandot 控制 Aria 执行，由 Aria 负责产生踪迹并传给 Turandot。

Aria 能在控制程序（Turandot）的指导下跟踪用户程序的执行。Aria 和 Turandot 以及被跟踪的用户程序都处在同一个地址空间内，这样工具和程序之间的通讯可以使用简单的函数调用，避免进程间通讯的开销。Aria 以基本块为单位

动态修改程序产生踪迹。它能够缓存已经处理过的基本块信息，消除重复转换。Aria 也能产生误预测路径上的指令序列。

MET 非常重视提高模拟速度。Arial 翻译每条目标指令代价大约只有 40 个机器指令，还采用缓存技术避免重复的工作。Turandot 模拟器设计中也采用了多种技术来加速模拟，包括：逆向运行流水线避免流水寄存器的读写控制；合并中间不停滞的几个连续流水级的工作；采用单队列最小化流水级间信息交换开销；支持采样技术等等。它甚至在程序风格上也充分考虑了优化：大量采用预处理技术来避免运行时的译码工作；使用编译时参数化避免运行时的参数选择；编码充分考虑编译优化。它在 200MHZ 的 RS/6000 工作站上能够达到约 100k 指令/秒的模拟速度。

MET 也突出了其验证工作。它使用了性能测试向量自动生成[Bos98]、循环性能边界分析[BKO97]等工具来进行系统化的验证。

1.2.3 Simplescalar

Simplescalar 工具集[BuA97, ALE02]是一个用于构建各种模拟程序的系统软件框架。Simplescalar 工具集提供了一些常见微体系结构模块的实现（如 CACHE、分支预测器等）以及统计分析、调试、验证和可视化等辅助功能。Simplescalar 还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，以及一个详细的超标量多发射处理器微体系结构模拟器。Simplescalar 工具集广泛用于计算机体系结构的研究和教学。2000 年，顶级体系结构会议上超过三分之一的文章使用 Simplescalar 来评价他们的设计。

Simplescalar 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。

Simplescalar 只提供单线程、静态链接、用户程序的模拟。用户程序对操作系统的调用是通过一个代理接口调用主机操作系统来完成的。这使得它的应用范围受到一定的限制。此外，Simplescalar 本身所提供的超标量处理器模拟并不针对任何特定结构，这使得模拟器本身比较简明和通用；但由于和实际硬件有较大的区别，使用它得到的一些结论不能简单地适用到别的结构。

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[BTM00]，多线程模拟器 SIMCA[Simca]，支持统计采样的 SMARTS 系统[WWF03]，用于选择模拟样本的 Simpoint[SPH02]以及一个经 Alpha EV6 工作站验证过的模拟器 Sim-alpha[DeB01]等。

1.2.4 SIMOS

SIMOS[RoH95]是 Stanford 大学开发的一个全系统模拟器，它不仅仅模拟处

理器，还模拟了一个实际计算机系统的各种外设，包括存储子系统、DMA 和中断控制器、终端、网络接口、磁盘、Frame Buffer、鼠标键盘等。

SIMOS 具有如下特点：

- 全系统模拟，可用于评估操作系统、网络、I/O 和多处理器。
- 可切换处理器模拟模块，支持多种处理器模拟器，包括简单的指令集模拟器 MIPSY，超标量处理器的模拟器 MXS，以及一个使用二进制翻译的模拟器 Embra[WiR96]。
- 灵活的系统控制功能，采用 TCL 脚本完成模拟器控制，性能数据收集等功能，可以根据需要随时扩充。
- 调试功能强大，可以用 gdb 调试器对被模拟程序进行源代码级调试。

SIMOS 的主要缺点是它运行的是一个经过修改的商业操作系统 IRIX，而现在该版本的操作系统已经不常见了。而且，一般用户无法获得该操作系统源代码，不能进行涉及操作系统的改进（比如增加新设备）。此外，Embra 模拟器也不能在非 MIPS 机器上使用。

1.2.5 Intel Pentium 4 性能分析和验证

Singhal 等人[Sin04]披露了 Intel Pentium 4 的性能分析和验证过程。Intel 也使用踪迹来作为模拟器输入，但是，Intel 使用的踪迹和传统意义的踪迹有所不同：它是由处理器体系结构状态、系统内存和一些外部事件（如中断，DMA）三元组构成的，被称为 LIT（Long Instrution Traces）。大部分 LIT 是由硬件从实际程序运行中收集而来的，也有一些使用软件得到。每个程序用约 25 个踪迹表示，每个包含约 30M 条指令。踪迹包含每个系统调用的信息，模拟时不实际执行遇到的系统调用（因为系统调用的效果不是确定的），而是用踪迹中保存的相关信息重现系统调用的效果，这样可以保证模拟是完全可重现。实际上，LIT 是一种检查点（checkpoint）。

和 Alpha、PowerPC 环境不同的是，Pentium4 的性能模型同时也负责驱动实际的指令流。因为设计者认为强制性能模型产生正确数据有利于提高对实现正确性的信心。他们采用一个大型的分布式环境来进行性能模型和 RTL 模型的交叉验证。发现的任何异常都被提交到一个数据库中，由一个专门小组负责解决。

Pentium 4 使用的硅后性能分析工具包括 EMON 和 VTune。EMON 被用来从硬件性能计数器获得数据，进而构成目标工作负载的高层特性集合。Vtune 则是一个面向用户的性能分析工具，它可以用来分析应用程序的性能问题。

1.2.6 其它性能分析环境/工具

SimICS[MCE02]是一个著名的商业模拟环境。SimICS 也是一个全系统模拟器，它支持多种体系结构，可以运行不经过任何修改的操作系统。它的运行速度

比较快。它支持 MAI（MicroArchitecture Interface），用户可以用它来实现踪迹驱动的性能模拟器，免去功能模拟部分的实现工作。Wisconsin 的 GEMS 模拟器[MSB05]是一个基于 SimICS 设计的多处理器模拟器。

RSIM[HPR02]/ML-RSIM/URSIM 是一组执行驱动的多处理器模拟器，它们模拟了超标量处理器、存储层次以及多处理器互连和协议。Rice 大学的 Hughes 等人开发了 RSIM；Utah 大学的张立新等人在 RSIM 的基础上做了一些增强，增加了内存控制器和 DRAM 模拟，并最终把指令集从 Sparc 转到 MIPS，他们的版本称为 URSIM；ML-RSIM 在 URSIM 2000 年版本基础上做了更多增强，增加了一个操作系统以及 PCI、SCSI 等 I/O 系统。

1.3 研究背景

龙芯 2 号处理器[HZL05]是中科院计算所研发的高性能通用处理器。从 2002 年 10 月开始设计，历经七次流片，于 2005 年 1 月 31 号通过中国科学院组织的鉴定。鉴定委员会认为：龙芯 2 号是一种类 MIPS 指令系统的高性能 64 位通用 CPU 芯片。该芯片采用四发射的动态超标量超流水线结构、寄存器换名、动态调度、动态分支预测、非阻塞 cache 访问、取数推测（Load Speculation）、分离读取（Split Read）等先进体系结构技术。龙芯 2 号在 1.8 伏正常供电电压下，最高主频达到 500MHz，实测得到的最高浮点运算速度单精度为每秒 19.9 亿次，双精度为每秒 9.93 亿次；龙芯 2 号高性能通用 CPU 芯片的总体性能已达到 2000 年左右的国际先进水平，居国内通用 CPU 研制领先水平。

龙芯 2 号的性能分析技术伴随着龙芯 2 号一起成长。设计早期，我们解决了用高级语言对硬件建模的问题，使用 C 语言编写的信号级模拟器来做性能分析。由于早期模拟器的灵活性和速度都有限，在很长时间内性能分析工作都跟不上处理器设计的进度。性能分析工作主要是评估不同的微体系结构方案的相对性能。

我们一直努力改善性能分析环境。一方面，我们对信号级模拟器进行了大幅优化，使得它的运行速度（在 AMD64 3200+上）达到了每秒 50-100K 处理器周期，加快了评估进程。另一方面，在借鉴现有研究结果的基础上，我们为模拟器增加了许多功能，包括各种数据收集和调试功能，提高了性能分析的效率。但这些工作没有根本解决速度和灵活性的问题。

2003 年 6 月龙芯 2B（第一个成功运行的龙芯 2 号样片）流片成功之后，我们利用实际芯片对性能分析工作做了一次全面的验证，发现了许多不足之处。针对这种情况，我们在如下方面改进性能分析工作：

- 提高性能分析的速度和灵活性。由于模拟速度和方法的局限，早期的性能分析只能运行 SPEC CPU2000 test 输入集的一部分，无法预测实际测试所使用的 reference 输入集的性能。原来的信号级模拟器也难以满足灵

活修改的要求。

- 考虑系统集成性能。包括处理器、配套芯片组、操作系统、编译器等。由于缺乏经验，早期的性能分析主要关心处理器核心，对其它部分的建模出现了比较大的偏差，导致实际系统的 IO 性能、访存性能等都不理想，最终集成性能也低于预期。
- 需要一套行之有效的性能分析方法和流程来定位性能瓶颈。要从比较两个给定配置的性能发展到能够分析特定配置和发现改进方案。

本文的工作很好地解决了这些问题，为提高龙芯 2 号的性能作出了显著的贡献。

1.4 本文的贡献

本文的研究旨在为高性能微处理器设计寻求一套有效的性能分析方案。主要的贡献和创新之处包括：

1. 结合龙芯 2 号的开发，建立了一套完整的、实用的高性能微处理器性能分析环境，包括行为级 C 模拟器、信号级 C 模拟器、RTL 仿真、FPGA 验证等分析平台和相应方法，以及不同性能分析平台之间的相互联系。
2. 针对 RTL 模型以及 SystemC 等高级言语模型的效率问题，提出了一种用 C 语言进行信号级建模的方法。该方法不仅能对处理器中的硬件单元进行建模，而且高效地解决了用串行语言对硬件并发行为的建模问题。
3. 实现了一个高效的龙芯 2 号处理器性能模型，并为其设计了一个系统化的验证流程。该模型结合了 Simplescalar 和 SIMOS 的优点，同时支持用户级模拟和全系统模拟。
4. 解决了大程序快速评估问题。利用本文改进后的大程序快速评估技术，我们能够在一个小时内准确地给出 SPEC CPU2000 的性能数据。
5. 利用上述性能分析平台和方法对龙芯 2 号处理器的结构进行了详细的性能分析，并根据性能分析结果提出了若干龙芯 2 号处理器的结构优化措施，包括 Load 猜测技术、访存瓶颈分析和优化方案等。同时，分析了操作系统和编译器对系统性能的影响，根据分析结果提出并实现了一些优化方案。

上述性能分析平台和方法在龙芯 2 号处理器的研发中得到了很好的应用，基于性能分析的结构优化工作把龙芯 2 号的性能提高了 30%以上。

1.5 论文的组织

本文紧密结合高性能微处理器龙芯 2 号设计过程，探索了微处理器性能分析和优化技术。本章首先给出了性能分析工作的定义，然后总结了性能分析领域相

关工作，包括关键技术、典型分析环境/工具，接着简要介绍了课题的背景和研究内容，最后指出本文的贡献和组织方式。

第二章先简要介绍龙芯 2 号微处理器性能分析环境的各个组成部分。接下来的三、四、五章分别详细讨论其中信号级模拟 ICT-godson 的设计和实现、Sim-godson 处理器性能模型设计和验证、RTL 和 FPGA 仿真环境。

第六章讨论性能分析方法。其中，我们分析和总结参数敏感性分析、PB 设计、交互代价分析等三种性能分析方法。

第七章比较全面地介绍龙芯 2 号处理器性能分析工作，展示龙芯 2 号性能分析环境和瓶颈分析方法的实际使用，同时介绍一些基于性能分析的处理器优化。

第八章总结全文，并指出未来工作。

第二章 龙芯 2 号性能分析环境

为了满足不同设计阶段和设计目的的性能分析需求，人们常常使用一组工具来进行实际处理器的性能分析工作。这些工具通常包括一个或者多个处理器性能模拟器和一些辅助工具。我们把这些工具统称为性能分析环境。各大处理器厂商都有自己的性能分析环境，如引言所介绍的 Alpha 性能分析环境，PowerPC 的 MET 以及 Intel Pentium 4 的性能分析和验证环境等。

龙芯 2 号的性能分析环境包括如下工具：信号级模拟器 ICT-godson、执行驱动模拟器 Sim-godson、RTL 仿真环境、FPGA 仿真环境和一些硅后优化软件。本章概要地介绍这些工具，包括它们的主要特点、使用场合等。后续章节将讨论具体的设计和实现。

2.1 ICT-godson 模拟器

该模拟器是一个详细的信号级模拟器，用 C 语言实现。它的主要特点包括：

- 详细的硬件模拟。模拟了龙芯 2 号处理器内部的所有硬件包括存储器、寄存器、触发器和连线，和 RTL 的逻辑行为几乎完全一致。
- 全系统模拟。模拟了必要的外设（如串口、磁盘、内存），能够运行操作系统。
- 比较完善的调试功能。通过命令行调试接口可以逐拍运行模拟器，设置断点，选择记录某些事件到事件日志，以及检查处理器内部状态等等。它还支持检查点（checkpoint）保存和恢复，极大地方便了调试。
- 多角度的数据收集功能。能够生成基于事件的统计、基于流水级的统计、基于指令的统计、基于数据分布的统计，从各种角度反映处理器的行为。
- RTL 测试向量生成。能够将模块级接口上的输入输出数据转化为 RTL 测试向量。
- 作业管理支持。能够在分布式机群上利用作业管理器批量运行。而且，我们开发了一批作业管理和数据分析脚本，能够有效地收集和整理数据。

在龙芯 2 号设计过程中，ICT-godson 是处理器的第一个模型，我们把它作为一个可执行的硬件设计文档，用于实现 RTL 模型。RTL 模型和 ICT-godson 的交叉验证对 RTL 模型的调试起了很大的帮助作用。

ICT-godson 模拟器也是早期龙芯 2 号的主要性能分析平台。经过精心优化后，该模拟器在 AMD64 3200+的微机上达到每秒约 50-100K 处理器周期的模拟

速度，能满足一些场合的性能分析要求。ICT-godson 具有高度的准确性，但是由于它模拟的细节过多，进一步提高它的速度以及对实现做比较大的修改都比较困难。

2.2 Sim-godson 模拟器

ICT-godson 模拟器满足了准确性要求，但它的速度和灵活性都受到比较大的限制。为了在庞大的设计空间中有效地寻找优化设计，我们需要更快速、更灵活的性能模拟器，为此我们设计了 Sim-godson 模拟器。Sim-godson 是基于 Simplescalar 工具集开发的执行驱动的性能模拟器。和 ICT-godson 相比，它有一些显著的优点：

- 运行速度快。我们采用了一系列加速技术来保证模拟器的速度。目前，在 3.0GHz 的 Pentium4 微机上，它每秒能模拟约 500K 个处理器周期。
- 灵活性高。一方面，我们采用功能模拟和时序模拟分离的执行驱动方式，降低了实现困难。另一方面，我们充分利用 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。
- 支持大程序评估。Sim-godson 实现了多种加速大程序评估（如 SPEC CPU2000 ref 输入数据集）的方法，包括 Simpoint[SPH02]，SMARTS[WWF03]。Sim-godson 能够在一个小时内准确地给出 SPEC CPU2000 的性能预测。
- 支持 WATTCH 功耗模拟器。

Sim-godson 并没有完全模拟龙芯 2 号的所有细节。为了保证模型抽象的误差在控制范围内，我们采取了一个系统化的验证流程来保证 Sim-godson 模拟器的准确性。

原始的 Simplescalar 工具集只支持用户态单进程代码的模拟。为了全面考察系统性能，满足将来多处理器（包括 CMP 和 SMT）性能分析的要求，我们为 Sim-godson 实现了全系统模拟。Sim-godson 的全系统模拟建立在 SIMOS 的基础上，支持多处理模拟。Sim-godson 同时支持用户级模拟和全系统模拟，为评估操作系统和外设对性能的影响提供了极大的便利。

Sim-godson 模拟器是龙芯 2 号性能分析环境中主要的设计空间探索工具。同时，它可以用来进行操作系统性能评估、芯片性能预测等工作。

2.3 RTL 仿真环境

我们为龙芯 2 号的 RTL 模型设计了一个仿真环境，它能直接运行一些简单的小程序，也能运行操作系统和真实应用程序。RTL 仿真环境除了功能验证外，

也用于和 ICT-godson 以及 Sim-godson 模拟器的相互验证。

2.4 FPGA 仿真环境

实际进行流片的处理器设计一般都会经过 FPGA 仿真验证。FPGA 仿真环境的速度快，I/O 系统接近实际目标系统，因此对功能验证具有不可替代的作用。同时我们发现，经过一些改进之后 FPGA 仿真环境也可以用于性能分析。

作为性能分析工具，FPGA 仿真环境可以直接用来评估不同实现的性能，也可以用来研究片上性能分析硬件的实现（如硬件性能计数器，shotgun profiling[FBH04]等）。而且，FPGA 的速度和真实性使得它可以成为 Sim-godson 全系统模拟器的一个理想验证工具。

2.5 硅后优化工具

硅后优化工具对于深入理解芯片的实际表现有重要的作用。理解现有芯片和工作负载的表现能够为下一代处理器设计积累宝贵经验。

龙芯 2 号实现了硬件性能计数器（performance counter），我们为其移植了一个工具软件 PERFCTR[Pet02]。利用 PERFCTR，我们能够收集硬件性能计数器所记录的各种程序运行特征。

龙芯 2 号的硅后优化工具还包括一个链接级优化器和一个踪迹生成工具，它们都是基于 ALTO[RDW01]开发的。ALTO 是 Muth 等人在 Alpha 平台开发的一个链接级优化器，它能够对已经链接好的可执行程序进行再优化。通过充分利用此时所拥有的整个程序的信息，它能够取得良好的效果。ALTO 能够修改二进制代码，因此可以比较容易地做成一个踪迹生成器。我们将 ALTO 移植到 MIPS 平台，并做了一些优化[刘张王 04]。

第三章 ICT-godson 模拟器的设计与优化

ICT-godson 模拟器是一个信号级模拟器。它模拟了龙芯 2 号处理器内部各种硬件资源的逻辑功能和连接关系，整个模拟器的逻辑行为和 RTL 模型零延迟仿真逐拍一致。虽然 ICT-Godson 是完全用 C 语言实现的，但它的每个模块都可以很容易地转换成硬件描述语言，模块间接口也和 RTL 模型完全一致。这种模拟器准确性很高，事实上在龙芯 2 号设计中，它是作为一个可执行的设计文档，用于实现 RTL 模型。

用 C 这样的串行语言去描述硬件，首先需要解决建模的问题。本章先讨论 ICT-godson 的建模方法，然后讨论性能分析相关功能的设计以及速度优化工作。

3.1 硬件建模

信号级模拟器要对硬件资源包括寄存器、触发器、连线等进行建模，还要处理 RTL 级的一些语义。最自然的方法是采用硬件描述语言，如 VHDL，Verilog 等。但是，一般结构设计人员和系统软件设计人员使用 C/C++编程，用硬件描述语言建立的模型和其它部分的工作配合起来会有很多困难。此外，硬件描述语言的仿真一般使用商用软件，它需要满足所有硬件描述语言特性的需求，不能对特定代码做针对性的优化。因此，对性能分析来说，使用 C/C++实现模拟器有比较大的优势。

3.1.1 C/C++对硬件建模的问题

使用 C/C++对硬件建模需要克服一些建模的困难，包括：

- 对硬件并发行为建模。C/C++是串行的语言，而硬件具有大量并发行为。例如，一个信号变化可能引起几个模块同时产生动作。
- 硬件中的赋值并不是立刻生效，而是有一定延迟。例如，寄存器、存储器是在时钟沿采样的，这一拍产生的值要下一拍才能看见。硬件描述语言里可以任意地使用一个寄存器的值而不用担心提早看到本拍对它的赋值，语句之间没有次序问题。由于这个延迟的存在，硬件描述语言里可以直接交换两个寄存器的值，而 C/C++中不引入中间变量无法交换两个变量的值。
- 数据类型表示。例如，硬件描述语言中可以表示任意长的线，可以表示不确定值 X，高阻 Z 等。

3.1.2 SystemC 建模言语

SystemC[SysC]是一种系统级的建模语言，它完全用 C++实现，能够描述 RTL 级行为和系统级行为。SystemC 采用进程来对硬件并发行为建模。每个进程是一个独立的控制流，在某些事件或者信号值的改变时被唤醒进行一些处理，完成处理后挂起等待下一次唤醒。对于赋值延迟问题，SystemC 引入 delta cycle 的概念来解决。一个 delta cycle 是用户不可见的很小的时间单位，每个赋值在一个 delta cycle 之后才能被其它进程观察到。数据类型则利用 C++的类和封装功能实现。

SystemC 是应 SoC（System-on-chip）设计的需求而诞生的。它主要的优点是可以用一个统一的语言描述不同抽象层次的功能、通信、软件和硬件，同时保持一定的效率。

3.1.3 ICT-godson 的建模方法

SystemC 虽然能够解决用高级语言对硬件建模的问题，但是它使用的方法具有很大的开销。一个复杂系统的模型需要用到大量的进程（或者线程），它们之间的通信、同步等开销会相当可观。SystemC 对数据类型的封装虽然可以方便熟悉硬件描述言语的设计者，但是这些封装也带来效率的问题，常常一个简单的相加都需要调用一个函数来完成。因此它在性能上并不能比 RTL 模型高很多。

ICT-godson 使用另一种建模方法，用 C 语言实现了信号级模拟器。这种建模方法避免了多进程和数据类型封装的开销，具有较高的效率。

首先，对于赋值延迟的问题，我们使用一对变量来表示所有的寄存器和存储器。例如寄存器 R 用 r 和 r_bak 表示。每一拍中，寄存器输出端的值用 r 表示，输入端的值用 r_bak 表示。所有寄存器赋值对输入端操作，一拍结束后统一“采样”到输出端（即把输入端的值拷贝到输出端）。这样，每个模块可以自由地对寄存器赋值，新值会被延迟到下一拍才会被其它部分看到。

对于数据类型问题，ICT-godson 用 C 的标量类型定义了各种 64 位以下的线和寄存器。大于 64 位的类型以及模块间的总线使用结构来表示。由于 ICT-godson 代码中涉及的数据类型较少，也不使用大于 64 位的运算，这种简单处理就已经足以表示了。这样，硬件中的各种运算大部分直接利用 C 的运算符完成，没有任何额外开销。ICT-godson 中无法表示信号的 X 和 Z 值，但这些值主要用于测试复位逻辑，对性能分析没有实质影响。

硬件的并发行为表现在信号级模拟器的 C 语言实现中是调用次序的问题。在 C 语言中函数和语句只能以一个固定的次序执行，这样其中一些逻辑上同时发生的事件会被指定某种次序，进而导致问题。例如，我们按先 A 后 B 的次序调用模块，那么只要 A 用到 B 的输出就会有问题。分解和重组模块把并行的事件放在一起能够解决问题，但这样不能满足信号级模拟的要求，也容易引起错误。

这个问题可以看作一个信号传播的问题。解决并发的问题即如何保证本拍信号的稳定值能够传播到每个模块。在上述例子中，就是 B 的输出如何传播到 A 中。

我们使用一种重复调用的策略来解决信号传播的问题。重复调用要求每个模块能在一个周期内执行多次而不影响其有效输出（即在输入完全有效时的输出，包括输出总线和寄存器）。而采用上述寄存器表示技术之后，我们很容易保证这一点。因为，每个模块的输出只依赖寄存器的（输出端）值和输入总线的值，寄存器值在一拍内不会被修改，因此只要输入相同，每次调用的输出一定是相同的。无效的输入可能导致错误的 r_bak，可以在每次重新调用前将其恢复（为了防止 r_bak 被引用）为输出端的值。

下列的代码展示了一个解决信号传播的方法：

```
run_one_cycle:
    for   (I=0;I<MAX_ITERATION;I++) {
        clock_begin();
        module_a();
        module_b();
        …
        module_…();
    }
    clock_end ();
```

其中，clock_begin 把寄存器输入端的值恢复为本拍开始时输出端的值，clock_end 把输入端的值复制到输出端，即使得寄存器的新值对下一拍可见。MAX_ITERATION 是为了达到稳定状态需要的循环次数。每次循环都有一些模块可以生成有效输出，而这个有效输出又将使下一个循环中更多的模块得到有效输入从而生成有效输出。迭代若干次之后，信号必定会稳定下来。仔细安排模块调用次序可以减少模块迭代需要的次数。3.3 节中我们将进一步讨论减少模块重复调用次数的方法。

3.2 性能分析功能设计

为了有效地进行性能分析工作，我们为 ICT-godson 增加了数据收集、调试、可视化和批处理支持等功能。

3.2.1 数据收集

ICT-godson 不仅能够产生基本的事件统计数据（如 IPC，CACHE 失效率等），还可以按流水级、按指令 PC 或者按数据分布产生统计数据。这些统计数据从不同的角度来展示目标处理器的行为，能够有效地帮助设计者理解和优化系统。按

流水级的统计数据包括每个流水级的输入情况、输出情况以及不能有效输出的具体原因分布等。按指令 PC 的统计数据包括每条指令执行的次数、遇到的各种事件（CACHE 不命中，分支误预测等）次数、平均在每个流水级呆的拍数以及该指令运行过程中处理器的 IPC 估计等等。按数据分布统计可以生成某些目标参数（如 PC、访存地址等）的统计分布。

事件计数统计信息的一个样例如下：

表 3.1 事件计数统计输出样例

统计量	*值*	*说明*
Clock count	139331863	总运行拍数
Commit bus 0 count	76411664	第一条指令提交总线的有效次数
Commit bus 1 count	47972945	第二条指令提交总线的有效次数
Commit bus 2 count	25411789	第三条指令提交总线的有效次数
Commit bus 3 count	17835246	第四条指令提交总线的有效次数
IPC	1.20	平均每拍执行的指令数
Branch bus count	26736754	运行的转移指令总数（包括猜测执行）
Branch error count	2258069	转移猜测错误的数目
Branch miss rate	0.084	分支预测失效率
Jr inst. Count	8252985	运行的 Jr 指令总数
Jr miss count	1001500	Jr 猜测错误次数
…		略（函数返回次数，RAS 命中率，各类指令数等）
OS clock count	702274	在核心态运行的拍数
…		（核心态其它统计）
Load inst. count	48273021	Load 指令数
Load miss count	1623	Load CACHE 不命中次数
…		（其它存储相关统计，例外次数等等）

更多的例子参见第七章。

3.2.2 调试

ICT-godson 模拟器是一个比较复杂的系统，为了方便状态观察和错误查找，我们为它设计了比较灵活的调试功能。

首先，和一般的调试器类似，ICT-godson 提供了运行、单步执行、设置断点、单步跟踪等功能。每一拍的各种微体系结构状态都可以通过命令行接口查看或者

记录到文件中。

ICT-godson 实现了检查点（checkpoint）支持，它能够把某个时刻的处理器内部状态记录到文件，利用该文件以后可以迅速恢复到该时刻的状态。检查点在定位错误的过程中非常有用。例如，如果模拟器运行 10 亿拍时发现错误，我们可以每个 1 亿拍保存一个检查点，然后从离错误现象最近的检查点开始，寻找错误的根源。检查点也能用于节省性能分析的时间，例如，我们可以把操作系统完成启动时的状态保存为一个检查点，并利用这个检查点节省模拟操作系统启动的时间。

3.2.3 可视化

我们实现了一个基于 TCL/TK 的可视化界面，用于直观地观察处理器内部状态变化，如图 3.1 所示。这种方式能有助于理解处理器的工作原理，也可以用于分析一些典型代码段的行为。

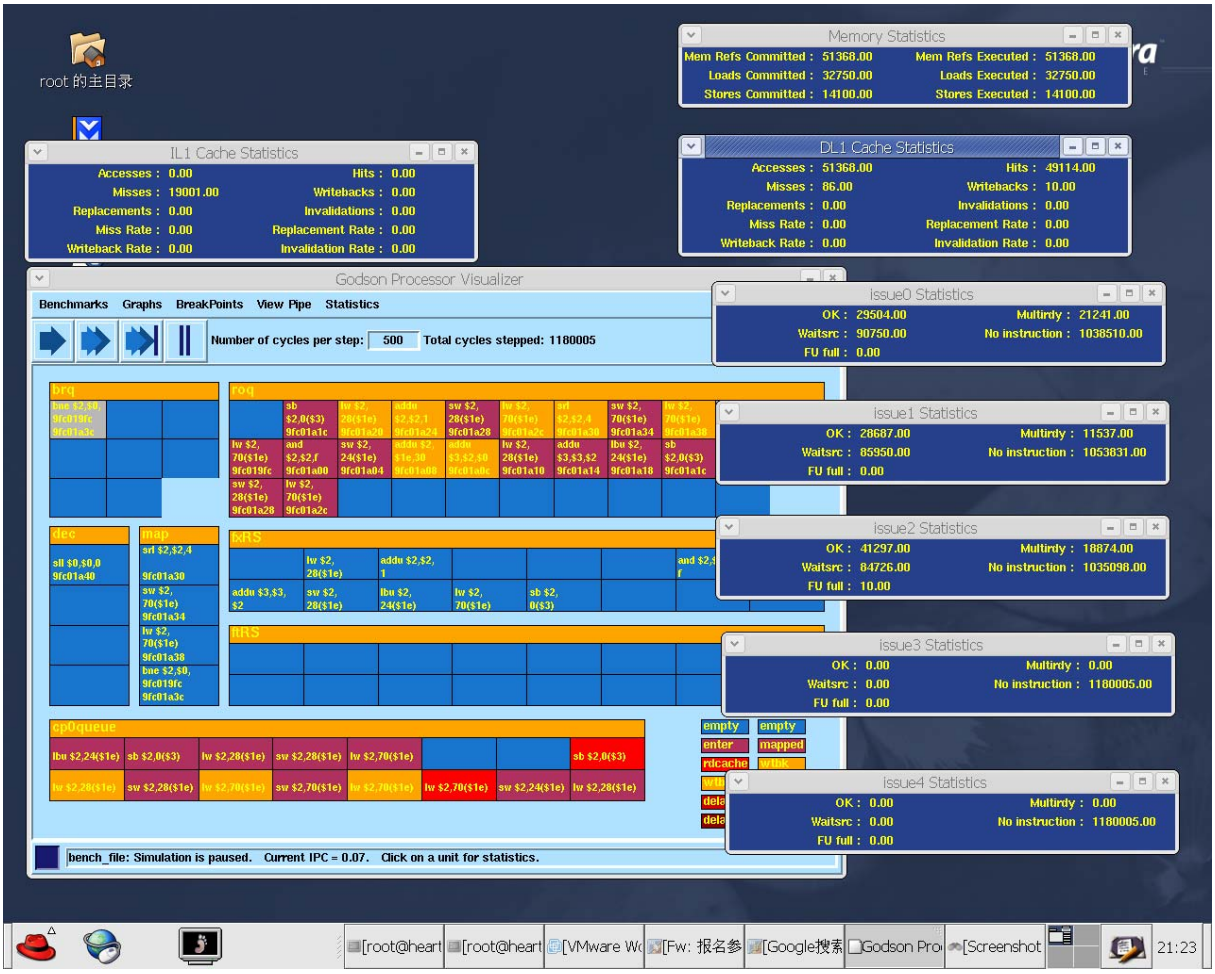


图 3.1 ICT-godson 的可视化界面

3.2.4 批处理

高性能处理器的性能分析需要处理大量的配置/工作负载组合，通常离不开一个处理能力很强的机群。ICT-godson 的性能分析是在 PC 机群上完成的。模拟器的运行和数据处理工作如果仅靠人工进行，不仅工作量极大，还容易导致错误。为此，我们引入了作业管理系统，并开发了许多自动化脚本来进行数据收集和分析。例如，要考察某个配置上所有 SPEC CPU2000 程序的性能，只需要一个命令将该配置提交给作业管理器。作业管理器会选择空闲节点运行所有的模拟，最后从各节点收集生成的数据，并自动生成汇总表格。

3.3 设计加速

前面提到的建模技术虽然有效地解决了硬件描述的问题，但是不加优化地实现这些技术有比较大的代价。寄存器输入端和输出端之间的拷贝非常耗费时间，循环调用更是成倍地增加代价。另外，早期的代码都是按照硬件思路设计，有时会很低效。对于性能分析和验证目的来说，往往只要保持模块级接口上的一致就足够了。这种情况下我们可以利用 C 语言的一些数据结构和算法改写某些模块的内部逻辑以提高效率。

优化之后 ICT-godson 每秒能够模拟 50-100K 个处理器周期（在 AMD64 3200+ 微机上），比未优化版本提高了 3 倍以上。

3.3.1 消除寄存器拷贝

这里说的寄存器其实还包括各类存储器，例如 CACHE 和分支预测器的 RAM 等等，因为它们也有类似的读写次序问题。通过仔细分析代码可以消除不必要的拷贝。

对于 clock_begin 中的拷贝，只要模块没有引用寄存器输入端的值，而且寄存器不是有条件赋值（否则可能第一遍调用时条件错误地成立，从而写入错误的值，而最后一遍调用时条件不满足没有修改它），则应该可以安全地略去。

Clock_end 中的拷贝可以通过仔细考虑对寄存器赋值和引用的代码来削减。只要能保证引用都出现在赋值后面（要考虑多个循环的情况），则可以把输入端和输出端用同一个变量表示，从而彻底消除拷贝。在模拟器编程风格上做一些要求可以帮助消除拷贝。例如，我们应该尽量把生成输出的代码放在输入处理代码前面。

3.3.2 主循环削减和代码分区

模块调用次序对达到稳定需要的循环次数影响很大。一些性能模型，如 Simplescalar 和 PowerPC 的 MET，通过按流水级倒序调用的方法，只需要一次循

环。但是在信号级模型中，流水级之间向前向后的信号都很多，不能照搬这个方法。最初 ICT-Godson 模拟器中调用次序由人工安排（可以通过代码分析将这个过程自动化），主循环为两次，但个别模块调用了不止两次。这两次循环中有许多不必重新运行的代码，我们通过代码分区技术把它们隔离开来，尽量减少重复执行。

每个模块的代码可以划分成三个部分：第一部分只由寄存器值生成输出总线；第二部分通过输入总线和寄存器生成下一拍的寄存器；第三部分则通过输入总线和寄存器生成输出总线。第一部分只依赖内部状态，只要在第一个循环运行就够了，第二部分则不影响其它模块,只要输入准备好后运行一次就可以。第三部分则看情况决定，一般需要两个循环都运行，但这种代码很少。每次调用模块时，根据当前循环次数来决定三个部分是否运行。

最终优化的结果是绝大部分的代码只运行一次，而且代码分区之后更多的寄存器拷贝能够被消除。

3.3.3 数据结构和算法替换

早期的 ICT-godson 模拟器作为一个硬件设计的可执行设计文档，是完全按照硬件思路写的，有时会很低效。另外，一些常用工具函数的效率有时也影响很大。利用编译器的 Profile 支持，我们可以找出模拟器耗费时间最多的部分，然后设法加速这些部分。

例如，ICT-godson 的 Load/Store 队列模块，它一度占 24%以上的总运行时间。使用主循环削减和代码分区之后大约减少了一半的时间，再通过各种代码变换最后减少到 4%左右。主要手段包括：

- 总线无效时避免空操作。早期 ICT-godson 按硬件的习惯，不管是否有有效输出，输出总线每个域的生成逻辑都会运行。可以利用有效位来避免没有意义的逻辑操作。
- 及早退出循环。很多队列操作都是遍历整个队列从中找出满足某个条件的项。针对硬件的写法通常对所有的项生成是否满足的条件，然后从中挑出满足的项。C 模拟器中可以利用队列的有序性，往往只需要查看很少的几项就能完成任务。

如果只要求保持模块级接口信号的话，我们可以采用一些能有效降低复杂度的数据结构来提高某些部件模拟的效率。例如，用哈希表来表示 TLB 可以大大加快其查找速度。

关于工具函数优化的一个例子是拷贝函数的优化。早期代码中数据拷贝通过一个拷贝函数来完成，该函数使用 for 循环逐字节拷贝，对大块存储效率很低。Profile 的结果表明该函数占用了可观的时间，替换成 memcpy 库函数后节省了不少时间。

第四章 Sim-godson 模拟器的设计和实现

ICT-godson 模拟器模拟了处理器所有细节，可以非常准确地反映处理器的行为，但是这同时也造成它的速度和灵活性受到比较大的限制。一些性能分析的重要任务，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，我们设计了 Sim-godson 模拟器。Sim-godson 是基于 Simplescalar 工具集开发的执行驱动的性能模拟器，和 ICT-godson 相比它有一些显著的优点：

- 运行速度快。我们采用了一系列加速技术来保证模拟器的速度。目前，在 3.0GHz 的 Pentium4 微机上，它每秒能够模拟约 500K 个处理器周期。
- 灵活性高。一方面，我们采用功能模拟和时序模拟分离的执行驱动方式，降低了实现困难。另一方面，我们充分利用 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。
- 支持大程序评估。Sim-godson 实现了多种加速大程序评估（如 SPEC CPU2000 reference 输入数据集）的方法，包括 Simpoint[SPH02]，SMARTS[WWF03]。

此外，它还支持 WATTCH 功耗模拟器，以及基于 SIMOS 的全系统模拟等。

本章讨论 Sim-godson 模拟器的设计和实现。首先我们给出 Sim-godson 的基本设计，接着讨论全系统模拟的实现以及快速评估方法，然后讨论模拟器的验证工作。

4.1 基本设计

Simplescalar 工具集是一个可用于实现微体系结构性能模型的系统框架，它包含一些参数化的常用微体系结构模块，如分支预测器，CACHE 层次；也包含实现性能模型所需要的参数选择、调试、统计、可视化等基础模块。Simplescalar 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实现分离，还提供了一系列用于各种目的的参考模拟器。

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 体系结构支持和实现龙芯 2 号微体系结构。

4.1.1 MIPS 体系结构支持

在 Simplescalar 的架构下，实现一个新体系结构的支持主要包括三项工作：

- 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。

- 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。

- 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，使得它看起来象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。

4.1.2 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[DeB01]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 针对龙芯 2 号处理器完全重新实现了微体系结构，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：

- 执行驱动

Sim-godson 和 Sim-outorder/Sim-alpha 一样，都采用执行驱动。它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访问是否命中（以计算访存指令的延迟），而不关心实际的数据。大体上我们可以说性能模型里只有控制通路，没有数据通路。这样可以减少大量的数据流动和处理操作，提高运行速度。

- 逆向调用流水

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，只调用一遍的话，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。

- 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）。误预测的分支被纠正时，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的寄存器读和访存操作接口需要相应改变：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。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。

- 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。因此，Sim-godson 采用类似 Sim-outorder 的做法，只维护指令间的依赖关系。但是，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。

- 功能部件

功能部件采用类似 Sim-outorder 的事件调度方式。这样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，使它能够模拟可变延迟的功能部件、端口竞争等情况。

- 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。

- 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。

Sim-godson 的设计非常重视速度问题。除了在组织形式和各部件模拟方法上充分考虑速度之外，我们也采用了一些软件实现上的优化，包括：

- 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。
- 用指针传递来完成指令在流水级间的流动，最小化数据复制。
- 用一个数据结构表示每条运行中的指令所有信息。系统初始化时分配足够的指令信息结构，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。

4.2 全系统模拟支持

Simplescalar 工具集只能模拟单进程、静态链接的用户态代码。虽然到目前为止，这种方式仍然为学术界所广泛使用，但它的局限性已经越来越明显了。

一方面，工作负载在多样化，以 SPEC CPU2000 为代表的计算密集型工作负载已经不能满足各类研究的需要了。而一些新的工作负载，如 web 服务器、电子商务应用等，对操作系统的依赖很大，只模拟用户态代码无法得到准确的结论。

另一方面，近年来，多处理器系统，包括片内多处理技术 CMP、SMT 等得到迅速发展。评估这些技术，也离不开全系统模拟。

此外，通过龙芯 2 号处理器的性能分析工作，我们认识到，高性能微处理器的性能不仅仅取决于处理器核心。处理器的接口、外围芯片组、编译器以及操作系统等都会影响最终系统性能。因此，要准确预测实际系统的性能，需要进行全系统模拟。

Sim-godson 以一种新颖的方式实现了全系统模拟和用户级模拟的同时支持。Sim-godson 的处理器性能模型既可以独立进行用户级模拟，也可以作为 SIMOS 的一个 CPU 模块参与全系统模拟。这样我们可以根据不同的需求选择使用哪种模式，还可以利用两种模式的差别来分析操作系统对性能的影响。

4.2.1 SIMOS

SIMOS[RoH95]是一个优秀的全系统模拟器，DEC 和 IBM 两个大处理器厂商都曾使用它来进行体系结构研究，学术界也不少研究小组使用它进行各种研

究。但是，可以免费获得的 SIMOS 版本停止开发已经多年，它所带的一些软件已经过时了。更重要的问题是它运行的是一个商业操作系统，一般用户无法进行涉及操作系统的修改。为了让 SIMOS 在龙芯 2 号性能分析环境发挥作用，我们对它进行了一系列改造。

- Linux/MIPS 操作系统支持

  SIMOS 可以看作一个虚拟机，要在其上运行新的操作系统有两种方法：在 SIMOS 中增加新操作系统所支持的硬件，或者在操作系统里增加现有虚拟硬件的驱动支持。我们综合使用这两种方法。在 SIMOS 端，我们新增了标准 8255 串口硬件支持，这是 Linux/MIPS 内核支持的控制台设备；我们对地址空间的安排做了一些改动，使它顺应 Linux/MIPS 内核的要求。在操作系统内核里，我们增加了对 SIMOS 虚拟机的识别和初始化；增加了对 SIMOS 的虚拟 SCSI 磁盘、虚拟网卡等设备的驱动支持。

  SIMOS 运行时需要装载操作系统内核。Linux/MIPS 内核是一个 ELF 格式的文件，为此我们实现了 ELF 文件装载的支持。

- 新版 gdb 支持

  SIMOS 支持用 gdb 交叉调试被模拟系统，这是一个非常有用的功能。但是可获得的 SIMOS 软件包中的 gdb 版本很低（4.16），不能支持现代 Linux/MIPS 系统的程序。我们在新版本的 gdb（5.3 和 6.0）上重新实现了 SIMOS 调试插桩（stub），使得它们能够用于交叉调试 SIMOS 所模拟的 Linux/MIPS 系统。

- Little-endian 支持

  原始的 SIMOS 系统只支持 Big-endian，我们为它增加了 Little-endian 系统的支持。

- TCL 脚本

  SIMOS 的一大特色是它可以用 TCL 脚本来灵活控制模拟器的运行。许多用于性能分析的 TCL 脚本是针对 IRIX 操作系统写的，需要根据 Linux 操作系统的情况重新编写。

- dwarf 调试符号

  原始的 SIMOS 版本不能解析 ELF 文件的调试符号。这使得 SIMOS 无法使用符号名称来引用内核函数和变量，TCL 脚本编写变得很困难。我们更新了相关代码，解决了这个问题。

4.2.2 Sim-godson 和 SIMOS 的融合

SIMOS 提供一个简洁的 CPU 模块接口，它把 CPU 模块内部实现和外部很好地隔离开来。但是把 Sim-godson 改造为一个 SIMOS 的 CPU 模块仍然是一个不容易的工作，我们进行了如下改造工作：

- 把 Sim-godson 的处理器内部状态封装到一个结构，以便 SIMOS 实例化多个处理器。
- 实现和提供 SIMOS 需要的接口，包括初始化、切换、单步运行、调试等。
- 系统态支持。Sim-godson 原来没有实现系统态指令，例外处理也很简单。我们需要补全缺少的指令，实现对系统态指令的处理，并完善例外处理。

为了保持 Sim-godson 用户级模拟的能力，和 SIMOS 相关的代码都用#ifdef 和#endif 隔离。

4.3 快速评估支持

现代高性能微处理器的结构复杂度及其工作负载的规模都在不断增加，这给基于模拟的体系结构研究带来了很大的挑战。例如，最快的现代微处理器详细模拟器的模拟速度大约是每秒几百 K 条指令。而使用 reference 输入集时，SPEC CPU2000 中平均每个程序/输入对需要运行的动态指令数接近 2000 亿（参见表 4.2），完全模拟需要几天甚至几个月的时间。

实际研究中人们采用各种方法来解决模拟时间的问题。最常用的大程序快速评估方法可以分为下列三类。

- 截断法

  这种方法只模拟整个程序的一个片断。具体形式有几个变种，最简单的一种方法是从头开始运行一定指令数后停止；或者先快速模拟前 X 条指令（希望能跳过不感兴趣的初始化部分）再切换到详细模拟方式模拟 Y 条指令；进一步地，可以在快速模拟 X 条指令后，详细模拟 Y 条指令来预热微体系结构状态，然后再统计 Z 条指令。

- 缩小输入集

  例如使用 SPEC CPU2000 自身提供的 test 或者 train 输入集，以及 MiniSPEC[KFM01]的 reduced 输入集。这种方法寄希望于缩小输入集的特性和原输入集相似。它的优点是程序的所有部分，包括初始化、主体计算过程和结束部分都能考验到。但几位研究者的经验[KFM01, VaB04]表明，保证某些程序的缩小集特性和原输入集特性全面相似非常困难。

- 采样法

  采样方法通过一些样本的情况来估计整体。采样方法是否准确的关键在于选择的样本能否反映整体的情况，一个好的采样方法应该给出误差估计和控制的方法。采样方法可以分为三类：采样有代表性部分、周期性采样、随机采样。

  第一类是采样有代表性部分的方法。这种方法通常要先分析整个程序，

然后用一定的统计方法选择有代表性的样本。这种方法的一个代表是 Simpoint[SPH02]。Simpoint 先完整地运行程序（用踪迹生成工具或快速模拟器），得到每个样本中所有基本块执行次数所构成的基本块矢量（BBV, basic block vector[SPC01]），再根据 BBV 的相似性通过聚类算法把所有样本分成若干类，然后为每类挑选一个样本来做为代表，最后得到一组作为代表的样本。BBV 与具体结构没有关系，因此选中的样本并不依赖于具体微体系结构。

周期性采样方法的一个近期例子是 SMARTS[WWF03]。SMARTS 利用使用采样原理和程序运行的特性来实现误差估计和控制。如果误差超过用户指定范围，它能自动推荐一个合适的采样频率。

随机采样是从所有样本中随机抽取 N 个样本，把它们的结果组合在一起作为最终估计。Conte 等人[CHM96]发现，可以通过增加样本数目或者每个被选中样本的预热时间来减少随机采样的误差。

Yi 等人[YLS05]从性能瓶颈分析误差、代码运行特性（BBV 等）、体系结构特征值（IPC、CACHE 失效率等）三个特征角度深入比较了以上各种方法的精确性，并分析了不同方法的速度以及它们对特定配置的依赖性。他们的结论包括：

- 缩小输入集和截断法的精确性都很差，配置依赖性高，而且速度也没有更快。
- 所有三个特征角度的分析数据和配置依赖性分析都表明 Simpoint 和 SMARTS 是非常精确的技术。SMARTS 精确程度略高一些，而 Simpoint 更快。它们都能够精确地得出加速比结果。

本文在 Sim-godson 中实现了 Simpoint 和 SMARTS 技术，并对它们的加速比和误差进行了评估，还提出了一些改进方法。下面我们介绍这些工作。

4.3.1 Simpoint 实现和评估

使用 Simpoint 首先需要得到目标工作负载的 BBV。生成 BBV 有两种方法，使用模拟器或者使用踪迹生成工具。

BBV 和微体系结构无关，因此生成 BBV 只需要能够解释执行指令的功能模拟器。这样的模拟器相对比较快：在 3.0G Pentium4 处理器上，Sim-godson 的快速版本每秒可以模拟 20M 条以上的指令。以这个速度运行的话，每个 SPEC CPU2000 程序/输入对都可以在几个小时内完成模拟。

使用踪迹生成工具的方法速度更快。我们基于 ALTO 开发了一个踪迹生成工具，它能够在 MIPS 可执行程序的每个基本块插入代码来进行运行次数统计。运行修改过的程序就能快速得到它的 BBV。使用这种方法需要注意一个问题。因为实际机器的运行环境和模拟器运行环境有一些差别，两边运行的指令序列并不

完全一致，使用指令数来指定选中的样本有时会导致偏差。一种解决的办法是通过某些指令的执行次数来指示样本的开始。

得到 BBV 之后，要使用 Simpoint 算法来选择样本，这些样本被称为模拟点（simpoint）。最后，我们使用模拟器详细模拟选定的样本，把各样本的数据按其权重综合起来得到对整体性能的估计。

我们用 Sim-godson 模拟器对 Simpoint 技术进行了加速比和误差分析。我们使用的模拟器配置如表 4.1 所示，运行的程序为 SPEC CPU2000（其中个别程序由于无法正确运行没有给出数据，如 4 个 Fortran90 程序），使用 reference 输入集。模拟器在一个 32 节点的 PC 机群上运行，其中每个节点的处理器为 3.0GHz Pentium 4。

表 4.1 Simpoint 分析使用的微体系结构配置

Fetch:decode:issue:commit width	4:4:5:4:4	主流水级	7 级
功能部件	2 定点，1 个访存，2 浮点	功能部件 load-use 延迟	定点 ALU 1，乘 8，除 12 浮点加 4，乘 5 访存（CACHE 命中）4
Roqueue	32	转移队列	8
定点发射队列	16	存储访问队列	16
浮点发射队列	16	失效访问队列	2
分支预测器	Gshare: 9 位 ghr，4096 项 pht. 128 项 BTB，两路组相联		
L1-ICACHE	64KB 4 路组相联 随机替换	L1-DCACHE	64KB 4 路组相联 随机替换
L2	1MB 直接相联，随机替换，延迟 18 拍		
访存延迟	第一个子块返回延迟 60 拍，子块间隔 3 拍		

表 4.2 Simpoint 的加速比

bzip2.source	117548094573	46.37	900000000	0.78	59.45
crafty	321508026022	121.52	800000000	0.44	276.18
eon.cook	121140807829	55.66	800000000	0.38	146.47
eon.kajiya	156197444895	66.33	700000000	0.30	221.10
eon.rushmeier	86431038281	36.57	800000000	0.33	110.82
gap	305951922287	117.22	700000000	0.33	355.21
gcc.166	44530861131	21.55	800000000	0.87	24.77
gcc.200	111288018911	46.51	900000000	0.63	73.83
gcc.expr	12197402147	5.42	1000000000	0.41	13.22
gcc.integrate	12877552938	5.61	900000000	0.39	14.38
gcc.scilab	63635747373	27.61	1000000000	0.42	65.74
gzip.graphic	117086666996	41.00	700000000	0.35	117.14
gzip.log	43892161089	14.60	700000000	0.44	33.18
gzip.program	170570793518	56.92	700000000	0.33	172.48
gzip.random	96407260916	33.23	700000000	0.37	89.81
gzip.source	88642520511	29.67	1000000000	0.47	63.13
mcf	83662461348	66.85	800000000	0.63	106.11
mesa	343913277941	125.41	700000000	0.26	482.35
parser	528584578964	222.79	900000000	0.38	586.29
swim	334715646128	151.15	900000000	0.42	359.88
twolf	383659833526	194.47	1000000000	0.46	422.76
vortex.vortex1	152974914896	65.95	600000000	0.52	126.83
vortex.vortex2	173530154512	74.48	800000000	0.69	107.94
vortex.vortex3	170364922329	73.66	700000000	0.63	116.92
wupwise	378970576315	162.11	600000000	0.30	540.37
avg.	189861807389	80	796774194	0.48	195

Sherwood 等人[SPH02]和 Wunderlich 等人[WWF03]都使用快速模拟的方法来跳过没有选中的样本，如果一些选中的样本在运行的后期，这种方法需要花比较多的时间。本文使用检查点技术来避免快速模拟的开销，进一步减少了运行时间。从表 4.2 可以算出，我们的方法平均只需要 0.48 小时就能完成模拟，而 Wunderlich 等人报告的数据为平均 2.8 小时。

表 4.3 Simpoint 的精确度

Program	IPC-f	IPC-s	IPC-e	DL1-f	DL1-s	DL1-e	BR-f	BR-s	BR-e
ammp	0.678	0.671	-0.010	0.050	0.052	0.047	0.039	0.040	0.036
apsi	0.636	0.660	0.038	0.030	0.025	-0.172	0.056	0.064	0.133
art.art1	0.299	0.298	-0.002	0.489	0.490	0.002	0.051	0.052	0.012
art.art2	0.302	0.302	-0.002	0.491	0.491	0.000	0.051	0.050	-0.014
bzip2.graphic	0.865	1.121	0.296	0.016	0.009	-0.432	0.064	0.055	-0.133
bzip2.program	0.948	1.025	0.082	0.017	0.017	0.015	0.066	0.048	-0.281
bzip2.source	0.816	0.866	0.060	0.023	0.023	0.031	0.072	0.075	0.037
crafty	0.903	0.906	0.004	0.003	0.003	0.011	0.267	0.266	-0.004
eon.cook	0.734	0.731	-0.004	0.000	0.000	0.545	0.149	0.149	-0.003
eon.kajiya	0.712	0.710	-0.002	0.000	0.000	1.667	0.161	0.161	-0.002
eon.rushmeier	0.776	0.768	-0.010	0.000	0.000	0.667	0.153	0.152	-0.001
gap	0.966	0.492	-0.491	0.012	0.032	1.630	0.124	0.085	-0.311
gcc.166	0.543	0.530	-0.024	0.076	0.058	-0.237	0.045	0.054	0.183
gcc.200	0.732	0.744	0.017	0.034	0.024	-0.299	0.116	0.136	0.170
gcc.expr	0.707	0.730	0.033	0.042	0.031	-0.265	0.116	0.132	0.134
gcc.integrate	0.676	0.671	-0.007	0.060	0.046	-0.228	0.072	0.090	0.248
gcc.scilab	0.696	0.721	0.035	0.034	0.024	-0.297	0.137	0.138	0.008
gzip.graphic	1.232	1.228	-0.003	0.020	0.019	-0.020	0.080	0.081	0.014
gzip.log	1.174	1.194	0.017	0.032	0.030	-0.062	0.050	0.049	-0.018
gzip.program	1.312	1.189	-0.094	0.033	0.061	0.848	0.057	0.058	0.025
gzip.random	1.179	1.177	-0.002	0.026	0.025	-0.022	0.054	0.053	-0.011
gzip.source	1.224	1.213	-0.009	0.035	0.037	0.070	0.064	0.066	0.031
mcf	0.184	0.185	0.004	0.266	0.269	0.011	0.071	0.077	0.088
mesa	1.232	1.238	0.005	0.004	0.004	-0.002	0.117	0.116	-0.003
parser	0.870	0.892	0.025	0.020	0.019	-0.025	0.086	0.087	0.012
swim	0.419	0.431	0.028	0.178	0.174	-0.023	0.005	0.003	-0.396
twolf	0.560	0.556	-0.007	0.069	0.069	0.004	0.227	0.228	0.004
vortex.vortex1	0.646	0.629	-0.027	0.008	0.009	0.048	0.081	0.083	0.031
vortex.vortex2	0.739	0.723	-0.022	0.005	0.005	0.006	0.094	0.098	0.043
vortex.vortex3	0.634	0.623	-0.018	0.008	0.009	0.017	0.081	0.084	0.037
wupwise	0.942	0.952	0.010	0.023	0.023	-0.009	0.179	0.177	-0.010

注：-f 为完全运行的结果，-s 为 simpoint 结果，-e 则是-s 对-f 的误差。DL1 表示一级数据 CACHE 失效率，BR 表示分支预测失效率。

表 4.2 和表 4.3 分别给出了 simpoint 的加速比和精确性情况。其中的 simpoint 样本大小为 100M 条指令，生成 simpoint 使用 K=10 的约束，即最多把样本分为 10 类。每类只取一个样本，因此需要模拟的指令最多只有 10*100M＝10 亿条，在 Sim-godson 模拟器上可以在一小时之内完成。加速比从 13 到 586 不等，一般运行时间越长的程序加速比越大。

表 4.3 中我们给出了 IPC、一级数据 CACHE 失效率和分支预测失效率的误

差。可以看到，对大多数程序，Simpoint 具有相当高的精确性。多数程序的 IPC 误差甚至在 1%以内。对于 IPC，明显不正常的程序包括 bzip2 和 gap，此外只有 apsi、gcc 和 gzip（对某些输入）的误差略高。对一级数据 CACHE 失效率，除了 bzip2 和 gap，apsi、eon、gcc 和 gzip.proram 的误差比较大。而分支预测失效率误差较大的程序是 apsi、bzip2、gap、gcc 和 swim。

从表 4.2 可以看到，apsi、bzip2、gap 和 gzip 的总指令数都比较高，因此有可能 K=10 时，由于每类的样本很多，用一个样本难以精确代表。Gcc 的程序行为比较复杂（参见第 7 章对程序周期性的分析），K=10 也不足以精确分类所有样本。事实上，从表 4.2 可以看到，对 gcc 用 K=10 分类，结果都得出了 9 个或者 10 个 simpoint，说明 gcc 中不相似的样本比较多。至于 eon 的一级 CACHE 失效率误差和 swim 的分支预测失效率误差，由于这些失效率的绝对值很小，这些误差不会对性能估计有明显的影响。

Simpoint 算法能够对所产生的 simpoint 给出误差估计，即该样本和它所在类的样本相似程度的方差。例如，对 K=10 时，gap 的 simpoint 方差如下：

Gap output of SimPoints, 7 centers:

Variances/cluster:

0.0589272 0.0426404 0.101805 0 0.052506 0.249623 0.0368992

其中第 6 个样本的方差明显比较大。不过，bzip2 的结果虽然比较异常，其 simpoint 的方差却没有特别异常地大。

用 K=20 重新生成 simpoint，得到的结果如下：

表 4.4 增加 K 对 simpoint 误差的影响（K=20）

Program	***IPC-f***	***IPC-s***	***IPC-e***	***DL1-f***	***DL1-s***	***DL1-e***	***BR-f***	***BR-s***	***BR-e***
bzip2.graphic	0.8645	0.8820	0.0202	0.0163	0.0177	0.0879	0.0637	0.0664	0.0424
gap	0.9663	0.9554	-0.0113	0.0123	0.0133	0.0813	0.1237	0.1321	0.0679
gzip.program	1.3124	1.3006	-0.0090	0.0331	0.0346	0.0453	0.057	0.0635	0.1140
gcc.166	0.5428	0.5634	0.0380	0.0762	0.0523	-0.3136	0.0453	0.0599	0.3223
gcc.200	0.732	0.745	0.018	0.034	0.025	-0.251	0.116	0.137	0.175
gcc.expr	0.7066	0.7021	-0.0064	0.0416	0.0313	-0.2476	0.116	0.1317	0.1353
gcc.integrate	0.6757	0.6839	0.0121	0.0723	0.0458	-0.3665	0.0601	0.0825	0.3727
gcc.scilab	0.6963	0.6879	-0.0121	0.0341	0.0285	-0.1642	0.1369	0.1465	0.0701

我们可以看到，在 K=20 时，所有程序的 IPC 误差都得到了很好的控制。bzip2、gap 和 gzip.program 的分支误预测率和一级 cache 失效率误差也降到了可以接受的程度。但是 gcc 的一级 CACHE 失效率和分支误预测率误差仍然没有好转，甚至有所下降。一方面，K=20 可能对 gcc 仍然是不够的。另一方面，由于 simpoint 的选择仅依赖于基本块执行次数，在 BBV 上相似但 CACHE 失效率有很大差别的样本会被分为一类，发生这种情况时 Simpoint 可能出现比较大的误差。事实上 Wunderlich 等人[WWF03]指出，gcc 中确实存在这种情况。

Perelman 等人后来提出了一种改进方法[PHC03]。这种方法生成多种候选的

simpoint 集（按最多分成 n 类生成，1<=n<=K），然后通过一次完全的详细模拟来选择其中样本 IPC 方差最小的 simpoint 集。这样可以避免出现上述情况。他们发现，如果使用 1M 指令的样本大小，指定 K=300，能够很好地控制 gcc 的误差。

Perelman 等人的方法有几点不足：(1)、它要求一次完整的详细模拟，时间代价比较大。(2)、为了得到方差比较小的 simpoint 集，K 通常要设得比较大；如果使用检查点技术，K 值对磁盘空间需求的影响比较大。(3)、使用详细模拟来辅助选择模拟点集使得选择结果对具体的微体系结构产生一定的依赖。本文认为，另一种解决方法是增加第二维的度量来辅助某个样本类中代表样本的选择。我们首先根据 Simpoint 算法按 BBV 分类样本，然后根据第二维度量继续细分每个样本类，为每个子类选择代表样本。Simpoint 的主要缺陷在于它有时会低估存储系统的影响，因此各样本的 CACHE 命中率应该是一个很好的第二维选择度量。CACHE 命中率生成已经是一个很成熟的技术[Mat70]，我们不仅不需要进行详细模拟，还可以用一次运行同时获得多个配置的数据，减小对具体微结构的依赖。

4.3.2 SMARTS 实现和评估

SMARTS 是一种严格的系统采样方法。首先，我们要确定样本大小 U（指令数）、采样频率 K 以及预热指令数 W。SMARTS 把每 K*U 条指令作为一个采样单元，先用快速模拟（K-1）*U – W 条指令，再详细模拟 W 条指令，然后开始采样统计，继续详细模拟 U 条指令。

为了支持 SMARTS，模拟器必须同时支持快速功能模拟和详细模拟，以及它们之间的切换。模拟器还需要支持有选择的数据统计，以便支持预热。MIPS 指令集的转移指令延迟槽特性给模式切换带来一些困难，因为转移指令和其延迟槽指令间有一些紧密相关的信息，如果在转移指令处切换模式则需要同时转移这些信息。我们把转移指令和它的延迟槽作为一个原子单位处理，解决了这个问题。

表 4.5 和表 4.6 分别给出了 SMARTS 的加速比和误差情况。其中，模拟器配置和程序同上节。SMARTS 的参数为 U=1000，K=1000，W=2000。

从表 4.5 中可以看到，平均每个程序需要详细模拟的指令数为 5.7 亿，平均模拟时间为 4.69 小时，和 Wunderlich 等人的结果基本一致。

表 4.6 显示，对 SPEC CPU2000 程序，SMARTS 预测的 IPC 误差平均为-1.7%。其中误差最大的程序为 crafty，-8.4%。SMARTS 预测的一级 CACHE 失效率误差极小，排除绝对值太小的情况外，只有 crafty 和 vortex2 误差超过 1%。但是，SMARTS 预测的分支预测失效率误差有时比较大。

表 4.5 SMARTS 的加速比

Program	*FullInsts*	*FullTime(h)*	*detailed Insts*	*Smarts(h)*	*Speedup*
ammp	405836710244	176.52	1238352148	10.1294	17.43
apsi	642421578030	257.47	1934484733	20.2903	12.69
art.art1	57527606984	37.49	172836430	1.5569	24.08
art.art2	63816096833	40.82	191731777	1.7092	23.88
bzip2.graphic	158890910512	64.27	477417171	3.4583	18.58
bzip2.program	136940441090	52.80	411462240	2.9556	17.86
bzip2.source	117548094573	46.37	353204189	2.5919	17.89
crafty	321508026022	121.52	965995323	7.6231	15.94
eon.cook	121140807829	55.66	365163593	2.9883	18.63
eon.kajiya	156197444895	66.33	471065935	3.8167	17.38
eon.rushmeier	86431038281	36.57	260326696	2.0872	17.52
gap	305951922287	117.22	919213256	7.6458	15.33
gcc.166	44530861131	21.55	133718756	0.945	22.80
gcc.200	111288018911	46.51	334036567	2.4494	18.99
gcc.expr	12197402147	5.42	36611830	0.2681	20.22
gcc.integrate	12877552938	5.61	38673834	0.2761	20.32
gcc.scilab	63635747373	27.61	190982110	1.4069	19.62
gzip.graphic	117086666996	41.00	351800650	2.5781	15.90
gzip.log	43892161089	14.60	131879085	0.8742	16.70
gzip.program	170570793518	56.92	512500829	3.445	16.52
gzip.random	96407260916	33.23	289677080	2.1017	15.81
gzip.source	88642520511	29.67	266342099	1.8522	16.02
mcf	83662461348	66.85	251295774	2.4589	27.19
mesa	343913277941	125.41	1042680514	8.8142	14.23
parser	528584578964	222.79	1422541028	11.1942	19.90
swim	334715646128	151.15	1006365793	8.5514	17.68
twolf	383659833526	194.47	1152658213	10.2175	19.03
vortex.vortex1	152974914896	65.95	466230001	3.8794	17.00
vortex.vortex2	173530154512	74.48	523961348	4.5397	16.41
vortex.vortex3	170364922329	73.66	520811612	4.3244	17.03
wupwise	378970576315	162.11	1138707845	8.5003	19.07
avg.	189861807389	80	566862208.35	4.69	18

表 4.6 SMARTS 的准确性

Program	*IPC-f*	*IPC-s*	*IPC-e*	*DL1-f*	*DL1-s*	*DL1-e*	*BR-f*	*BR-s*	*BR-e*
ammp	0.678	0.678	0.000	0.050	0.0498	0.000	0.039	0.0403	0.047
apsi	0.636	0.635	-0.002	0.030	0.0302	0.000	0.056	0.0766	0.363
art.art1	0.299	0.3	0.002	0.489	0.4893	0.000	0.051	0.0513	0.000
art.art2	0.302	0.302	-0.001	0.491	0.4921	0.001	0.051	0.051	0.004
bzip2.graphic	0.865	0.864	-0.001	0.016	0.0163	0.002	0.064	0.0681	0.069
bzip2.program	0.948	0.947	0.000	0.017	0.0167	-0.005	0.066	0.0726	0.095
bzip2.source	0.816	0.814	-0.003	0.023	0.0227	-0.002	0.072	0.0777	0.078
crafty	0.903	0.827	-0.084	0.003	0.0029	0.090	0.267	0.3227	0.208
eon.cook	0.734	0.714	-0.028	0.000	0.0001	-0.091	0.149	0.138	-0.074
eon.kajiya	0.712	0.692	-0.028	0.000	0	-1.000	0.161	0.1574	-0.024
eon.rushmeier	0.776	0.754	-0.028	0.000	0.0001	0.667	0.153	0.1441	-0.055
gap	0.966	0.925	-0.043	0.012	0.0123	0.000	0.124	0.1029	-0.168
gcc.166	0.543	0.539	-0.008	0.076	0.0763	0.000	0.045	0.0429	-0.053
gcc.200	0.732	0.71	-0.030	0.034	0.0338	0.003	0.116	0.1139	-0.021
gcc.expr	0.707	0.687	-0.027	0.042	0.0419	0.008	0.116	0.111	-0.044
gcc.integrate	0.676	0.662	-0.021	0.060	0.0603	0.004	0.072	0.0675	-0.066
gcc.scilab	0.696	0.677	-0.027	0.034	0.0338	-0.010	0.137	0.1317	-0.038
gzip.graphic	1.232	1.224	-0.006	0.020	0.02	0.010	0.080	0.0849	0.063
gzip.log	1.174	1.166	-0.007	0.032	0.032	0.006	0.050	0.0603	0.213
gzip.program	1.312	1.306	-0.005	0.033	0.0331	0.000	0.057	0.0627	0.106
gzip.random	1.179	1.179	0.000	0.026	0.026	0.000	0.054	0.0545	0.009
gzip.source	1.224	1.218	-0.005	0.035	0.0347	0.003	0.064	0.0726	0.138
mcf	0.184	0.184	-0.001	0.266	0.2655	-0.001	0.071	0.0715	0.010
mesa	1.232	1.19	-0.034	0.004	0.0045	0.009	0.117	0.1141	-0.021
parser	0.870	0.859	-0.013	0.020	0.0197	0.000	0.086	0.0977	0.131
swim	0.419	0.419	0.000	0.178	0.1782	0.000	0.005	0.0049	0.021
twolf	0.560	0.545	-0.028	0.069	0.0689	-0.003	0.227	0.2508	0.104
vortex.vortex1	0.646	0.634	-0.019	0.008	0.0083	0.004	0.081	0.1022	0.270
vortex.vortex2	0.739	0.717	-0.030	0.005	0.0049	0.019	0.094	0.1249	0.330
vortex.vortex3	0.634	0.624	-0.016	0.008	0.0084	-0.007	0.081	0.1016	0.256
wupwise	0.942	0.899	-0.046	0.023	0.0229	0.004	0.179	0.1838	0.027
avg.			-0.0174						

注：-f 为完全运行的结果，-s 为 simpoint 结果，-e 则是-s 对-f 的误差。DL1 表示一级数据 CACHE 失效率，BR 表示分支预测失效率。

4.4 模拟器验证

本节我们讨论 Sim-godson 模拟器的验证。

4.4.1 验证流程

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（参见 4.4.2 节）验证模拟器

（3） 根据误差分析结果修正模拟器。回到步骤（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

a) 如果迹象显示误差可能由某些特性模拟偏差所致，回到步骤（1），增加针对该特性的微基准程序

b) 根据误差分析结果修正模拟器。回到步骤（2）

步骤（1）中关键的一点是，我们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。

步骤（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，某些程序误差小了，但另一些原先结果貌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。

步骤（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。

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，就会发现异常之处了。

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。误差根源可能是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速地定位问题。例如，对微基准程序，直接观察两个模拟器运行核心循环时的行为常常就能发现问题。

4.4.2 微基准程序

常规的应用程序运行时一般同时考验到微处理器内部多种特性，如访存、分支预测等，而现代微体系结构的复杂性使得我们很难确定误差和各部分的关系。为了隔离模拟器各个部分的建模问题，我们使用所谓微基准程序[DBK01]。每个微基准程序是一个比较简单的程序（一般是一个循环），设计来考察特定的微体系结构特性。这样，通过考察每个微体系结构特性是否被正确模拟，我们可以完

成对模拟器的一个系统验证。

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。

控制类程序考察分支预测器的设计，包括 6 个程序。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。C-recur 递归调用一个函数 1000 次，该函数把两个参数相加。这个程序可以测试转移地址栈。C-switch1、C-switch2、C-switch3 都是测试 switch-case 语句，只是选中各 case 语句的频率不同，它们用于测试 BTB 设计。C-complex 是 C-cond 和 C-switch* 的组合，用于进一步考察分支预测的综合表现。

执行类程序考察功能部件和流水线设计，包括 9 个程序。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。E-D{1-6}是一系列的相关指令链，用于考察指令调度和发射策略。D1 表示每条指令和其上一条相关，D2 则是和上上条指令相关。E-DM1 是一系列的相关整数乘法指令，用来考察整数乘法器的延迟。

访存类考察存储层次设计，包括 4 个程序。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。M-F 也包含一系列不相关的 CACHE 命中的 Load，但是 Load 的结果是一个浮点数，用于累加。M-D 包含一系列连续相关 Load，每个 Load 都在 CACHE 中命中。它可以考察 Load 指令延迟。M-M 则通过一系列 CACHE 不命中的相关 Load 指令来测试访存延迟。

我们以 ICT-godson 模拟器为基准来校验 Sim-godson。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，今后 Sim-godson 所模拟的处理器模型，很可能还没有实际硬件存在，只能和更详细的模拟器进行验证。

为了排除操作系统的随机因素，我们在两个模拟器中都使用用户级模拟的方式收集数据。

模拟器的配置参照龙芯 2 号的一个中间版本设置，如表 4.7 所示。验证结果列在表 4.8 中。

表 4.7 Sim-godson 模拟器验证使用的微体系结构配置

Fetch:decode:issue:commit Width	4:4:5:4:4	主流水级	7 级
功能部件	2 定点，1 个访存，2 浮点	功能部件 load-use 延迟	定点ALU 2，乘 4，除可变[1]

			浮点加 4，乘 5 访存（CACHE 命中）5
Roqueue	32	转移队列	8
定点发射队列	16	存储访问队列	16
浮点发射队列	16	失效访问队列	2
分支预测器	Gshare: 9 位 ghr，4096 项 pht. 128 项 BTB，直接相联		
L1-ICACHE	64KB 4 路组相联 随机替换	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		

1: 定点乘法和除法内部被分成两个子操作，除法的延迟和操作数相关。

表 4.8 微基准程序验证结果

Prog	***IPC-ICT***	***IPC-Sim***	***BR-ICT***	***BR-Sim***	***DL1-ICT***	***DL1-Sim***
C-cond	1.50	1.48	0.00	0.00	-	-
C-complex	0.67	0.68	0.25	0.25	-	-
C-recur	1.72	1.70	0.00	0.00	-	-
C-switch1	0.74	0.74	0.17	0.17	-	-
C-switch2	0.82	0.81	0.10	0.10	-	-
C-switch3	0.91	0.93	0.08	0.08	-	-
E-I	1.97	1.97	-	-	-	-
E-F	1.01	1.01	-	-	-	-
E-d1	0.67	0.67	-	-	-	-
E-d2	1.06	1.06	-	-	-	-
E-d3	1.56	1.55	-	-	-	-
E-d4	1.97	1.97	-	-	-	-
E-d5	1.93	1.95	-	-	-	-
E-d6	1.97	1.97	-	-	-	-
E-dm1	0.75	0.75	-	-	-	-
M-I	1.66	1.66	-	-	0.00	0.00
M-F	0.99	0.99	-	-	-	-
M-D	0.57	0.57	-	-	0.00	0.00
M-M	0.15	0.15	-	-	1.00	1.00

微程序验证有助于及早发现模型实现的错误。在校验过程中，我们发现了多个模型实现问题。例如，Sim-godson 早期实现中， ghr（g-share 预测器的全局历史寄存器）的更新有问题。在表达式 ghr=ghr<<1 | taken，taken 值应为 0 或 1，实际却用到饱和计数器的值，可能取 0-3，导致 C-cond 的 IPC 只有 0.55。程序的

指令统计信息显示其中一个转移指令全部猜测错误，由此我们很快发现了这个错误。

M-F 程序也暴露了一个问题。最初的 Sim-godson 中 M-F 的 IPC 为 1.23，而 ICT-godson 为 0.99。分析汇编代码和 ICT-godson 的指令统计信息可以发现，原本应该不相关的浮点 load 操作在龙芯 2 号中表现为相关。这是由于龙芯 2 号在 MIPS I 兼容模式下用一个 64 位物理寄存器存储两个 32 位浮点寄存器，写一个 32 位浮点寄存器先要把目标寄存器读出，以保留另 32 位的值，造成浮点 load 操作对目标寄存器的隐性依赖。早期 Sim-godson 中没有模拟这个特性。

一些微基准程序的 IPC 还存在微小的误差。仔细观察之后我们发现，一些随机因素可能导致两个模拟器中核心循环的初始状态不同，而循环初始状态对循环的性能影响有时会比较明显。

4.4.3 SPEC CPU2000

微基准程序主要单独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，我们使用 SPEC CPU2000 来进行更进一步的验证。作为处理器的基准程序，SPEC CPU2000 能够比较全面地考察处理器模型。

SPEC CPU2000 的验证结果如表 4.9。其中模拟器配置同上节，运行时使用 test 输入集，每个程序最多运行 10 亿条指令。

表 4.9 SPEC CPU2000 程序验证结果

prog	*ipc-ict*	*ipc-sim*	*ipc-err*	*dc-ict*	*dc-sim*	*dc-err*	*br-ict*	*br-sim*	*br-err*
ammp	0.24	0.26	0.08	0.08	0.08	0.01	0.06	0.07	0.14
applu	0.61	0.62	0.02	0.02	0.02	0.00	0.04	0.04	0.07
apsi	0.51	0.52	0.02	0.05	0.06	0.01	0.04	0.04	0.09
art	0.15	0.15	0.03	0.13	0.12	-0.01	0.03	0.03	0.06
bzip2	0.96	1.04	0.08	0.01	0.01	0.03	0.05	0.05	-0.06
crafty	1.05	1.06	0.01	0.00	0.00	0.05	0.19	0.19	0.00
eon1	0.97	0.98	0.01	0.00	0.00	0.00	0.20	0.19	-0.01
eon2	0.97	0.98	0.01	0.00	0.00	0.00	0.20	0.19	-0.01
eon3	0.92	0.93	0.01	0.00	0.00	0.00	0.21	0.20	-0.01
equake	0.82	0.83	0.01	0.01	0.01	0.04	0.11	0.11	-0.01
gap	0.83	0.84	0.02	0.01	0.01	-0.01	0.18	0.18	-0.01
gcc	0.71	0.71	0.00	0.01	0.01	0.01	0.20	0.20	0.00
gzip	0.81	0.82	0.01	0.01	0.02	0.02	0.14	0.14	0.00
mcf	0.41	0.43	0.03	0.05	0.05	0.00	0.19	0.19	-0.01
mesa	1.09	1.12	0.03	0.01	0.01	-0.02	0.09	0.09	0.00
mgrid	0.52	0.54	0.04	0.03	0.03	-0.06	0.01	0.01	-0.01

parser	0.75	0.76	0.01	0.02	0.02	0.04	0.19	0.19	-0.02
perlbmk	0.57	0.55	-0.04	0.01	0.01	0.02	0.24	0.24	-0.01
sixtrack	1.04	1.03	-0.01	0.00	0.00	0.00	0.19	0.18	-0.03
swim	0.45	0.46	0.01	0.03	0.03	0.00	0.05	0.05	-0.02
twolf	0.84	0.89	0.06	0.00	0.00	0.00	0.21	0.21	0.00
vortex	0.66	0.64	-0.03	0.00	0.00	0.10	0.18	0.19	0.02
vpr1	0.97	0.97	0.01	0.00	0.00	0.09	0.18	0.19	0.01
vpr2	0.68	0.69	0.01	0.02	0.02	-0.02	0.17	0.19	0.09
wupwise	1.17	1.14	-0.03	0.00	0.00	0.00	0.15	0.15	0.00
avg.	0.75	0.76	0.02	0.02	0.02	0.01	0.14	0.14	0.01

表 4.9 是多轮验证和修改的结果，其中平均 IPC 误差约为 2%，DCACHE 失效率和分支预测失效率误差均为 1%左右。第一轮验证发现了模拟器的几个功能错误，包括一些死锁情况和内存越界访问。第二轮中，我们发现大多数程序 IPC 误差已经小于 10%，但还有几个程序的 IPC 误差很大（20%以上），包括 crafty、twolf、bzip2、gap 等。调查发现其中 crafty、twolf 等程序的分支预测失效率明显高于 ICT-godson，有时甚至达到 2 倍，其原因是 ghr 寄存器更新时机模拟不正确。修复这个问题后，不再有明显失真的分支预测失效率。第三轮中，针对少数误差偏大的程序，我们逐步实现了更多的细节模拟，包括把定点乘、除指令分解为两个内部操作；实现指令 CACHE 的 hit under miss 支持（即在一个取指访问失效访存时，另一个取指访问能够继续在指令 CACHE 命中）；实现更精确的发射策略；实现 Load 猜测例外恢复等等。在此过程中我们也发现和修正了一些模型实现的问题。

尚存的误差源可能包括：

- 被忽略的细节，例如功能部件端口竞争，DCACHE 部分命中情况（一个访问命中正在失效重填中的数据）等。
- 模型和硬件的细微不同。为了保持代码可读性和灵活性，有个别事件发生的时刻可能和硬件有所不同。例如，CACHE 中被替换块可能比硬件早一拍写回。
- 随机因素。例如，功能部件分配算法，CACHE 随机替换等。
- 其它隐藏的实现问题。龙芯 2 号的处理器模型是一个高度复杂的软件，虽然我们已经对它进行了比较彻底的验证，但某些细节的地方仍然有可能存在实现问题。

第五章 RTL 和 FPGA 仿真环境

实际处理器一定拥有一个用硬件描述语言实现的 RTL 模型，通常也会使用 FPGA 进行流片前验证。RTL 模型确定处理器的逻辑行为，FPGA 进一步考察处理器和目标系统的配合情况。RTL 和 FPGA 仿真环境一般主要用于功能验证，而本文则考察了两者在性能分析中的用途。

本章从性能分析的角度介绍龙芯 2 号的 RTL 仿真环境和 FPGA 仿真环境设计。我们讨论了它们和高层性能模型的交叉验证，并探讨了利用它们进行性能分析的可行性。

5.1 龙芯 2 号 RTL 仿真环境

龙芯 2 号的 RTL 模型使用 Verilog 语言实现，使用 Cadence NCVerilog 或者 Synopsys VCS 进行综合和仿真。龙芯 2 号的 RTL 仿真环境支持虚拟的串口、内存控制器等设备，可以进行全系统仿真。

RTL 的实现中可能引入各种问题，需要进行全面的验证。一些实现上的问题可能不会引起功能错误，但会影响最终的性能。例如，即使分支预测器每次都预测错误，处理器也能正常工作，只是效率会受到影响。仅进行功能方面的验证是不够的。

多数性能分析工作使用高层的模拟器(如 ICT-godson 和 Sim-godson)进行。为了防止这些模型出现大的偏差，我们也需要随时验证它们和 RTL 模型的一致性。

5.1.1 RTL 与 ICT-godson 的交叉验证

RTL 模型和 ICT-godson 模型在信号级应该是一致的，由此我们使用两种方法来实现对两者的交叉验证。

第一种方法是通过 RTL 仿真工具提供的高级语言接口（如 IEEE 1364 Programming Language Interface 标准），将 ICT-godson 和 RTL 模型链接在一起进行仿真，cycle-by-cycle 地比较两个模型的信号。这样，任何功能上和性能上的偏差都能够很快地被发现。

另一种方法是在 ICT-godson 中产生测试向量供 RTL 模型仿真使用。这种方法可以更有针对性，速度也比较快。

5.1.2 RTL 与 Sim-godson 的交叉验证

Sim-godson 模拟器并没有实现所有的处理器细节，不能和 RTL 模型直接进

行信号级比较。通常我们可以使用经过验证的 ICT-godson 模拟器来验证 Sim-godson 模拟器，但有时我们也使用 RTL 模型来验证 Sim-godson。因为 ICT-godson 和 RTL 的验证需要比较长的时间，有些情况下 ICT-godson 会滞后。

RTL 与 Sim-godson 的交叉验证主要通过一些小型的测试程序来完成。这些程序包括第四章所使用的微基准程序，也包括一些常见的核心循环，如 DAXPY。通过比较这些程序的性能，我们可以快速判断两者有无大的差别。

5.2 龙芯 2 号 FPGA 仿真环境

FPGA 仿真中，我们使用 FPGA 实现处理器的逻辑，并把它连接到目标系统的主板上运行。FPGA 仿真环境的速度快，配套芯片组和板卡都是真实的，因此对功能验证具有不可替代的作用。同时我们发现，利用 FPGA 仿真环境来做性能分析不仅可行，在一些场合下还有准确性和速度上的优势。

龙芯 2 号处理器使用了三片 FPGA（型号为 Altera Stratix 1S80）进行 FPGA 仿真。代码使用 Quartus 3.0 进行综合，处理器核心频率能达到 3MHz 以上（单片 FPGA 逻辑运行频率可达 20MHz，但由于片间连接引脚不足，采用 20 倍频方式复用，限制了整体频率）。处理器接口逻辑一般运行在 33－66MHz 的频率下，通过分频处理来与处理器核心连接。由于龙芯 2 号的 FPGA 系统运行速度相对比较快，它能够运行真实操作系统和大量的应用。

5.2.1 FPGA 用于性能分析的可行性研究

FPGA 系统和实际目标系统的主要不同之处是处理器的频率。实际系统中一般处理器频率比接口频率高，而 FPGA 系统中相反。因此，如果一个访存操作需要 n 个总线周期完成，在 FPGA 上处理器核心会认为它是 n/X（X=接口频率/核心频率，目前约 20）个处理器周期，实际芯片则认为是 n*Y（Y=核心频率/接口频率，目前是 1-5）个处理器周期。两者有巨大的差别，导致性能数据根本不能类比。

我们可以对处理器接口做一些特殊处理来解决这个问题。一种方法是在处理器中设置一个可编程的配置寄存器，用来控制各种访问返回处理器核心的时机。每个访问回到处理器端时，根据其类型延迟指定的拍数后再送入核心。这样在处理器核心看来，每个访问的延迟就变得比较真实。

这种方法允许我们根据需要设置不同的延迟，因此可以用来评估不同的延迟对系统性能的影响。不过实际处理器中，存储访问的延迟会有一定的变化，把延迟设置为固定值会带来一些误差。但是，如果没有多个并行的访问，内存控制器也没有使用 Open page 等优化的话，延迟的变化并不大。

我们使用实验的方法测试这种方法的准确性。龙芯 2 号中主要影响性能的访

问延迟包括内存读、内存写以及二级 CACHE 读（它的二级 CACHE 为写穿的片外二级 CACHE，不需要考虑二级 CACHE 写延迟）。我们用龙芯 2 号的实际机器和相同 RTL 的 FPGA 系统进行比较。两者接口频率相同，实际机器接口为三倍频。用逻辑分析仪测得访存读操作的延迟约为 15 个总线周期，即大约 45 个处理器周期；而二级 CACHE 读延迟为 6 个总线周期，即 18 个处理器周期；写操作延迟依赖于北桥内部逻辑，通过实验来确定。

表 5.1 和表 5.2 显示，设置合适的延迟后，FPGA 测得的延迟、带宽表现可以和实际机器非常一致。其中延迟测试使用一个简单的程序，它进行连续多个 CACHE 不命中 Load 访问，然后取平均访问延迟。带宽测试使用 Stream[McC99]。其中第一列的数字分别为访存写延迟、二级 CACHE 读延迟、访存读延迟，单位是处理器周期。

表 5.1 FPGA 和实际机器延迟测试结果比较

	Read (cycle)	***Write (cycle)***
实际机器	26.00	37.88
<35,18,46>	25.97	37.98
<32,18,45>	25.65	37.77

表 5.2 FPGA 和实际机器带宽测试结果比较

	Copy (MB/s)	***Scale (MB/s)***	***Add (MB/s)***	***Triad (MB/s)***
实际机器	71.88	70.00	75.12	76.09
<35,18,46>	71.7	70.3	76.2	77.2
<32,18,45>	73.46	72.02	77.95	79.19

表 5.3 进一步显示了 SPEC CPU2000 的运行时间误差。我们可以看到，所有程序的误差都小于 10%，而且平均误差很小。

表 5.3 FPGA 和实际机器 SPEC CPU2000（test）运行时间比较

	Real（s）	***Fpga（s）***	***Error***
gzip	23.9	23.4	-0.021
vpr	20	19.9	-0.005
gcc	25.1	24.6	-0.020
mcf	3.91	3.79	-0.032
crafty	73.1	69.3	-0.055
parser	37.3	35.9	-0.039
perl	19.3	19	-0.016
gap	11.4	11.3	-0.009
vortex	134	144.6	0.073
bzip2	58.9	60.9	0.033
twolf	1.92	2.1	0.085

Fix. Avg.			-0.000
wupwise	66.1	63	-0.049
swim	11	10.6	-0.038
mgrid	213	229	0.070
applu	4.28	4.06	-0.054
mesa	15.7	16	0.019
art	96.7	91.3	-0.059
equake	12.3	12.1	-0.017
ammp	130	127	-0.024
sixtrack	96.5	94.3	-0.023
Apsi	82.4	78.9	-0.044
Fp.avg.			-0.02

设定固定延迟的方法可以用于评估不同延迟对系统性能的影响，但它有一定的局限性。当处理器和北桥采用更多的优化措施后（例如支持多个并发的存储访问），这种方法会难以准确反映实际情况。

另一种方法可以很精确地模拟实际系统的延迟。由于龙芯 2 号 FPGA 的接口频率是真实的，我们可以在处理器接口中使用一些计数器来检测实际的访问延迟（以总线周期为单位），然后对核心延迟相应的周期数。这样 FPGA 系统可以精确地反映实际系统性能，只有不由处理器控制的访问（例如，硬件 DMA 操作）例外。

5.2.2 FPGA 性能分析的用途

FPGA 仿真环境速度快，经过一定处理后真实性也很强，因此它有许多潜在的利用空间。我们可以用 FPGA 来运行更多更大规模的程序；我们也可以用它来评估一些与操作系统或者 I/O 性能关系密切的基准程序性能。此外，FPGA 也可以用于研究硅后优化技术，例如硬件性能计数器设置以及其它硬件性能分析支持方案。

另一个非常重要的用途是利用 FPGA 来验证全系统模拟。高性能微处理器的性能不仅仅依赖于处理器核心，它和处理器接口、配套芯片组、内存以及系统软件等都有密切关系。高层的全系统模拟器（如 Sim-godson）必须正确地建立这些组件模型才有可能准确地预测最终的系统性能。而这些组件本身也常常是一个很复杂的系统，对它们建模同样需要一个验证过程。利用 FPGA 的真实性和可编程性，可以有效地验证全系统模拟。

第六章 性能分析方法

一个完善的性能分析环境能帮助设计者有效地测量和观察目标系统的行为，以及快速地实现和比较不同的方案。测量和观察能够暴露一些性能瓶颈，但对于复杂的高性能微处理器，仅靠经验性的测量和观察不足以形成对系统的全面理解。我们需要一些系统的方法来有效地理解系统的行为。

高性能微处理器内部可以看作一个并行系统，其中存在各种对输出（即性能）有影响的瓶颈因素，性能分析就是理解这些瓶颈因素和它们之间的互相作用对性能影响的过程。性能分析的困难之处在于这些影响常常不能很直观地理解。例如，有时我们注意到某个队列满的概率比较大，但是简单地增加它的大小可能没有任何效果，个别情况下甚至会降低性能。

瓶颈分析方法可以帮助我们确定系统中各瓶颈的重要程度和它们之间的关系。本章介绍并比较了三种瓶颈分析方法。第七章中我们将给出这些方法的应用例子。

6.1 参数敏感性分析

参数敏感性分析方法每次改变处理器的一个或者多个参数，然后测量它们对性能的影响。这种方法常被用来认识微处理器各个组成部件对整体的影响。理论上，如果可以测量所有参数组合，我们就能量化每个参数以及每组参数之间的相互作用对性能的影响程度。但是，假设有N个参数，每个参数有a个值，这种方法需要a^N次模拟，随着a和N的增加以指数速度增加。

实际研究一般只能对一些参数子集进行敏感性分析。例如，Skadron 等人[SAM99]对指令窗口大小、分支预测精度、一级 CACHE 大小等参数做了大量的敏感性分析。在分析其中一些参数的敏感性时，他们把另外一些设为固定值。例如，为了分析一级 CACHE 对 IPC 的影响，他们把分支预测的精度固定为 100%，指令窗口的大小固定为 128。

参数敏感性分析简单易行，但是它不足以解决瓶颈分析的需求。首先，处理器的设计空间非常大，分析所有的参数组合是不现实的。如果每次只研究一个参数，那么参数之间的相互作用将无法体现。如果选择性地研究其中一些参数，那么由于其它设计参数的重要性是未知的，它们可能对研究的结果有不可预知的影响。而且，在被选择的参数里，如果为了简化工作而将某些参数固定用于研究另一些参数（如上述例子），那么被固定参数的值对研究结论的影响也是未知的。不考虑这些问题，使用参数敏感性分析可以导致不准确甚至误导的结论。尽管如此，如果结合设计者的经验或者其它方法使用的话，参数敏感性分析仍可能是一个有效的分析方法，特别是在处理器设计早期。

6.2 PB 设计

Yi 等人[YLH03]提出了一种基于统计的性能分析方法。他们使用 Plackett Burman 设计[PlB46]（以下简称 PB 设计）来进行处理器参数分析、基准程序选择以及优化方案分析。

PB 设计是一种用来确定可变参数对输出的影响的统计方法（统计学上称为部分多因素试验）。对于有 N 个可变参数（每个可以取两个值）的系统，PB 设计可以通过 X=（N/4 + 1） * 4 次试验来得出结果。一个改进的 PB 设计可以通过 2X 次试验来给出任意单个参数和选定的参数组合对输出的影响。PB 设计不能给出由三个或者三个以上的参数交互作用的效果，不过 Yi 证明了，体系结构设计中几乎所有影响显著的交互作用都是两个参数的交互。而且，如果一个交互作用的影响比较显著，那么参与交互的参数本身也都具有显著的影响。因此，PB 设计可以可靠地测量处理器参数对性能的影响。

使用 PB 设计测量处理器参数对性能的影响时，所有的参数取两个值：高、低。高值略微高于该参数正常取值范围，低值则略低于正常取值范围。高、低值不一定是数字，例如，分支预测器的高值可以是理想预测器，低值可以是 bimod 预测器。值的选择应该刚好在正常取值范围外，取的范围过大会人为加大该参数的影响，过小则可能使得影响很小。对于每一次试验（模拟），都有一半的参数取低值，另一半取高值。每个参数的取值由 PB 矩阵确定，该矩阵每一行都由 1 和－1 组成。第 n 次试验的第 m 个参数根据矩阵第 n 行 m 列确定取值，对应 1 的则取高，－1 则取低。

每一次试验都会产生一个输出，在处理器性能分析中，这个输出可以是程序执行时间。做完所有 2X 个试验后，对第 m 个参数，PB 矩阵的第 m 列和输出矢量的点积结果就是它对输出的影响（不计符号，取其绝对值）。如图 6.1，其中参数 A 的影响计算如下：

$Effect_A = 1 * 9 + （-1） * 11 + … + 1 * 112 = 191$

可以看到，其中影响最大的参数依次是 G、A、C。

对不同程序，计算出来的数值差别可能很大。为了便于互相比较，我们可以把影响按大小排序得出一个排序矢量。把所有基准程序得出的排序矢量加起来，我们就能够知道每个参数对最终性能重要程度。有了这个信息，再针对性地进行敏感性分析，能够大大提高结论的可靠性。此外，利用参数影响程度分析的结果，人们可以针对显著的瓶颈参数提出优化措施，还可以削减一些影响很小的参数的相关硬件，最终得到一个更平衡的设计。

PB 设计的结果可以用来进行优化方案分析。通常人们评价一个优化方案时只给出一个简单的加速比数字，即最终效果。PB 设计可以用来帮助更深入地理解优化方案对处理器的影响。假设一个优化使得处理器速度提高了 25%，同时，处理器里有 A、B 两个参数是主要性能瓶颈。那么，这个优化到底是同时改善了两个瓶颈，还是改善了一个，但加剧了另一个？通过比较优化前后的参数显著性排名，我们可以得到答案。也就是，PB 设计能够指出一个优化方案是如何起作用的，这对理解系统性能和进一步优化很有好处。

A	B	C	D	E	F	G	Execution Time
+1	+1	+1	-1	+1	-1	-1	9
-1	+1	+1	+1	-1	+1	-1	11
-1	-1	+1	+1	+1	-1	+1	2
+1	-1	-1	+1	+1	+1	-1	1
-1	+1	-1	-1	+1	+1	+1	9
+1	-1	+1	-1	-1	+1	+1	74
+1	+1	-1	+1	-1	-1	+1	7
-1	-1	-1	-1	-1	-1	-1	4
-1	-1	-1	+1	-1	+1	+1	17
+1	-1	-1	-1	+1	-1	+1	76
+1	+1	-1	-1	-1	+1	-1	6
-1	+1	+1	-1	-1	-1	+1	31
+1	-1	+1	+1	-1	-1	-1	19
-1	+1	-1	+1	+1	-1	-1	33
-1	-1	+1	-1	+1	+1	-1	6
+1	+1	+1	+1	+1	+1	+1	112
191	19	111	-13	79	55	239	

图 6.1 PB 设计矩阵（X=8）

6.3 交互代价分析

Fields 等人[FBH04]提出使用交互代价（interaction costs, icost）来分析微体系结构瓶颈。

在这种方法里，一个性能瓶颈是一个影响执行时间的事件集合。这里的事件包括前面所说的处理器参数，也可以指一些具体事件，例如特定的一个 CACHE 不命中。一个事件的代价（cost）定义为将该事件理想化（对一个 CACHE 不命中来说，理想化就是把它换为命中；对指令窗口大小来说，则是把指令窗口设为无限大）之后能够获得的加速。设 t 为原执行时间，t（e）为将事件 e 理想化之后的执行时间，则

$$Cost(e) = t - t(e).$$

这个定义可以自然地扩展到一个事件集合的代价。Cost 是一个很有用的值，比如给出一个特定 load 指令所有动态不命中的代价，可以用来决定对它预取是否值得；对于处理器设计者来说，它可以指出优化某个部件有多大的潜力。

只了解单个事件的代价是不够的，因为它没有体现事件之间的交互影响。正如 6.1 节所讨论的，其它事件的变化可能引起这个代价极大的变化。Fields 等人用两个事件（集合）之间的交互代价来量化交互影响，交互代价的定义如下：

icost（{e1,e2}） = cost（{e1,e2}） – cost（e1） – cost（e2）

icost（{}） = 0

icost（U） = cost（U） - Σicost（V）；其中 V 取 U 的所有真子集。

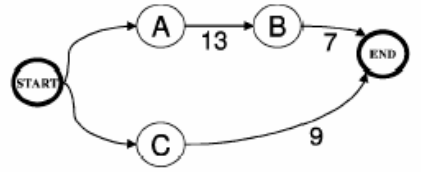
使用交互代价可以更清楚地理解并行事件之间的关系。例如，我们可以把两个事件（集合）之间的交互关系分为三类：

icost（{e1,e2}） = 0 → e1 和 e2 互相独立，可以单独优化

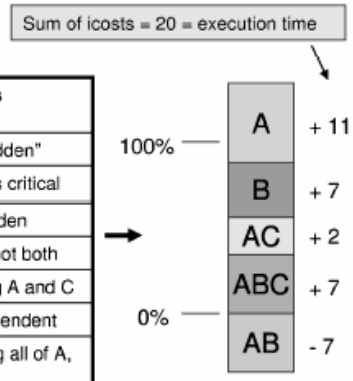
icost（{e1,e2}） > 0 → e1 和 e2 并行相关，必须同时优化两者才能优化性能

icost（{e1,e2}） < 0 → e1 和 e2 串行相关，单独优化其中一个可能起效，但两个一起优化不会比优化一个更好。

交互代价也可以用于更准确地切分总体执行时间。传统上的执行时间切分方法试图把一些时间分配给特定的一个原因，而事实上常常有几个原因同时起作用。交互代价定义有一个自然的推论：所有事件的代价（即总执行时间）等于所有事件之间的交互代价总和。这样所有交互代价就组成了总执行时间的一个切分。Fields 等人用图 6.2 来说明交互代价的计算和执行时间切分：



Cat. Name	Expression for computing icost	Interaction Cost (icost)	Comments
A	cost(A)	11	A is partially "hidden"
B	cost(B)	7	All of B's latency is critical
C	cost(C)	0	C is totally hidden
AB	cost({A,B}) - icost(A) - icost(B)	11 -11 - 7 = -7	Optimize A or B, not both
AC	cost({A,C}) - icost(A) - icost(C)	13 - 11 - 0 = 2	Bonus for optimizing A and C
BC	cost({B,C}) - icost(B) - icost(C)	7 - 7 - 0 = 0	B and C are independent
ABC	cost({A,B,C}) - icost({A,B}) - icost({A,C}) - icost({B,C}) - icost(A) - icost(B) - icost(C)	20 - (-7) - 2 - 0 - 11 - 7 - 0 = 7	Bonus for optimizing all of A, B, and C



(a) Breakdown example　　(b) One possible visualization

图 6.2 交互代价的计算和执行时间切分

要获得所有的交互代价，如果用模拟的方法需要 2^n次模拟。这和完全的参数敏感性分析类似，也是不现实的。PB设计方法能够把需要模拟的次数降低到O（n），但它不能给出事件代价的量化结果。Fields等人采用一种依赖图模型[FRB01]来解决这个问题。该模型用带延迟的边来对程序行为和处理器各种限定条件（控制相关、数据相关、指令窗口限制等）建模。模型建立之后，对某个事件的理想化简化为在依赖图中改变相应边的延迟或者删除它。一个事件（集合）的代价通过计算基本配置和理想化后依赖图关键路径的长度得到。而交互代价可以由事件代价计算得出。这样，我们不用实际对某个事件实现理想化的模拟器并进行模拟就能得到它的效果。

Fields 等人使用两种方法来建立依赖图模型，并把交互代价计算结果和进行多次模拟

的实际结果进行了比较。第一种方法是通过模拟器运行程序来建立模型。另一种是在处理器中增加硬件性能监控器，然后使用一种被称为 shotgun profiling[FBH04]的软件算法来生成依赖图模型。两种方法的平均误差分别为 9%和 11%，这说明依赖图模型具有合理的精度。

解决了交互代价计算的问题后，我们可以用它来进行瓶颈分析。图 6.3 是 Fileds 等人举的一个优化长流水线的例子。

Table III. Breakdowns for Optimizing a Long Pipeline

Category	bzip	crafty	eon	gap	gcc	gzip	mcf	parser	perl	twolf	vortex	vpr
dl1	**22.3**	**24.5**	**17.5**	**13.6**	**18.6**	**31.9**	**7.4**	**19.1**	**31.6**	**19.1**	**28.5**	**19.8**
win	14.6	16.3	16.5	34.7	12.6	4.5	4.6	15.9	5.8	22.3	39.4	21.2
bw	4.0	6.0	6.1	2.2	5.6	5.3	0.5	2.7	7.2	3.4	4.8	4.2
bmisp	37.8	26.4	14.3	11.8	24.6	24.0	25.3	15.5	36.0	22.6	1.6	23.3
dmiss	21.8	6.7	0.7	21.8	25.1	8.1	78.8	30.4	1.4	32.5	19.7	31.1
shalu	9.8	11.3	4.9	12.5	5.3	20.6	1.4	17.3	7.3	7.5	5.4	7.8
lgalu	0.3	0.8	11.5	5.8	0.3	0.5	0.0	0.1	0.8	4.0	1.5	3.8
imiss	0.0	0.6	9.0	1.2	2.1	0.1	0.0	0.1	5.1	0.0	3.3	0.0
dl1 + win	**−5.2**	**−11.9**	**−7.9**	**−6.9**	**−3.9**	**−10.5**	**−0.1**	**−6.6**	**−5.6**	**−4.2**	**−25.9**	**−6.7**
dl1 + bw	5.9	10.1	7.3	3.0	11.4	5.6	0.3	4.6	9.2	1.5	16.8	2.2
dl1 + bmisp	**−9.1**	**−4.6**	**−3.7**	**−2.8**	**−6.3**	**−3.4**	**−2.3**	**−2.4**	**−7.3**	**−5.7**	**−0.2**	**−4.5**
dl1 + dmiss	−0.8	−1.2	−0.4	−0.5	−1.4	−1.0	−0.7	−1.8	−0.2	−2.3	−1.8	−2.4
dl1 + shalu	**−4.3**	**−4.6**	**−0.8**	**−2.1**	**−1.8**	**−12.7**	**−0.1**	**−4.8**	**−1.8**	**−0.4**	**−4.7**	**−2.0**
dl1 + lgalu	−0.3	0.2	−1.1	−0.6	−0.3	−0.5	0.0	−0.1	−0.7	−0.0	−1.3	−0.7
dl1 + imiss	0.0	0.0	1.1	0.3	0.3	0.0	0.0	0.0	1.1	0.0	0.6	0.0
Other	3.2	19.4	25.0	6.0	7.8	17.5	−15.1	10.0	10.1	−0.3	12.3	2.9
Total	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**

(a) CPI contribution breakdown (%) with four-cycle level-one cache

图 6.3 利用 IPC 切分图来优化长流水线

假设物理设计人员已经对一级数据 CACHE（dl1）做了尽可能的优化，但再也无法使得它的访问延迟小于 4 拍了。现在的问题是，如何优化微体系结构来掩盖这个高延迟？上表的数据首先表明，dl1 访问的代价很大，平均占总执行时间的 15－25％。此外，指令窗口大小（win）、分支预测（bmisp）和 ALU 延迟有明显的串行相关，因此改善这些因素可能有助于隐藏 dl1 访问延迟。

6.4 各种方法的比较和评价

现代的高性能微处理器是一个复杂的并行系统，对它进行性能分析需要理解各种事件之间的交互关系。参数敏感性分析可以用于分析单个事件的影响和事件交互的影响，借助一些统计方法也可以量化这些影响。由于直接进行完全的参数敏感性分析需要过多的时间，通常人们只能经验性地选择一些子集分析，这影响了结论的全面性和可靠性。PB 设计解决了参数敏感性分析中模拟次数的问题，可以通过约 2N 次模拟来确定 N 个参数的影响程度。但它没有直接给出事件交互影响的量化数据。而且，PB 设计要求每个参数只取两个值，它能给出的信息量比完全的参数敏感性分析少。此外，PB 设计中没有提供对高、低值选择的系统方法，不同的选择会影响最终结论。交互代价分析法一方面是

一种解释数据的方法，把事件相关归类为独立、并行相关和串行相关可以精炼地表达它们之间的关系，而且这种分类可以直接用于瓶颈分析和优化。另一方面，交互代价分析方法提出了使用依赖图生成交互代价的方法，可以解决模拟次数的问题。依赖图模型可以通过模拟器生成，也可以由实际处理器生成。不过由于代价计算只考虑事件理想化的情况，交互代价分析方法能得到的信息量也比完全的参数敏感性分析少。而且，对复杂处理器来说依赖图模型的精度还有待考察。

第七章 龙芯 2 号性能分析和优化

本章介绍龙芯 2 号处理器的性能分析和优化工作。我们按照性能分析工作的任务类型来组织本章内容。对每一项任务，我们都给出一些案例，以展示前面所讨论的性能分析环境和分析方法的实际应用情况。

微处理器的性能是处理器和目标工作负载相互作用的结果，因此性能分析的任务既包含处理器结构分析，也包含工作负载分析。我们把处理器结构相关的任务分为三项：处理器行为的测量和表示、瓶颈分析、提出和评估优化方案。这些都是实际处理器设计过程中常见的任务。工作负载分析方面我们也从三个方面进行讨论：工作负载特性分析、编译器影响、操作系统影响。

7.1 处理器行为的测量和表示

处理器行为的测量和表示是性能分析的基础工作。但是对高性能微处理器来说，这并不是一件很容易的事情。高性能微处理器内部结构非常复杂，大量内部事件都能够对最终的运行时间（或者功耗等其它指标）产生影响，而这些影响和最终结果常常没有很直观的联系。

在实际运行的处理器上，内部的情况只能由处理器中一些专门的性能分析硬件（如硬件性能计数器）提供。由于硬件资源代价较大，通常我们只能获得少量的数据。设计能够有效支持性能分析的硬件是一个重要的研究课题[ZLS96, DHW97, FBH04]。

对处理器设计者来说，实际硬件还未必存在，因此他们通常只能使用某种处理器模型和基于模拟的方法来获得数据。模拟的准确性和速度对测量都很重要。龙芯 2 号的性能分析环境中 ICT-godson 和 Sim-godson 模拟器都能全面地测量处理器行为，前者更为详尽和准确，后者则更快，可以充分适应不同场合的要求。

处理器行为测量的目的是为了更好地理解系统行为，为系统瓶颈分析和优化提供数据支持。为了避免淹没在海量的数据里，我们需要精心选择待测量的数据及其表示方法。第三章中我们已经提到了 ICT-godson 模拟器的多角度数据收集支持，Sim-godson 模拟器也具有同样的功能。本节中，我们以龙芯 2B 处理器为例，给出各种数据的例子和相关分析。所运行的程序为 SPEC CPU2000 和两个典型的嵌入式基准程序 whetstone 和 drystone；SPEC CPU2000 使用 test 输入集，每个程序最多运行 10 亿拍。

龙芯 2B 的配置如表 7.1 所示。

表 7.1 龙芯 2B 微体系结构配置

Fetch:decode:issue:commit 宽度	4:4:5:4:4	主流水级	7 级
功能部件	2 定点，1 个访存，2 浮点	功能部件 load-use 延迟	定点 ALU 1，乘 4 浮点加 3，乘 4
Roqueue	32	转移队列	8
定点发射队列	16	存储访问队列	16
浮点发射队列	16	失效访问队列	2
分支预测器	Gshare: 9 位 ghr，4096 项 pht. 无 BTB		
指令 TLB	8 项，全相联	数据 TLB	64 项，全相联
一级指令 CACHE	32K，2 路组相联	一级数据 CACHE	32K，2 路组相联
访存延迟	50 拍	L2	无

7.1.1 事件计数统计

处理器运行某个程序时，内部会发生各种事件，可以用一些简单的计数器来统计各种事件发生的次数。通过这些计数器，我们能得出一些重要的总体性能参数，包括平均每拍运行的指令数 IPC（或者平均每条指令花费的拍数 CPI），分支预测失效率（或命中率），各级 CACHE 访问失效率（命中率），TLB 访问失效率（命中率）等等。如果把这些计数器用硬件实现在处理器内部，就成了硬件性能计数器，它是一种常用的硅后性能分析工具。

事件计数统计能够提供对处理器及其工作负载的整体认识。例如，从图 7.1 可以看到，art、apsi 等程序的 IPC 很低；从图 7.2 可以看到，定点程序的分支预测失效率比浮点程序大。把各种参数和它们的预期值比较，可以判断目标系统是否达到了设计要求。例如从图 7.1 和 7.2 可知，龙芯 2B 的平均 IPC 约 0.6，定点程序的分支预测失效率平均约 27%，浮点平均约 16%，这些性能是低于预期的。这和早期性能分析流程不完善有关。当然，这个版本主要目的是验证物理设计，并没有针对性能优化。

事件计数统计孤立地看待各种事件，因此它往往只能给人整体的印象，不能确切地指出性能瓶颈和优化方向。

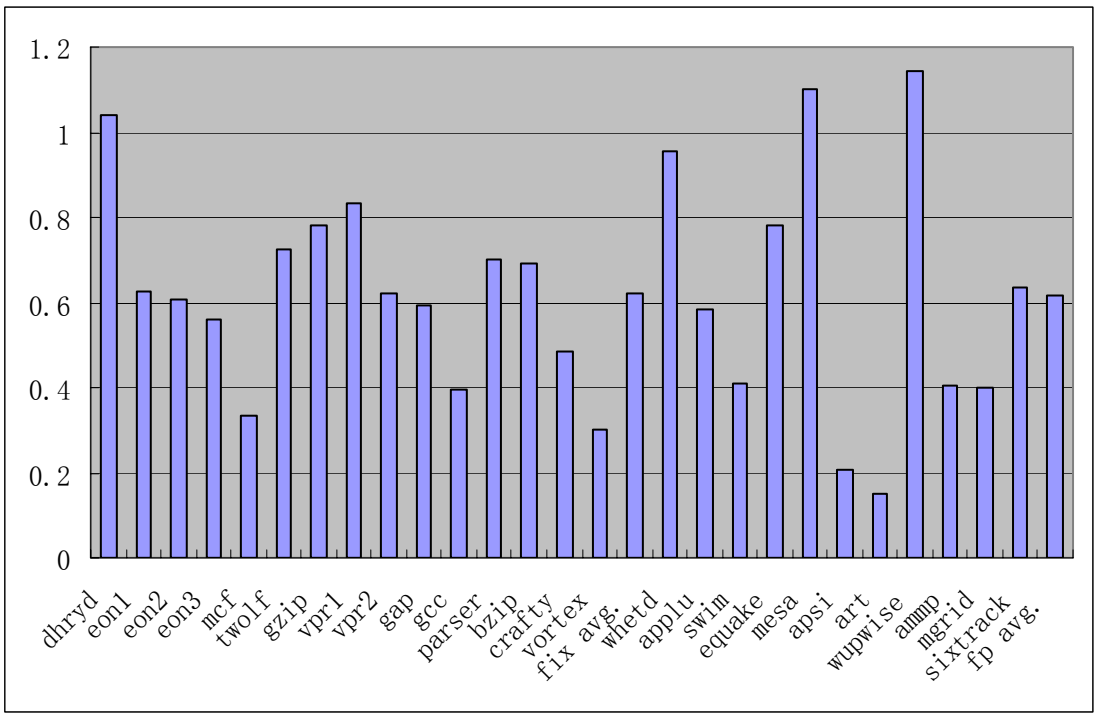


图 7.1 龙芯 2B 的 IPC

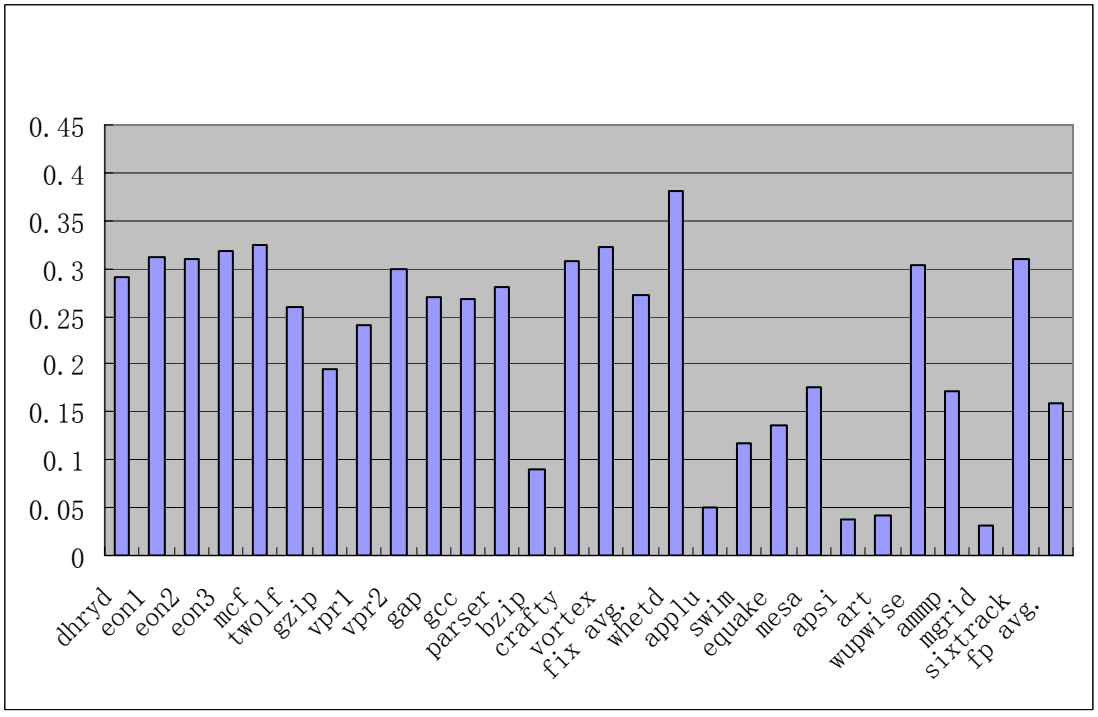


图 7.2 龙芯 2B 的分支预测失效率

7.1.2 流水级吞吐率分析

流水级吞吐率分析方法按处理器内部各流水级把一些事件关联起来，以更清晰地反映事件之间的关系。下面我们以龙芯 2B 的 IPC 分布为例来说明这种数据的表示和作用。

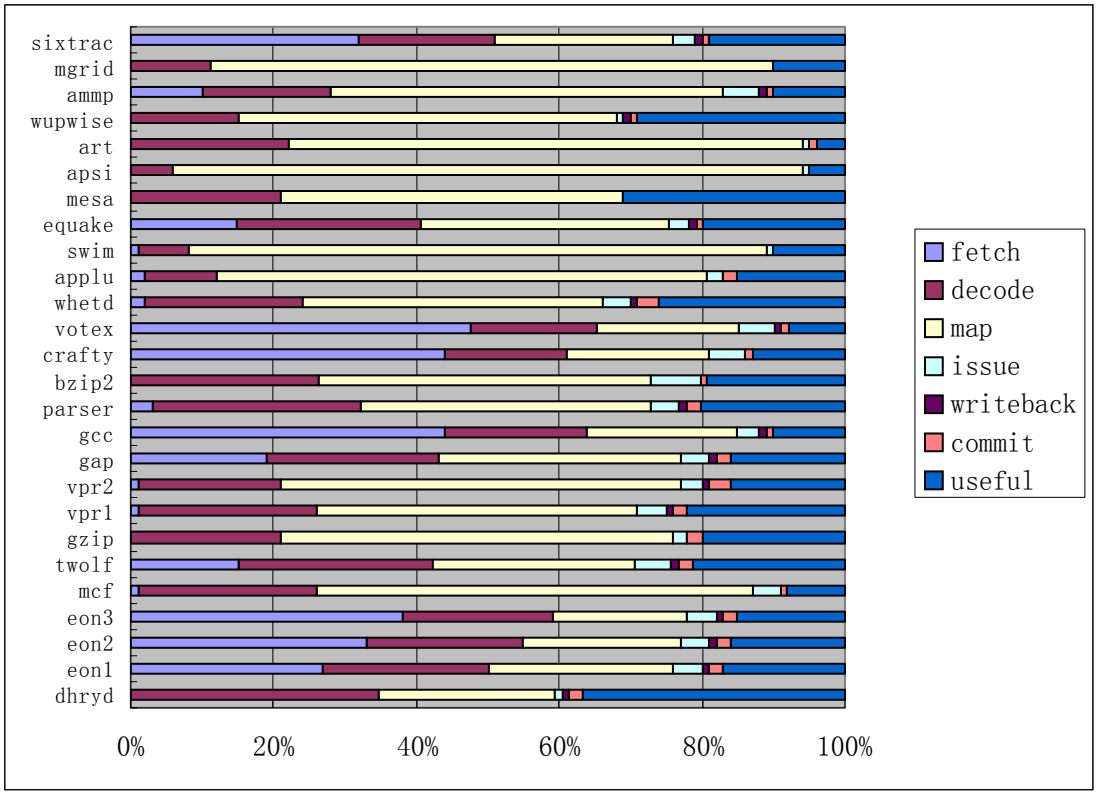


图 7.3 各流水级 IPC 分布图

龙芯 2B 是一个 4 发射的超标量处理器，理想的 IPC 应该为 4，但最后在 commit 流水级平均只有 0.6，那么性能损失在哪里呢？图 7.3 中，我们把理想的 IPC 4.0 在处理器各流水级的损失分布表示出来，其中 fetch、decode、map、issue、writeback、commit 分别表示取指级、译码级、寄存器重命名级、发射级、写回级和提交级的损失，useful 为最终有效的 IPC 输出。一个流水级的损失是指上个流水级输出的 IPC 和它的 IPC 输出的差。取指级的损失都是由指令 CACHE 不命中引起。译码级的损失由指令 TLB 不命中、指令 CACHE 不命中、转移指令猜测错误和 CACHE 行边界等引起。CPU 内部多个队列和重命名寄存器资源不足引起的阻塞都体现在寄存器重命名级。发射级的损失由指令相关或者功能部件不足引起。而写回和提交级的损失主要来源于猜测错误和例外取消。

我们可以清楚地看到，sixtrack、vortex、 crafty、 eon 等程序的指令 CACHE 失效比较多，导致它们取指级有效输出平均每拍只有不到 3 条指令；而多数程序在寄存器重命名一级损失最严重，这暗示处理器核心对这些程序的指令级并行性

开发不是很成功。

为了进一步了解每个流水级 IPC 损失的原因，我们进一步给出导致某个流水级无法生成有效输出的原因分布，如图 7.4－7.6。由于处理器内部的并行性，有时会同时存在几个导致不能生成有效输出的原因。为了简洁起见，对这种情况我们按固定次序取第一个原因（为所有的原因组合生成统计可以更准确地体现瓶颈情况，但是这样数据量比较大，难以直观表示）。

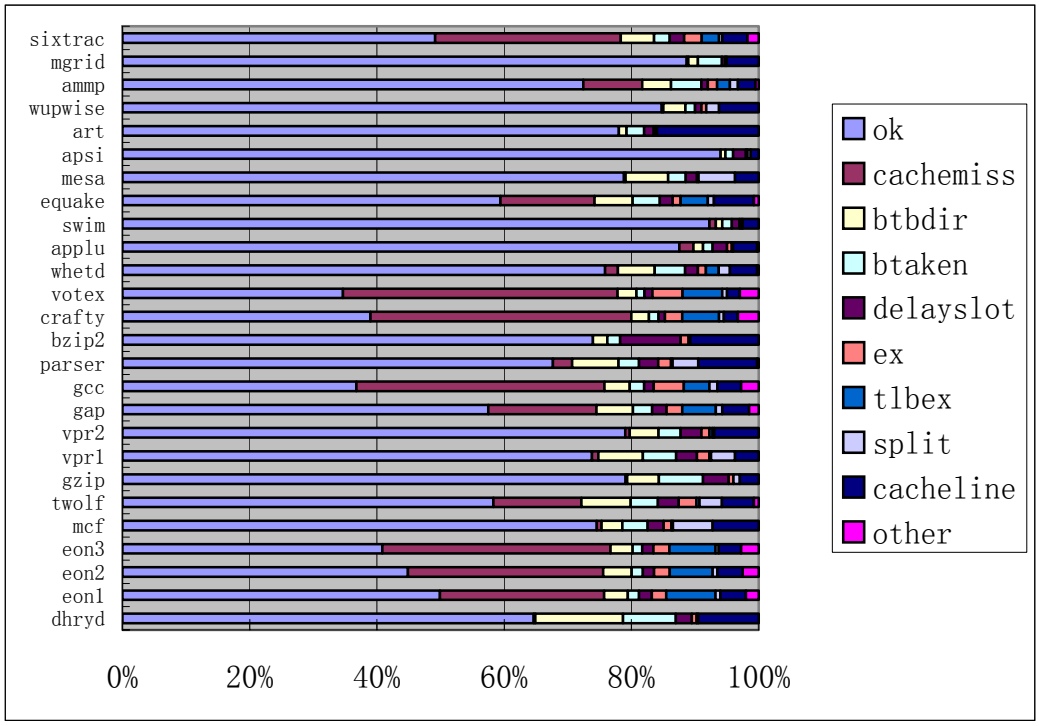


图 7.4 decode 级 IPC 分布

图 7.4 中我们把取指级的 IPC 输出作为 100%，即译码级有有效输入的总时间；然后我们统计每拍输出的情况：如果（对 4 条输出总线的每一条）有输出则累加 OK 次数，否则根据导致不能输出的原因累加相应计数器。其中，cachemiss 表示指令 CACHE 不命中导致没有输出的情况（龙芯 2 号在译码阶段检查指令 CACHE 的 TAG，取指级的指令 CACHE 不命中是指取指时发现指令 CACHE 正在等待访存部件返回该块，因此无法生成有效输出；而译码级不命中指 TAG 不命中），btbdir、btaken、delayslot、ex、tlbex、split、cacheline 和 other 则分别表示由于 BTB 猜测的方向不正确、转移指令跳转（引起其后的指令被取消）、例外（包括分支预测取消）、指令 TLB 不命中、指令划分（少数情况下龙芯 2 号不能在一拍内处理完 4 条指令），遇到 CACHE 块边界以及其它一些未归类原因导致没有有效输出。

从图 7.4 中可以看到对 sixtrack、vortex、crafty、gcc、eon 等程序，指令 CACHE 不命中的损失相当大，同时指令 TLB 不命中的损失也不小。这暗示它们是比较大型的程序，事实上我们发现这些程序的源代码都超过了 10000 行。

整体来说分支预测的影响也比较明显，尤其是 BTB 的影响。其它比较大的损失包括由于取指不能越过 CACHE 行的限制引起的空指令槽。这些观察可以用于指导相应的软硬件优化。例如，通过重新安排代码位置，编译器有可能减小取指不能越过 CACHE 块边界的限制所带来的损失。

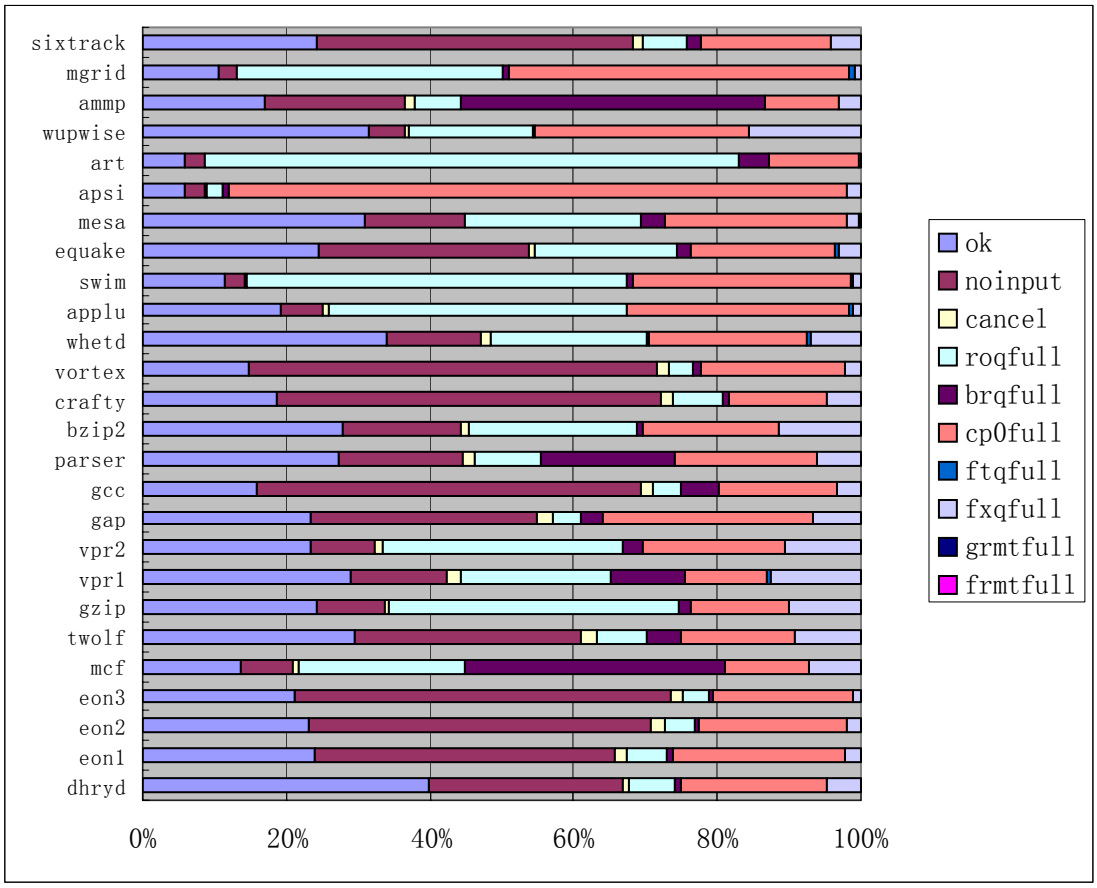


图 7.5 regmap 级 IPC 分布

图 7.5 给出了寄存器重命名级的 IPC 分布。龙芯 2 号在这个流水级检查处理器核心各种队列，任何一个队列满都将阻塞这个流水级。图 7.5 中 roqfull、brqfull、cp0full、ftqfull、fxqfull、grmtfull 和 frmtfull 分别表示由于指令重排序队列 roq、分支操作队列 brq、访存操作队列 cp0、浮点发射队列 ftq、定点发射队列 fxq、定点重命名寄存器、浮点重命名寄存器等资源不足导致阻塞的情况。从图中可以看到，重命名表资源不足的情况基本没有，这是因为龙芯 2 号的重命名寄存器数目和指令窗口相当的原因。资源阻塞比较严重是 cp0 队列和 roq 队列，这反映了 Godson2B 中访存部件效率不够高以及指令窗口比较小的问题。Ammp、parser 和 mcf 三个程序中 brq 满的情况比较突出，暗示这些程序有比较多的小循环。实际测量表明，这三个程序平均每个基本块的指令数（IPB）都小于 6，为所有程序中最小。

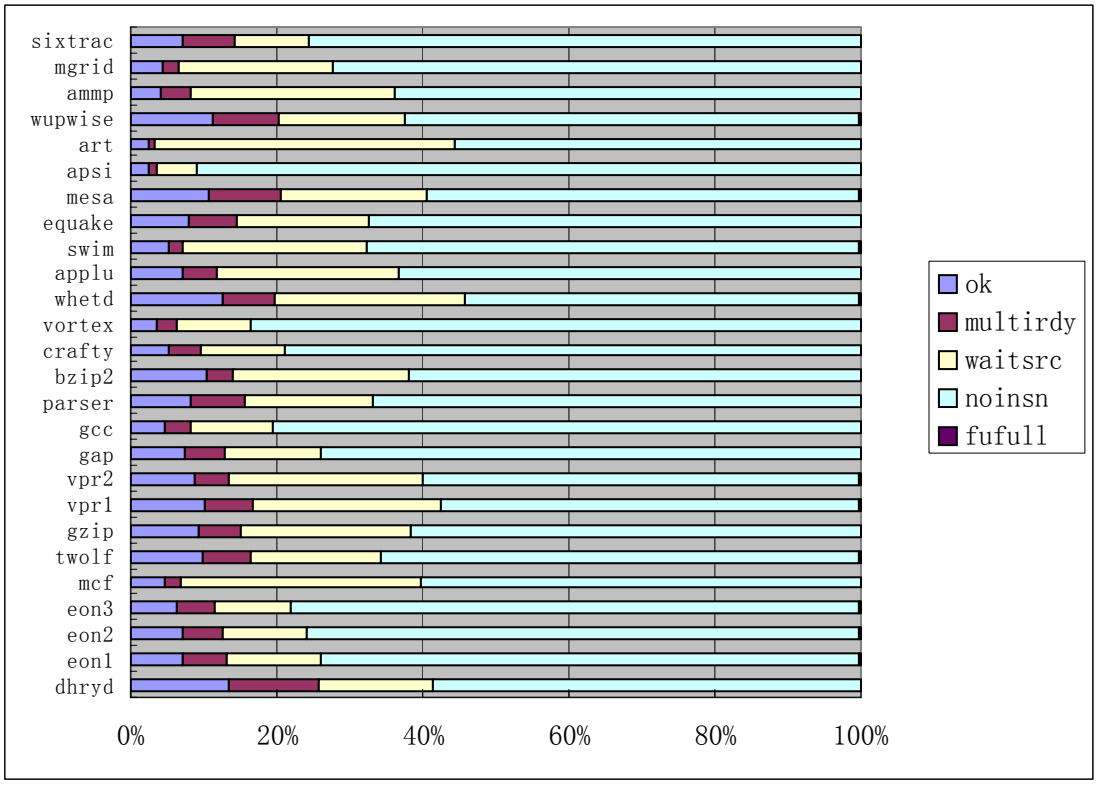


图 7.6 issue 级 IPC 分布

图 7.6 中，100%对应 4.0 IPC；ok 的情况表示对某个功能部件恰好有一条指令准备好可以发射的情况；multirdy 表示有多个准备可以发射到该功能部件的情况；waitsrc 表示存在需要用该功能部件的指令，但是它们的操作数尚未准备好；noinsn 则表示指令队列里没有要用该功能部件的指令，fufull 表示功能部件忙不能接受新的指令。

指令相关是这个阶段主要的损失来源，但是它并不单是这个流水级本身的问题。需要进一步分析程序固有的相关情况才能确定问题所在。此外，有多个操作可以发射的情况也不少，暗示增加功能部件可能有一定用处。

7.1.3 按指令统计

按指令统计是指把一条指令通过处理器流水线时遇到的各种事件记录下来，即按指令把一些事件关联起来。

这种统计非常有利于定位和分析关键代码（运行频率高的代码）。表 7.2 和表 7.3 给出一个按指令统计的例子。

表 7.2 一个关键循环的反汇编代码

```
433e58:    lw      $v0,0($a1)
433e5c:    addiu   $a2,$a2,-1
433e60:    addiu   $a1,$a1,4
433e64:    sw      $v0,0($v1)
```

```
433e68:    bgez          $a2,433e58
433e6c:    addiu    $v1,$v1,4
```

表 7.3 指令相关统计

PC	Execution Times	Fetch/ Dec	Regmap	Issue	Fu	Commit	Branch Miss	DCACHE miss	Avg. IPC For last 64 cycle
433cb0	2238640	3.8	2.1	5.4	1.0	3.8	0	0	1.2
433cb4	2238640	3.8	2.1	3.5	5.0	2.2	0	27761	1.2
433e58	290096	2.3	1.5	2.4	18.5	0.0	0	8609	0.8
433e5c	290096	2.3	1.5	2.4	1.0	17.3	0	0	0.8
433e60	290096	2.6	1.0	2.4	1.0	16.5	0	0	0.8
433e64	290096	2.6	1.0	20.8	5.0	0.0	0	15105	0.8
433e68	290096	2.6	1.0	2.9	1.0	21.9	27181	0	0.8
433e6c	290096	2.6	1.0	3.4	1.0	21.5	0	0	0.8
433ca4	279830	3.0	1.2	2.8	1.0	6.0	0	0	1.1
433ca8	279830	3.0	1.2	2.5	1.0	6.3	0	0	1.1

对表 7.2 所示代码，ICT-godson 模拟器（配置为龙芯 2B）生成的指令相关统计数据如表 7.3 所示（按执行次数排序）。表 7.3 中记录了每条指令被执行的次数、在各流水级停留的平均时间、遇到的分支预测和 CACHE 访问失效次数，以及该指令执行时的平均 IPC。例如，0x433e58 这条指令执行了 290096 次，遇到 8609 次 CACHE miss，它平均每次在功能部件呆 18.5cycle，它所在循环执行的 IPC 大约是 0.8。

这类信息既可以用于分析处理器结构，也可以用于分析目标代码。例如，我们可以看到，表 7.2 所示代码运行的 IPC 低于其它部分代码。而且，有些指令在一些流水级滞留的时间很长。从处理器结构角度深入分析，我们提出了 Load 猜测的优化措施；从编译器角度，利用循环展开和指令调度也能够加速这段代码。具体优化分析参见 7.3.1 节。

通过一些文本处理手段，我们可以把指令统计信息关联到程序源代码中，直接指出程序的关键代码和瓶颈。例如下面是 applu 的例子片断：

```
…
/home/cpu/pub/spec2000-analysis/spec2000_source/CFP2000/173.applu/src/applu.f:565
                    v（ m, i, j, k ） = v（ m, i, j, k ）
  401b50:    d4c40000    ldc1 $f4,0（a2）
CT 1 DC 20 IC 0 BR 0
Times: 1250000 latency: 2.9 1.1 2.2 9.7 22.2 events: 0    76218        53 IPC 0.3 0
…
/* 上面几行是 401b50 这条指令对应的程序源代码、反汇编代码以及指令统计数据。
   第四行为统计排名：
```

CT n 表示本指令所在基本块运行次数排名为 n
DC n 表示本指令遇到的数据 CACHE 失效次数排名为 n
IC n 表示本指令遇到的指令 CACHE 失效次数排名为 n
BR n 表示本指令遇到的分支预测失效次数排名为 n
其中名次按从多到少排序，只取前 100 名，100 名之后 n 为 0。*/

7.1.4 数据分布统计

有时候我们对某些数据的分布情况感兴趣，例如，我们可能想了解，所有的访存请求中，有多少是指令访问，多少是数据访问；数据访问中又有多少是栈访问，多少是堆访问等等。按数据类型、按时间、按空间的数据分布都可能反映出处理器行为的某些特性，为寻找优化方案提供线索。

龙芯 2 号的性能分析中，我们做了大量的数据分布统计。例如，图 7.7 给出指令按它使用的功能部件的分布情况。这种信息有利于我们观察功能部件设计的平衡情况以及指令调度策略。例如，早期龙芯 2 号中的两个定点 ALU（alu1,alu2）执行的指令数常常差别较大。其原因是某些频率较高的指令只能在 alu1 执行，而且指令调度策略不够公平。

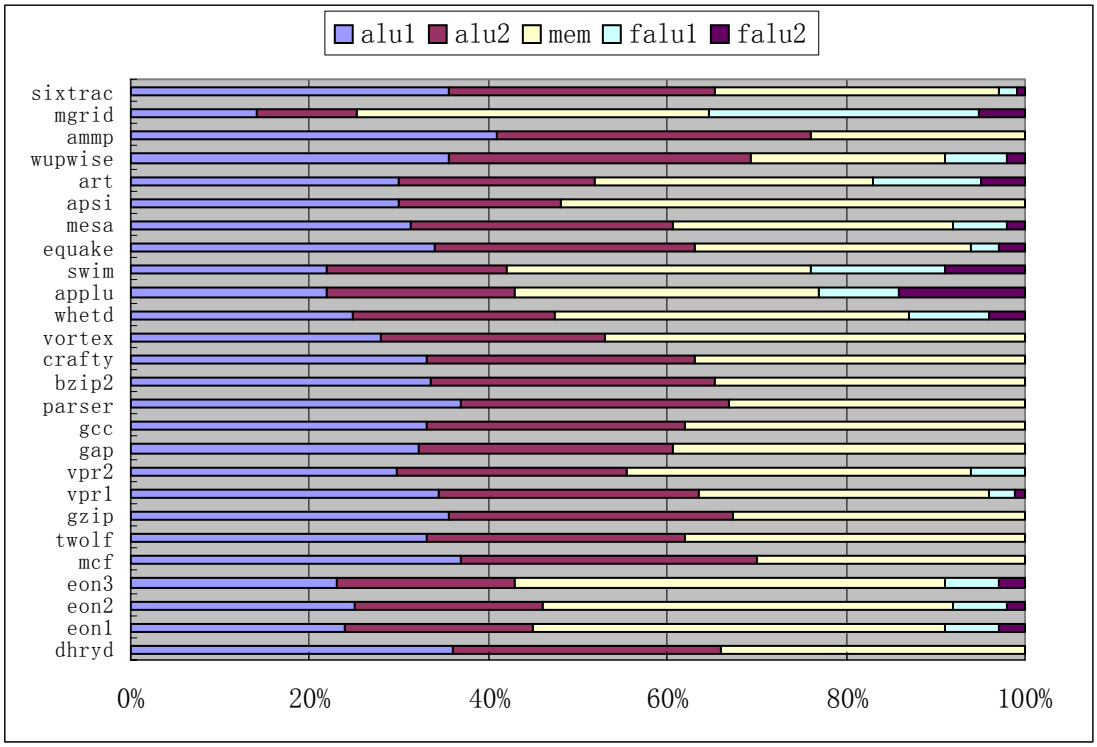


图 7.7 不同功能部件的指令分布

图 7.8 和图 7.9 中，我们分别给出了处理器接口上访存操作的空间、时间局部性分布。空间局部性统计一个时间上相邻的接口访存操作的物理地址间隔的分布概率。时间局部性则统计在一个有限的时间窗口中访存操作具有相同物理地址的概率。图 7.8 中，地址间隔以 CACHE 块为单位，0 表示相邻的两个访存操作

的物理地址对应相同的 CACHE 块。图 7.9 中横轴以接口所见的读操作个数为单位。

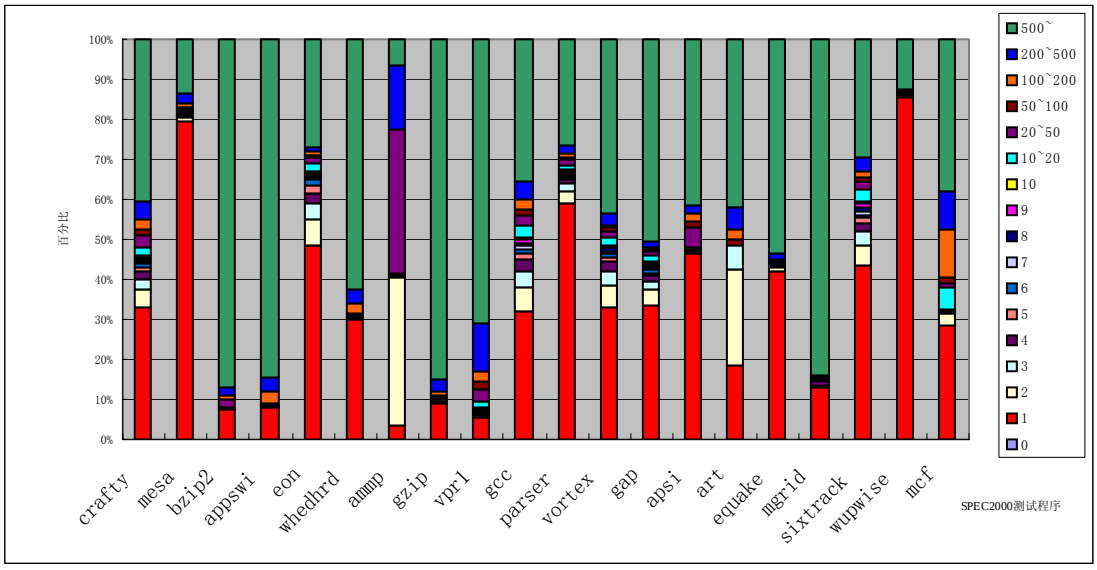


图 7.8 SPEC2000 测试程序所体现的接口访存行为空间局部性统计

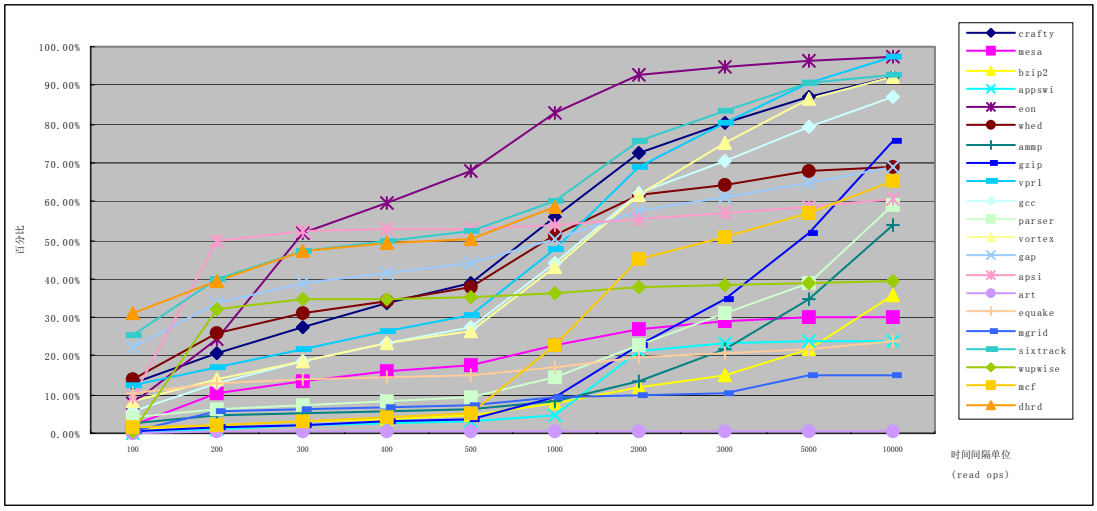


图 7.9 SPEC2000 测试程序所体现的处理器系统总线访存操作时间局部性

大多数测试程序表现出一定的空间局部性，也有一些测试程序的空间局部性比较差一些，比如说 bzip2。以 mesa 为例，时间上相邻的两次访存操作，在地址上相差为一个 CACHE 行的数量占程序整体访存数量的 80%，表现出相当好的空间局部性。另外，对于 mcf，相邻两次访存操作，跨 CACHE 行的占 20%，而且在地址上是累减的。这些信息可以用于指导预取电路设计以及内存控制器优化。

7.2 性能瓶颈分析

第六章中我们讨论了瓶颈分析方法。这里我们给出它们在龙芯 2B 处理器上的实际应用情况。

7.2.1 参数敏感性分析

我们已经知道，对所有处理器参数进行完全的敏感性分析是不现实的，而选择部分参数分析可能不能得出精确的结果。但在设计早期，我们可以经验性地选择一些最重要的参数进行敏感性分析，尽早获得一些定性的认识，以指导后续的设计过程。例如，为了确定龙芯 2B 各个主要部件的性能优化潜力，我们选择了访存、分支预测、队列大小相关的几组参数进行敏感性分析。其中我们使用的配置如表 7.4 所示。

表 7.4 参数敏感性分析所用的各种配置

配置名称	*说明*
Godson2B	龙芯 2B 实际配置，参见表 7.1
Godson2B1	在龙芯 2B 基础上加入 16 项全相联 BTB 和 4 项返回地址栈
Perfi	在 Godson2B1 基础上，使指令 CACHE 访问永远命中
Perfc	在 Godson2B1 基础上，使指令 CACHE/数据 CACHE 永远命中
Perfpred	在 Godson2B 基础上，使分支预测接近 100%正确
Perfall	在 Godson2B 基础上，分支预测和访存都永远命中
Bigq	在 perfall 基础上，ROQ/BRQ/CP0Q/FXQ/FPQ 队列大小增加一倍
Bigq-plus	在 bigq 基础上，访存部件增加到两个，接口可以同时处理两个访问，指令 TLB 增加到 64 项

所运行的程序为 whet stone、dhryd stone 和 SPEC CPU 2000 程序，每个程序最多运行 10 亿拍。图 7.10 和图 7.11 分别显示了各种配置下定点程序和浮点程序的 IPC 情况。

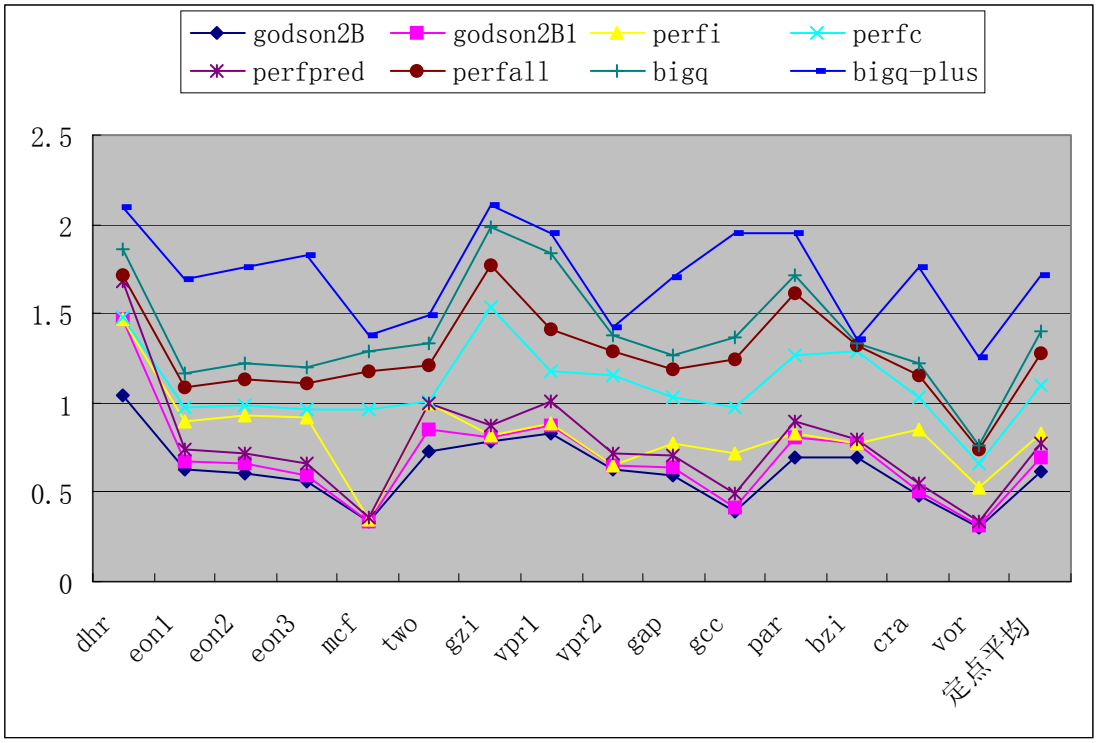


图 7.10 SPEC INT2000 在各种配置下的 IPC

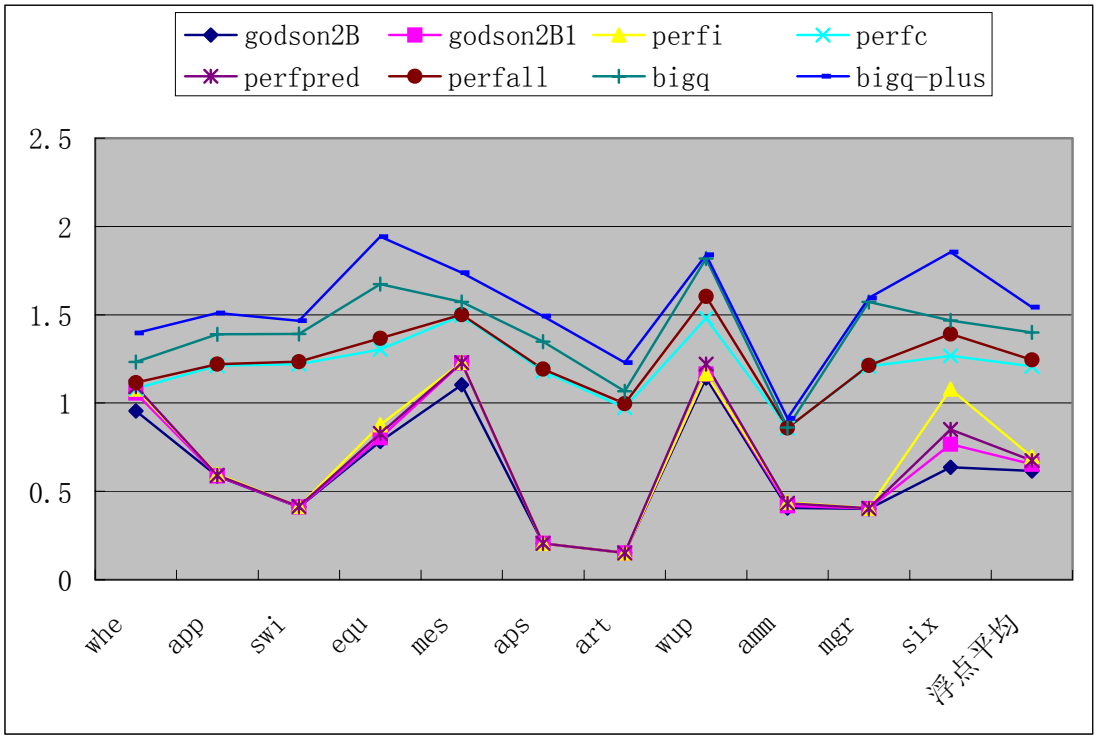


图 7.11 SPEC FP2000 在各种配置下的 IPC

这些数据可以帮助我们分析分支预测、访存（包括指令和数据）和队列大小方面的性能发展潜力。例如，我们可以算出 perfpred 比 godson2B 定点程序平均

性能提高 23.9%，浮点程序平均提高 9.5%。这些数据使我们对分支预测部件的优化潜力有了比较清晰的认识。Godson2B1 添加的 BTB 和 RAS 对性能的提高分别是 11.7%和 6.2%，说明这些优化潜力中很大一部分可以以不大的硬件代价来实际实现。实际上，Godson2B 中分支预测失效比较高的一个重要原因是没有 BTB 和 RAS，图 7.12 显示了这个情况。其中 jrmiss/brmiss 表示 MIPS 指令集中 JR（相对寄存器跳转）类指令发生分支误预测的情况占全部误预测情况的比率。在没有 BTB 和 RAS 的情况下，JR 类指令 100%预测错误。可以看到，JR 预测失效是主要的分支预测失效来源。BTB 和 RAS 能够大幅度减小 JR 类误预测，从而提高性能。

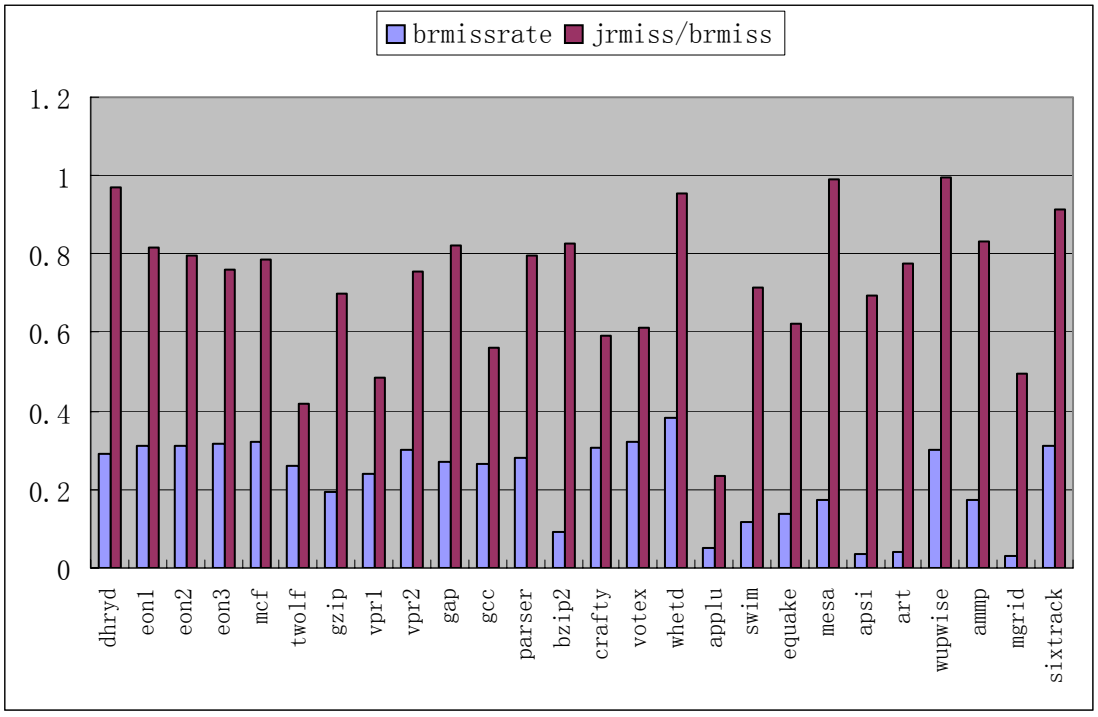


图 7.12 Godson2B 的分支预测失效率和 JR 预测失效的比率

Perfi（理想指令 CACHE）的性能比 Godson2B1 定点程序平均提高 19%，浮点程序平均提高 6%。实际上各程序性能提高很不相同，定点程序中 gcc、crafty 和 vortex 的提高约 70%，twolf 和 gap 约 20%，eon 30-50%，其它程序的效果很不明显。浮点中主要是 sixtrack（40%），equake（9%），其它比较小。指令 CACHE 失效率和程序的大小是直接相关的，上面列出的程序除了 equake 之外源代码都超过一万行。这些数据说明，如果目标工作负载规模比较大，应该考虑指令 CACHE 方面的投资。

现代高性能微处理器中，访存通常是一个主要瓶颈，上面的数据也验证了这一点。考虑 Perfc（理想 CACHE 配置）对 Godson2B1 配置的加速比：只有 dhryd（0.6%）、whetd（2.4%）两个典型嵌入式程序几乎不受影响；加速比较小的有 twolf（18.2%）、vpr1（34.1%）、eon1（%44.8%）、eon2（%49.6）、mesa（21.8%）

和 wupwise（27.1%）；其余的都超过 50%，其中有 10 个超过 100%，apsi 和 art 分别为 475%和 542%。总之，龙芯 2B 的访存系统优化潜力很大。

处理器中各种队列是支持乱序执行、隐藏长延迟操作和挖掘指令级 IPC 的工具，它们的大小和平衡直接影响处理器的性能。Perfall-bigq 比 perfall 定点程序平均性能提高 9.2%，浮点程序平均提高 12.4%。这两个配置中 CACHE 是理想的，已经消除了许多长延迟的存储访问指令。在现实的存储系统配置下，队列的作用会更加突出。

7.2.2 交互代价分析

交互代价分析方法可以用于解释参数敏感性分析的结果。例如，利用上节的数据，我们可以计算出分支预测和 CACHE 的代价以及交互代价，如表 7.5 所示。我们可以看到，多数情况下两者发生正相关，定点程序相关程度高于浮点程序。正相关意味着要提高性能，我们需要同时优化这两个参数。

表 7.5 CACHE 和分支预测的交互代价

Prog.	***cost（CACHE）***	***cost（pred）***	***cost（c,p）***	***icost（c,p）***
dhryd	0.0094	0.21667	0.25139	0.02532
eon1	0.30295	0.06012	0.41667	0.0536
eon2	0.32616	0.06546	0.47054	0.07892
eon3	0.3651	0.06315	0.51103	0.08278
mcf	0.61989	0.0184	0.83527	0.19698
twolf	0.15664	0.14408	0.35795	0.05723
gzip	0.72328	0.059	0.95464	0.17236
vpr1	0.29844	0.1284	0.54112	0.11428
vpr2	0.50455	0.06255	0.63799	0.07089
gap	0.38752	0.06606	0.54605	0.09247
gcc	0.56527	0.07726	0.83239	0.18986
parser	0.4567	0.08289	0.80199	0.2624
bzip	0.5226	0.02855	0.55536	0.00421
crafty	0.52772	0.03704	0.6502	0.08544
vortex	0.34945	0.02768	0.4237	0.04657
fix avg.	0.407711	0.075821	0.585753	0.102221
whetd	0.02554	0.03617	0.05975	-0.00196
applu	0.62463	0.00281	0.63315	0.00571
Swim	0.80904	0.00277	0.82247	0.01066
equake	0.49988	0.0247	0.56344	0.03886
Mesa	0.26805	0.00128	0.27348	0.00415
Apsi	0.97549	0.0004	0.98782	0.01193
Art	0.82278	4E-05	0.84543	0.02261
wupwise	0.316	0.05556	0.43949	0.06793

ammp	0.44061	0.0132	0.44111	-0.0127
Mgrid	0.80528	0.00138	0.81187	0.00521
sixtrack	0.50152	0.08494	0.62429	0.03783
fp avg.	0.553529	0.020295	0.591118	0.017294

使用依赖图分析可以得到更为详尽的结论，但是对实际的高性能微处理器建立准确的依赖图模型需要更多的工作。下一步工作中我们将尝试解决这个问题。

7.3 提出和评估优化方案

处理器行为测量和瓶颈分析可以帮助我们提出和评估优化方案。提出和评估优化方案不仅仅是实现方案然后报告一个执行时间的比较结果，还应当分析优化措施为什么能够起作用，以及进一步优化的空间。

在龙芯 2 号设计过程中，我们根据性能分析的结果提出了许多优化措施，并做了深入的评估。本节给出两个例子来说明这个过程。

7.3.1 Load 猜测

7.1.3 节的表 7.2 和表 7.3 给出了一段代码和它的指令统计信息，对照汇编代码，我们知道这是一段数组拷贝代码，433e58（LW）从数组 a 取一个数据，433e64（SW）把它存入数组 b，每个循环拷贝一个数据。为何 LW 要在功能部件等待 18.5 拍，而 SW 要在发射队列等待 20.8 拍呢？有什么办法能提高这段循环运行的效率？

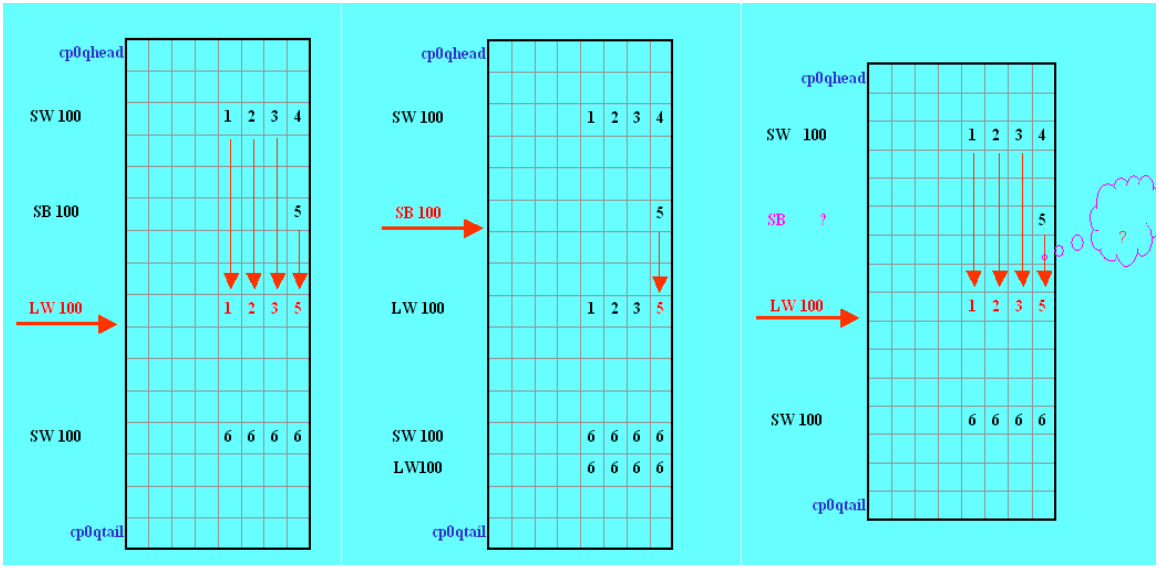


图 7.13 龙芯 2 号处理器 cp0 队列工作原理

对这段代码的深入分析暴露了一个问题。龙芯 2 号采用一个 Load/Store 队列（称为 cp0queue）来解决存储相关[HZL05]，cp0queue 是一个有序队列，它能检

测前后的 Load/Store 指令是否发生相关，在发现相关时能够自动进行值的传递，如图 7.13 所示。早期龙芯 2 号不支持猜测写回，也就是说，只要有更早的地址未知的 Store，则无论 CACHE 是否命中，Load 操作都不能立刻写回，而是要等待它之前所有的 Store 被发射到 cp0queue。如图 7.13 右边的 LW100，因为前面还有一个 SB 操作地址未知，它不能立刻写回。这样，虽然硬件可以同时容纳好几个循环，而且不同循环的 LW 操作完全不相关，但所有的 LW/SW 必须串行执行（每个 Store 发射时须等待同个循环的 Load 写回数据，而每个 Load 写回前须等待上一个循环的 Store 被发射到 cp0queue），从而导致所观察到的现象。

进一步的实验表明，Load 由于等待前面的 Store 地址解决而被延迟写回的情况相当普遍（平均约 30%），而在龙芯 2 号中一个被延迟写回的 load 至少需要 5 拍。这样，命中的 Load 操作延迟由 3 拍变为 3×0.7 + 5× 0.3 = 3.6，原本 Load 延迟就已偏长，这进一步恶化了情况。

解决的方法是让准备好的 Load 不管前面有没有地址未知的 Store 都写回，如果后来检测到发生相关则利用例外机制取消这个 Load。因为相关检测和取消机制是现成的，这个改动几乎没有增加任何硬件代价。由于实际发生相关的情况很少（直观上，编译器一般不会刚把数写回内存又去取），这个优化的效果非常理想。不仅这个循环的性能改善了，所有 SPEC CPU 程序以及两个典型的嵌入式基准程序 dhryd 和 whetd 都从中受益，定点程序性能平均提高达 11%，浮点程序性能平均提高 7%。如图 7.14 所示。我们还实验了使用各种预测表来减少猜测错误的效果，发现使用 1K 位硬件预测表，平均性能能够再提高约 1%。

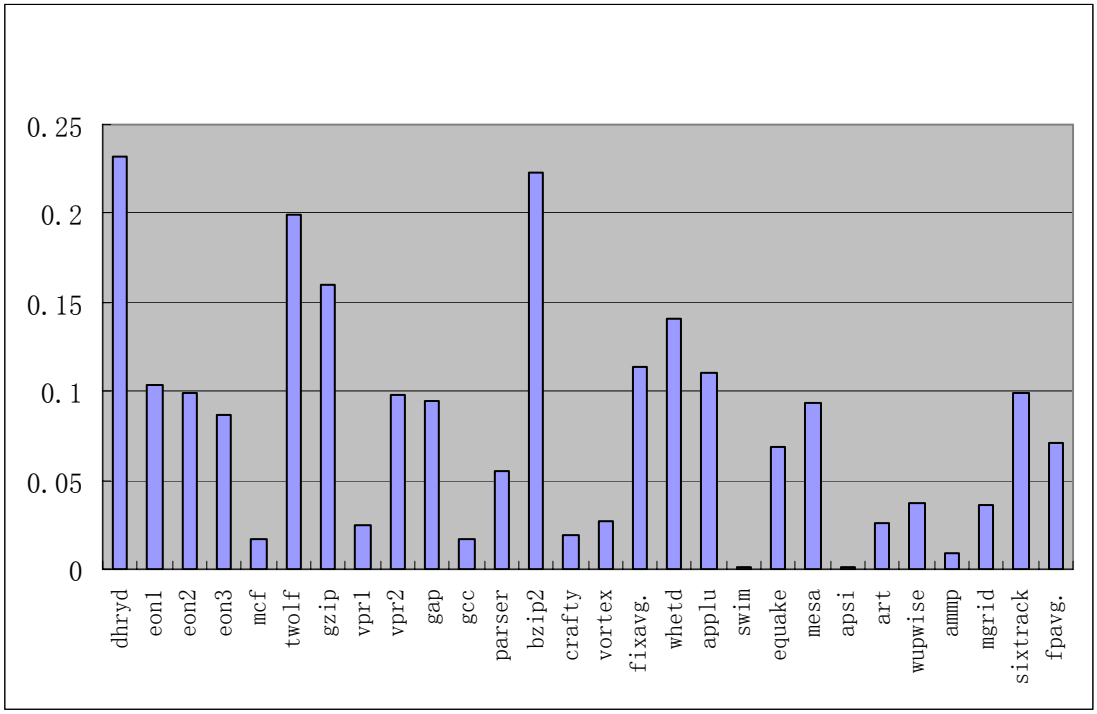


图 7.14 Load 猜测优化对性能提高的百分比

Load 猜测之所以能够起到比较大的效果，是由于它充分利用软件的行为特征，减小了指令延迟并提高了指令并行程度。

实际上，Load 延迟一直是动态调度处理器中一个瓶颈因素，而猜测技术是减小这个延迟的有效技术[]。上述方案对 Load 和 Store 之间的相关关系进行猜测。我们也可以对 Load 的地址进行猜测，减小

在这个优化方案上，我们还进行了进一步的思考和分析。一方面，另一方面，目前这个技术通过猜测是否和前面 Store 发生相关来减小 load-use 延迟，我们是否能够进行更多的猜测？由于龙芯 2 号 CACHE 数据比 CACHE 是否命中的结果早一拍出来，如果猜测 CACHE 命中，可以再早一拍写回数据，进一步减小 load-use 延迟。再进一步，我们也可以研究值预测等技术在龙芯 2 号上的应用。

7.3.2 龙芯 2B 访存优化

性能瓶颈分析表明龙芯 2B 的访存通路存在比较大的瓶颈，为此我们对它进行了更深入的分析并提出了一系列解决方案。

访存性能的两个主要指标是延迟和带宽。我们采用自底向上的流程，对访存通路做了全面的分析。我们首先测量北桥内存访问的硬件延迟和带宽；然后在龙芯 2 号上用专门程序测试程序所见的延迟和带宽；接着测试内存延迟和带宽对 SPEC CPU2000 性能的影响；再接着我们分析一些优化方案的效果。下面我们给出其中一些数据及其分析。

表 7.6 逻辑分析仪测量的 SDRAM 访问延迟

接口频率(MHz)	*内存频率(MHz)*	*倍频*	*同步/异步*	*访问延迟（ns）*
66	66	1	同步	225 （~15cycle）
66	66	1	异步	300（~20cycle）
66	133	1	异步	180（~12cycle）
66	100	1	异	225（~15cycle）

注 1：访问延迟指一个存储访问离开处理器接口到回到接口的时间

注 2：SDRAM 相关配置采用为龙芯 2 号 ev64240 开发板缺省值（Trcd=CL=Trp=3, Tras=7, refresh count=0x80, openpage=off, interleave=off）。

总体来说在 ev64240 的缺省设置下内存访问时间相当长。但是内存控制器本身应该不需要这么多拍：从一个行 active 到下一个行 active 最多只要 Tras+Trp=10 拍，第一个数据从内存条到内存控制器只需要 Trcd+CL＝6 拍。这说明北桥内部仲裁和缓冲的开销比较大。

表 7.7 程序测试的 SDRAM 访问延迟

选项					*RM7000*	*Godson2B*
CACHED	*Interleave*	*Openpage*	*Refresh*[1]	*Latency*[2]		
N	N	N	N	N	19	25
Y	N	N	N	N	21	25
N	Y	N	N	N	19	25[3]
Y	Y	N	N	N	21	25
N	N	Y	N	N	19	25
Y	N	Y	N	N	18	23
N	N	N	Y	N	19	25
Y	N	N	Y	N	19	24
N	N	N	N	Y	16	23
Y	N	N	N	Y	17	21
N	Y	Y	Y	Y	16	21
Y	Y	Y	Y	Y	15	20

注 1：Refresh 栏指是否对 SDRAM 的刷新周期 refresh counter 进行优化设置。Y 表示 counter=0x200，N 表示 refresh counter=0x80。

注 2：Latency 栏指是否对 SDRAM 的访问时序进行优化配置。Y 时 Trcd=CL=Trp=2,Tras=5，N 则 Trcd=CL=Trp=3，Tras=7。

注 3：Interleave 只有当处理器支持 split read 时才能发挥作用，Godson2B 不支持这个特性。

这里我们用一个简单的程序来计算存储访问的平均时间。具体方法是：在无其它负载的情况下，用连续进行多次同种类型的访问，然后取平均每个访问使用的总线周期数。我们测试了两种情况：

1. 对连续不同地址的 UNCACHED 读
2. 对连续不同 CACHE 块读

为了对比，我们测试了 RM7000 和 Godson2B 两种处理器。而且，因为我们

使用的 GT64240 北桥实际上支持不少提高访存性能的优化选项，我们还对各种优化选项做了一些测试。测试使用 EV64240 开发板，接口频率和 SDRAM 频率都设置为 66MHZ，使用同步方式，两倍频。

表 7.7 显示，Godson2B 缺省情况下访问一个 CACHE 块需要 25 个总线周期（表 7.6 中的 15 拍是接口外的延迟，从逻辑分析仪的信号看，连续块读时一个 validout 到下一个 validout 的时间是约 26 拍，两者基本吻合），如果使用 4 倍频，处理器核心将要忍受 100 拍的访存延迟。此外，我们可以看到同样的外部环境下 Godson2B 的平均访问时间比参考处理器 RM7000 多 5－6 个总线周期。每个 CACHE 块需要 25 – 4（4 个数据块） – 15 = 6 拍的接口开销。问题的根源是龙芯 2B 处理器总线接口部件没有流水和并行，一个访问从内部的访存失效队列发出后，直到它回到该队列，下一个访问都不能发出。

表 7.8 龙芯 2B 的 streams 带宽

接口频率（MHz）x 倍频	***内存频率（MHz）***	***COPY（MB/s）***	***SCALE（MB/s）***	***ADD（MB/s）***	***TRIAD（MB/s）***
66x1	66	43.21	38.09	42.23	45.20
66x1	100	37.9	32.5	38.4	40.33
66x2	66	54.30	52.14	55.88	60.5
66x3	66	63.3	61.40	65.27	65.66
66x4	66	65.25	64.02	65.27	68.54
66x4	133	68.15	65.89	75.77	78.86
100x2	100	82.35	79.10	84.75	91.75

Streams 基准程序[McC99]是广泛使用的处理器带宽测试程序。从表 7.8 可以看到，龙芯 2B 的 Streams 带宽也不理想，大部分情况下还不到峰值带宽的 1/8。在延迟比较长的情况下，又没有流水和并行，平均 25 个总线周期最多只有 4 个有效数据，带宽也就相应受到限制。

处理器内部的 CACHE 和乱序执行技术可以在一定程度上容忍访存延迟，那么龙芯 2B 的情况如何呢？

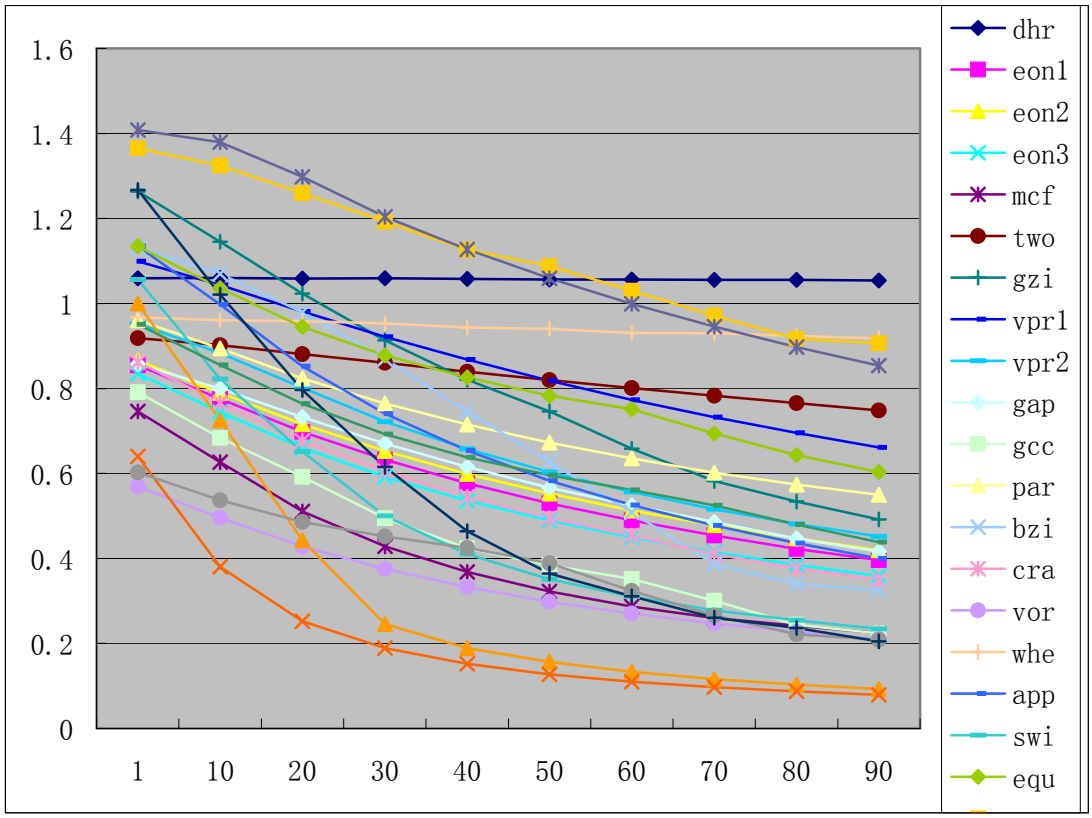


图 7.15 龙芯 2B 的性能和访存延迟的关系

图 7.15 给出了龙芯 2B 性能随访存延迟变化的情况。其中横轴为访存延迟（以处理器周期为单位），纵轴为 IPC。我们可以看到，对 SPEC CPU2000 程序，访存延迟的影响相当明显。大部分程序的 IPC 随着访存延迟增加下降比较快。

图 7.16 中我们给出了延迟和带宽对 SPEC CPU2000 性能的影响。我们把运行时间分为四个部分：T = Texec + Tsys + Tlatency + Tbandwidth，其中，

- Texec：理想 CACHE 情况下运行时间
- Tsys：访存失效队列和 CACHE 重填的代价，即访存延迟为 1 时的运行时间 - Texec，
- Tlatency：访存失效队列和接口串行限制的代价。即把访存队列和接口并发度理想化后的运行时间 - （Texec + Tsys）
- Tbandwidth：有限带宽的代价。即实际设置运行时间 - （Texec + Tsys + Tlatency）

基础配置同表 7.4 中的 Godson2B1。

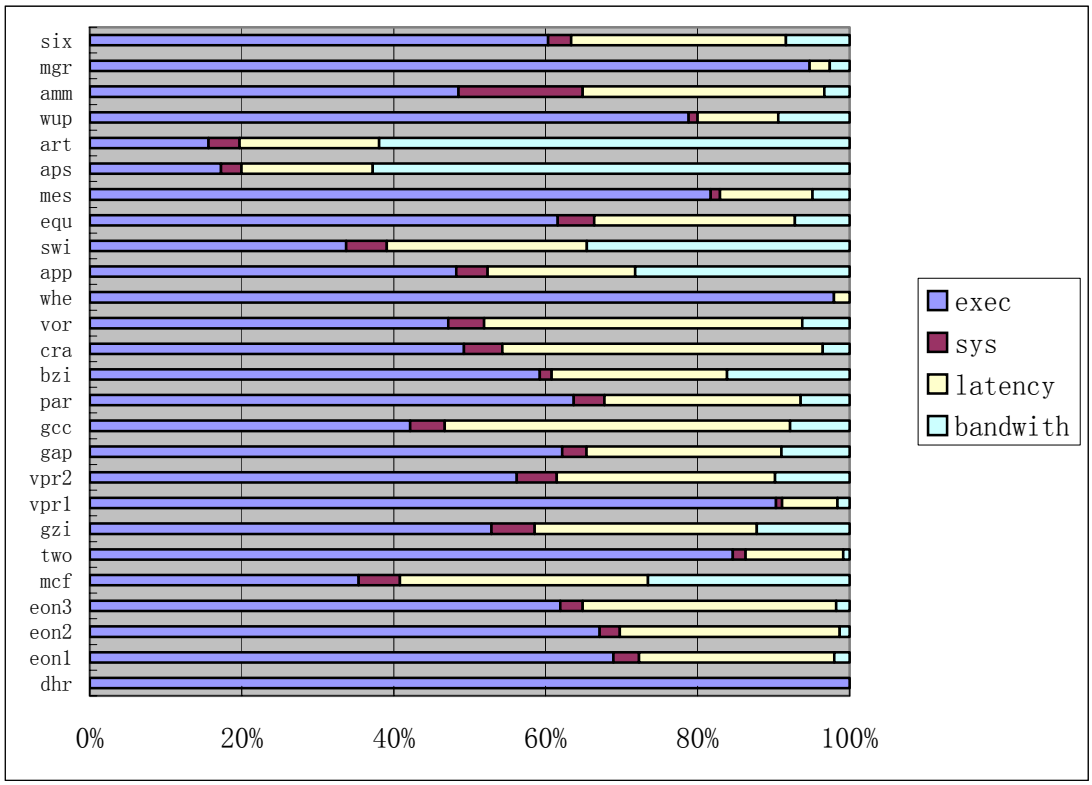


图 7.16 延迟和带宽对 SPEC CPU2000 性能的影响

图 7.16 显示，一些程序（包括 art、apsi、swim、applu 和 mcf）受带宽的影响比较大，但多数程序受延迟的影响比较大。

上述分析说明，龙芯 2B 访存性能受到多方面的限制，包括北桥性能、接口协议、接口宽度和 CACHE 层次设计等等。为了提高访存性能，我们必须协同考虑所有这些因素，而不能仅仅关心处理器内部。

提高存储性能涉及很多工作，这里我们简单介绍对二级 CACHE 性能分析。二级 CACHE 是一个有效的提高存储系统性能的方法，从图 7.17 中我们可以看到增加片外二级 CACHE 带来的好处，其中基本配置为表 7.4 的 Godson2B1，片外二级 CACHE 访问延迟设置为 12 拍，图中纵轴为 IPC。对定点程序，1M 二级 CACHE 平均加速 33%，8M 平均%37；而浮点分别为 22%和 32%。为了更好地理解二级 CACHE 的作用，图 7.18 给出了平均每条指令花费的存储访问周期（MCPI）对照图。

二级 CACHE 本身有许多技术参数需要权衡选择，包括片内/片外、大小、相联度、总线带宽、访问端口等等。我们对此做了大量的评估。目前龙芯 2 号中采用的是片外二级 CACHE，并复用系统总线。这种方案的好处是不会增加处理器的尺寸和功耗，而且可以根据不同的需求配备不同大小的二级 CACHE；复用系统总线可以减小对处理器引脚和封装的要求。但是这种片外二级 CACHE 的延迟和带宽比片内 CACHE 都有一定差距，为此我们正在积极寻求一些补偿措施，包括预取、一些专用缓冲区以及总线协议改进等。

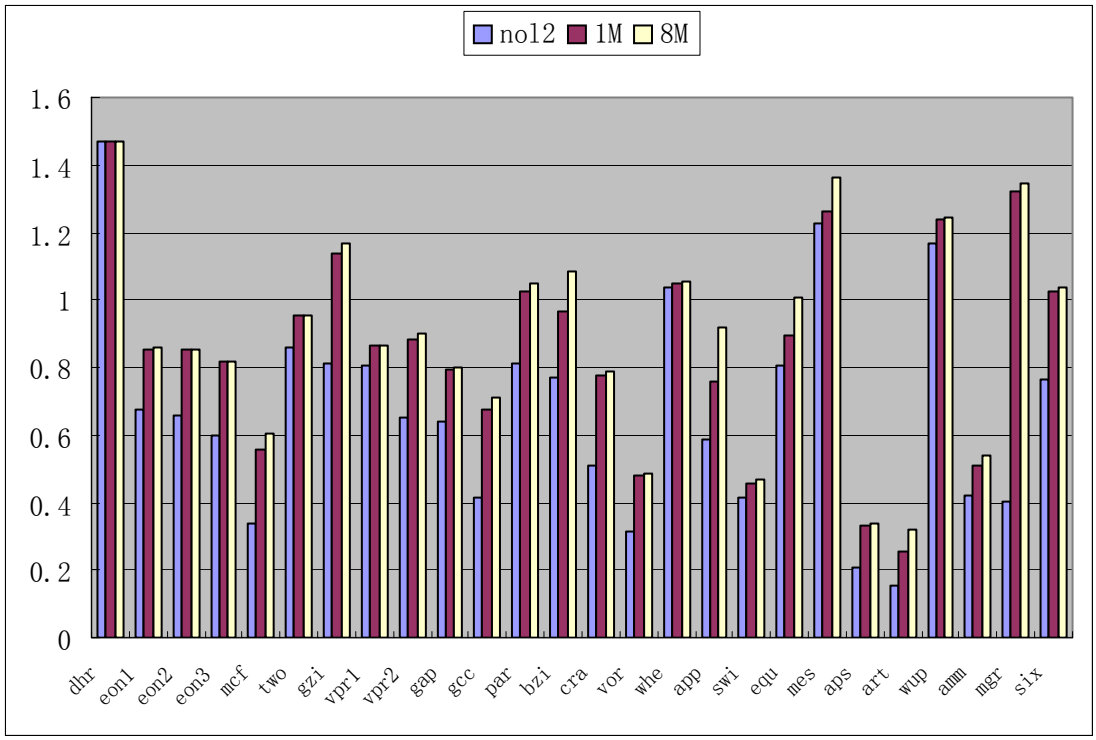


图 7.17 片外二级 CACHE 对程序 IPC 的影响

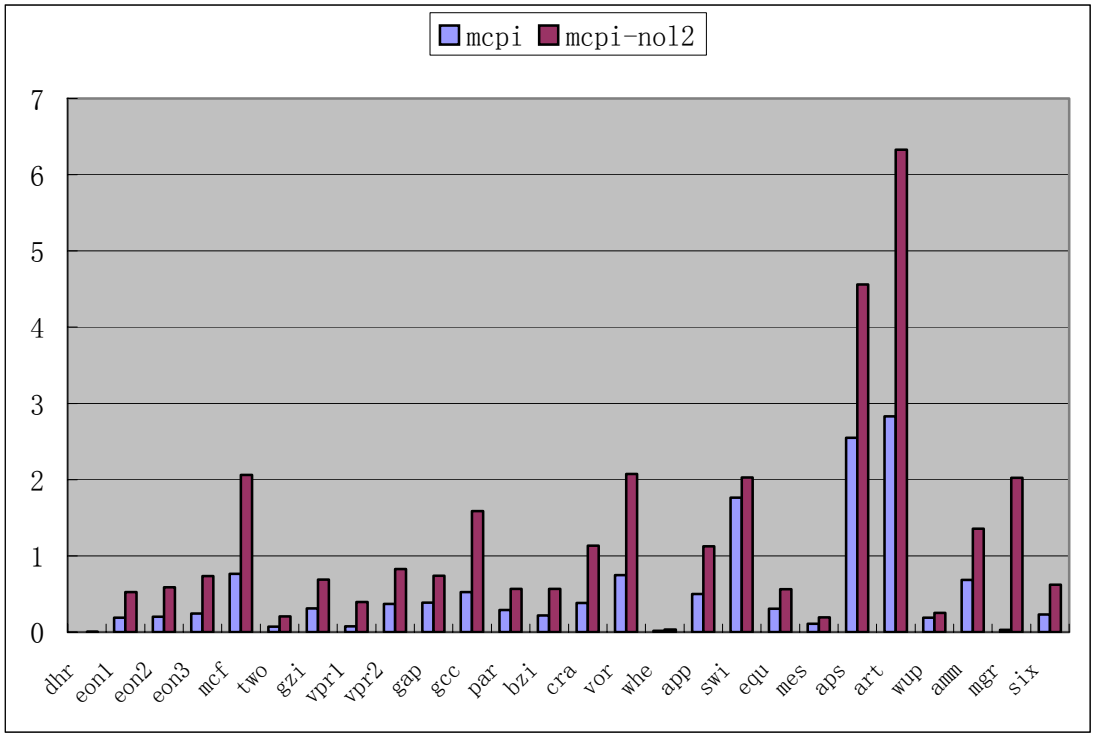


图 7.18 平均每条指令花费的存储访问周期

7.4 工作负载分析

龙芯 2 号性能分析目前主要使用的工作负载是 SPEC CPU2000。有些场合也

使用 whet stone、dry stone、stream 以及其它一些基准程序。

SPEC CPU2000 包括 14 个浮点程序和 12 个定点程序，如表 7.9 和表 7.10 所示。

表 7.9 SPEC CPU2000 浮点程序

程序	*语言*	*驻留内存大小（MB）*	*源代码大小（行）*	*描述*
168.wupwise	F77	176	2225	物理/量子色动力学
171.swim	F77	191	489	浅水模型
172.mgrid	F77	56	531	多格点解算器：3 维势场
173.applu	F77	181	4021	抛物线/椭圆偏微分方程
177.mesa	C		59262	3 维图形库
178.galgel	F90	63	15334	计算流体动力学
179.art	C	3.7	1279	图象识别/神经网络
183.equake	C	49	1518	地震波传播模拟
187.facerec	F90	16	2409	图象处理：脸部识别
188.ammp	C	26	13492	组合化学
189.lucas	F90	142	2999	数论/奇偶性测试
191.fma3d	F90	103	60122	有限元模拟
200.sixtrack	F77	26	47296	高能原子物理加速器设计
301.apsi	F77	191	7529	气象：污染源分布

表 7.10 SPEC CPU2000 定点程序

程序	*语言*	*驻留内存大小（MB）*	*源代码大小（行）*	*描述*
164.gzip	C	181	8633	压缩
175.vpr	C	50	17738	FPGA 布局布线
176.gcc	C	155	222190	C 编译器
181.mcf	C	190	2419	组合优化
186.crafty	C	2.1	21157	下棋
197.parser	C	37	11399	字处理
252.eon	C++	0.7	41011	计算机可视化
253.perlbmk	C	146	85053	Perl 编程语言
254.gap	C	193	71371	群论，解释器
255.vortex	C	72	67222	面向对象数据库
256.bzip2	C	185	4649	压缩
300.twolf	C	1.9	20467	布局和路由模拟器

SPEC CPU2000 广泛地用于体系结构研究，因此人们对它本身的特性也做了很多深入研究。龙芯 2 号性能分析中我们参考了其中一些研究结果，并从不同角度补充了一些研究。

7.4.1 相关研究

Cantin等人[CaH00]给出了SPEC CPU2000 在各种CACHE配置下的失效率表，其中部分数据被Patterson和Hennessy的《Computer Architecture－A Quantitative Approach》一书第三版采用。所有的数据可以从http://www.cs.wisc.edu/ multifacet/misc/spec2000CACHE-data获得。

Sair 等人[SaC00]研究了 SPEC CPU2000 的存储行为。他们的研究表明，多数 SPEC CPU2000 程序对存储系统没有很高的要求。他们也研究了各种 CACHE 配置的性能，结论包括：各种 CACHE 参数中，增大 CACHE 大小对性能提高最显著；指令 CACHE 的行大小和相联度增加都有好处，而数据 CACHE 则不一定，增加行大小和相联度对有些程序有好处，有些程序则相反。

Kandiraju 等人[KaS02]研究了 SPEC CPU2000 的数据 TLB 行为。他们发现，SPEC CPU2000 中大约有 1/4 的程序，包括 ammp、apsi、galgel、lucas、mcf、 twolf 和 vpr 有显著的 TLB 失效率。增加 TLB 的大小或者相联度都能减小这些失效率，软件 TLB 管理策略也很有可能能够显著降低失效率。

Lilja等人[KFM01]试图用缩减数据集的方法来解决SPEC CPU2000 reference 输入集模拟时间过长的问题。在此过程中他们生成了所有程序各种输入和优化级别下（O0 - O3）的profile。数据可以从http://www-mount.ee.umn.edu/~lilja/spec2000/ index.html获得。

Sherwood等人[SPH02，SPH03，PHC03]则观察到程序随时间变化行为的周期性，提出使用基本块矢量[SPC01]和聚类算法来寻找程序运行中有代表性的片断，然后通过模拟那些片断来估计整个程序的行为。详细信息可以参见http://www.cs.ucsd.edu/users/calder/ simpoint。

7.4.2 指令局部性和关键基本块

前面我们提到使用指令相关统计来研究处理器和程序行为。这种分析能否有效执行取决于指令局部性的情况，如果程序的性能主要由小部分的代码决定，指令相关统计分析的可行性将大大增加。表 7.11 给出了统计结果。统计方法如下：用 test 输入集完全运行所有 SPEC 程序，统计动态指令总数和每条指令执行次数，然后按执行次数从多到少排序所有的指令，累加前 n 条指令执行次数，找到使执行次数大于指定域值的 n。

表 7.11 SPEC CPU2000 的指令局部性

程序	*静态指令总数*	*占总运行次数 90%的指令数*	*百分比*	*占总运行次数 80%的指令数*	*百分比*
Ammp	236612	1237	0.005	887	0.004
applu	163804	4001	0.024	3160	0.019

apsi	243892	2082	0.009	1298	0.005
art	136104	233	0.002	145	0.001
bzip2	143392	623	0.004	395	0.003
crafty	199532	6527	0.033	4497	0.023
eon1	512532	5108	0.010	3273	0.006
eon2	512532	4557	0.009	2629	0.005
eon3	512532	3736	0.007	2459	0.005
equake	153512	1846	0.012	1373	0.009
gap	280300	4300	0.015	2642	0.009
gcc	587236	27671	0.047	15817	0.027
gzip	150652	478	0.003	384	0.003
mcf	129308	272	0.002	185	0.001
mgrid	152968	178	0.001	126	0.001
parser	160016	2697	0.017	1700	0.011
perlbmk	372840	7339	0.020	4082	0.011
swim	184444	626	0.003	307	0.002
twolf	215232	2172	0.010	1453	0.007
vortex	342844	3099	0.009	1328	0.004
vpr1	203876	1164	0.006	802	0.004
vpr2	203876	444	0.002	282	0.001
wupwise	184256	700	0.004	451	0.002

从表 7.11 可以看到，很多程序中 90%的时间都在运行其中不到 1%的一些指令，具有非常好的局部性。除了执行次数，我们对 CACHE 失效次数、分支预测次数等参数也做了类似的分析，结论都是相似的。

根据指令统计的结果，我们把每个程序中执行次数最多的那些基本块称为关键基本块。对关键基本块行为的分析常常能够提高对程序行为的认识，也有助于发现处理器优化方案。

7.4.3 程序随时间变化的运行特性

了解程序随时间变化的特性有许多潜在的用处。例如，Simpoint 就是基于对程序运行中周期性行为的观察得出的一种有效采样方法。程序随时间变化的行为也可能用于指导程序的动态优化。

我们在龙芯样机上使用龙芯 2 号的硬件性能计数器来得出了 SPEC CPU2000 随时间变化的 IPC 情况。一些例子如图 7.19-7.22 所示，其中横座标的单位为 1 亿条指令。有些程序总指令数太多，为了清晰起见，最多只取前 200 亿条的分布。

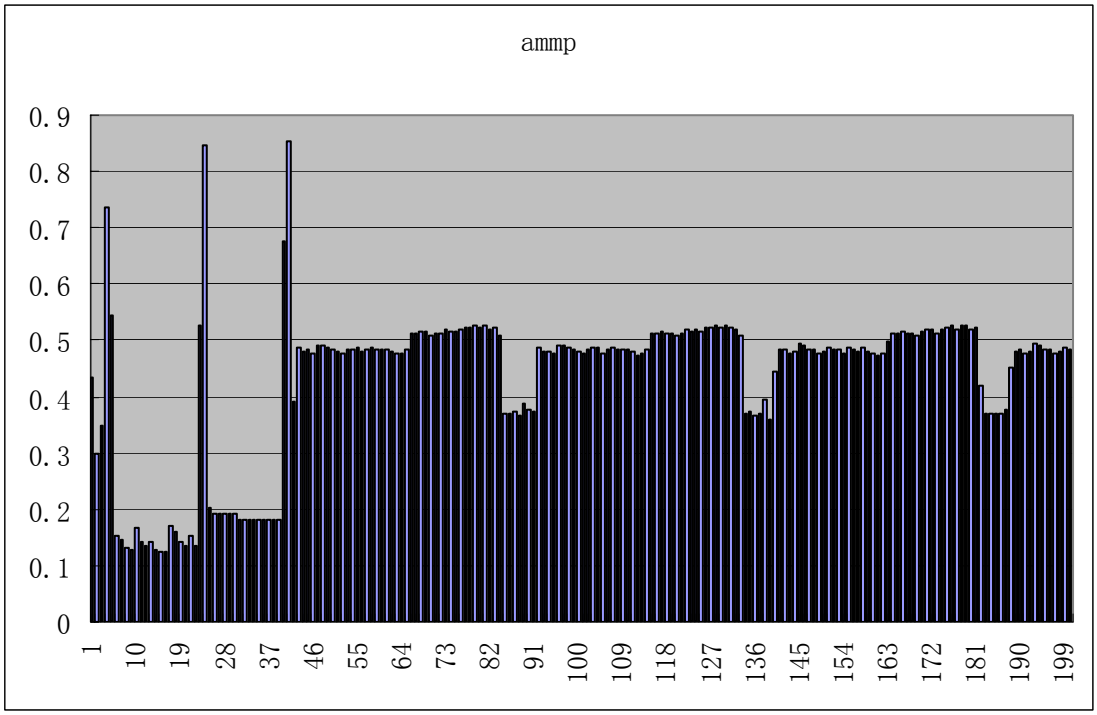


图 7.19 ammp 的 IPC 随时间变化情况

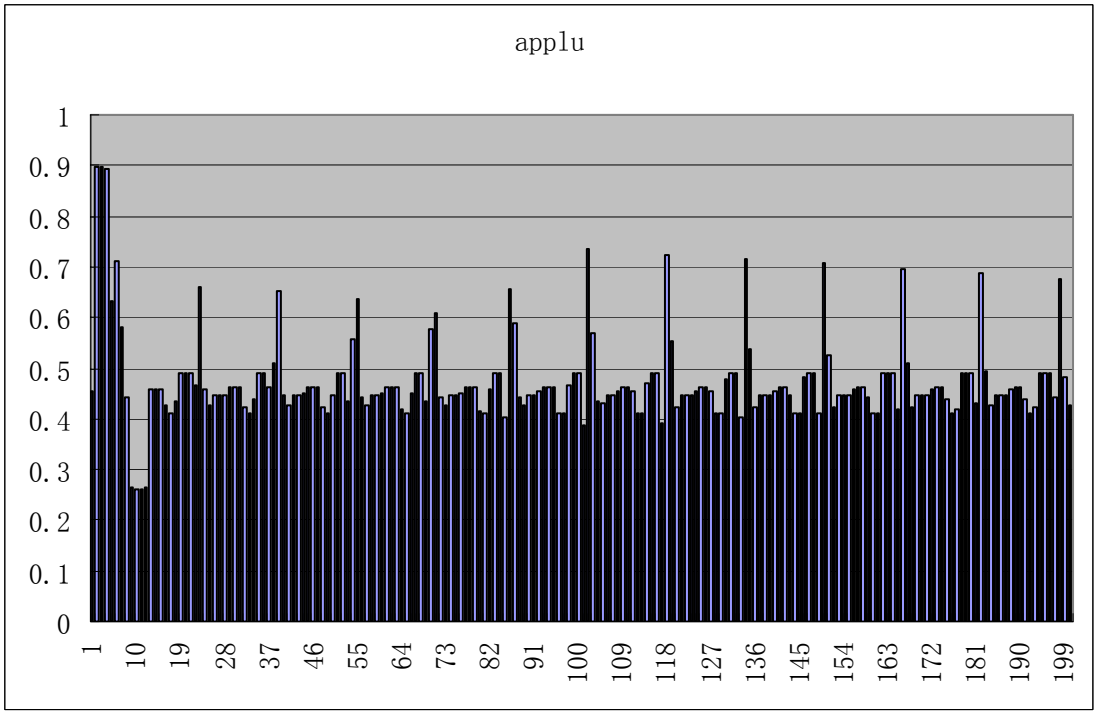


图 7.20 applu 的 IPC 随时间变化情况

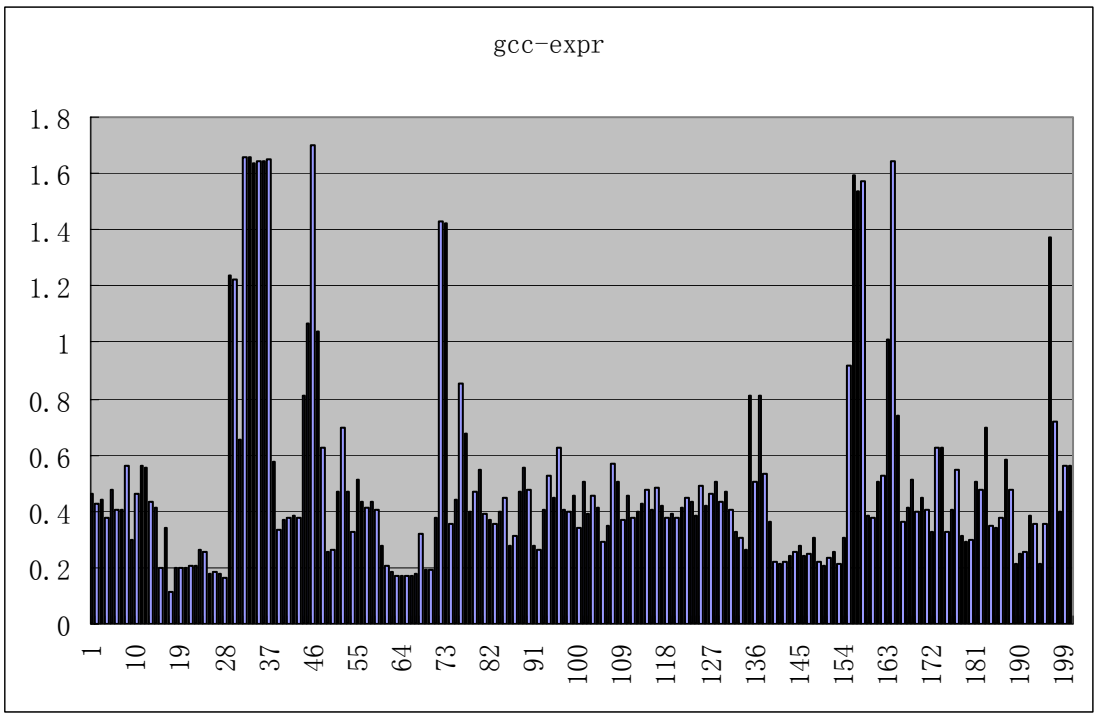


图 7.21 gcc-expr 的 IPC 随时间变化情况

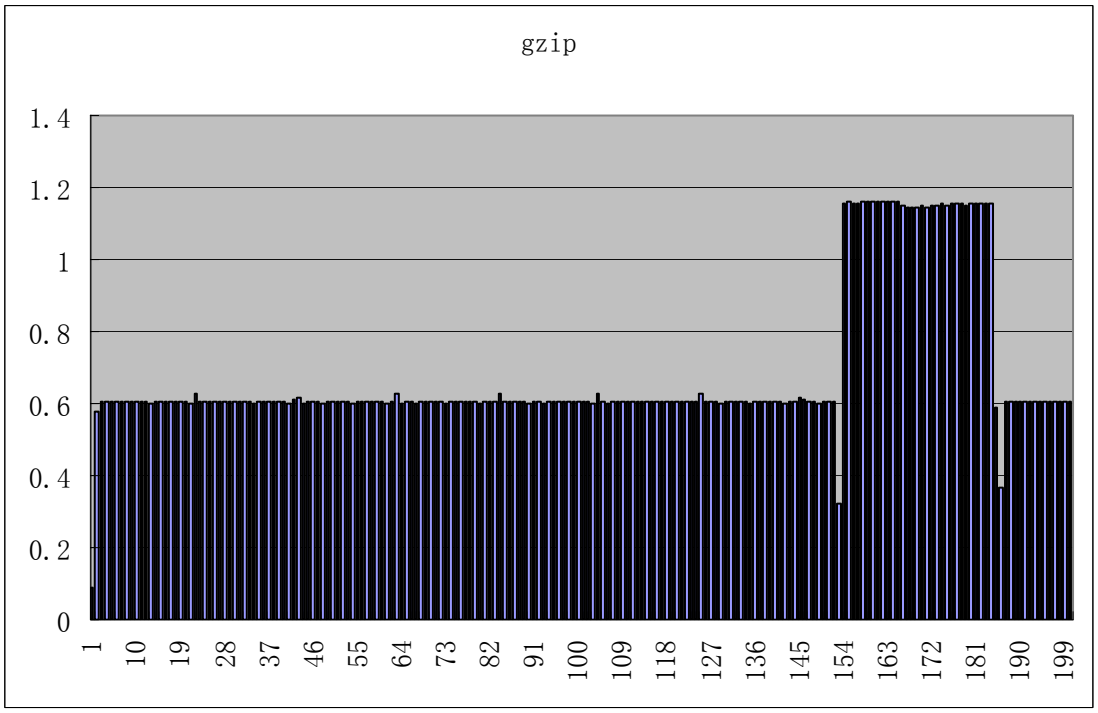


图 7.22 gzip-random 的 IPC 随时间变化情况

从这些图中我们可以看到，各个程序随时间变化的情况差别比较大。有些程序，如 gzip-random，IPC 变化很少，而一些程序则相反，如 gcc-expr。第四章我们评估 simpoint 时，曾经碰到 K=10 时 gzip-program 误差比较大的情况，这里我们给出它的 IPC 随时间变化情况，如图 7.23 所示。我们可以看到，gzip-program 有比较复杂的阶段性行为，能够分成比较多不同的样本类。这就解释了 K=10 时

误差比较大的原因。

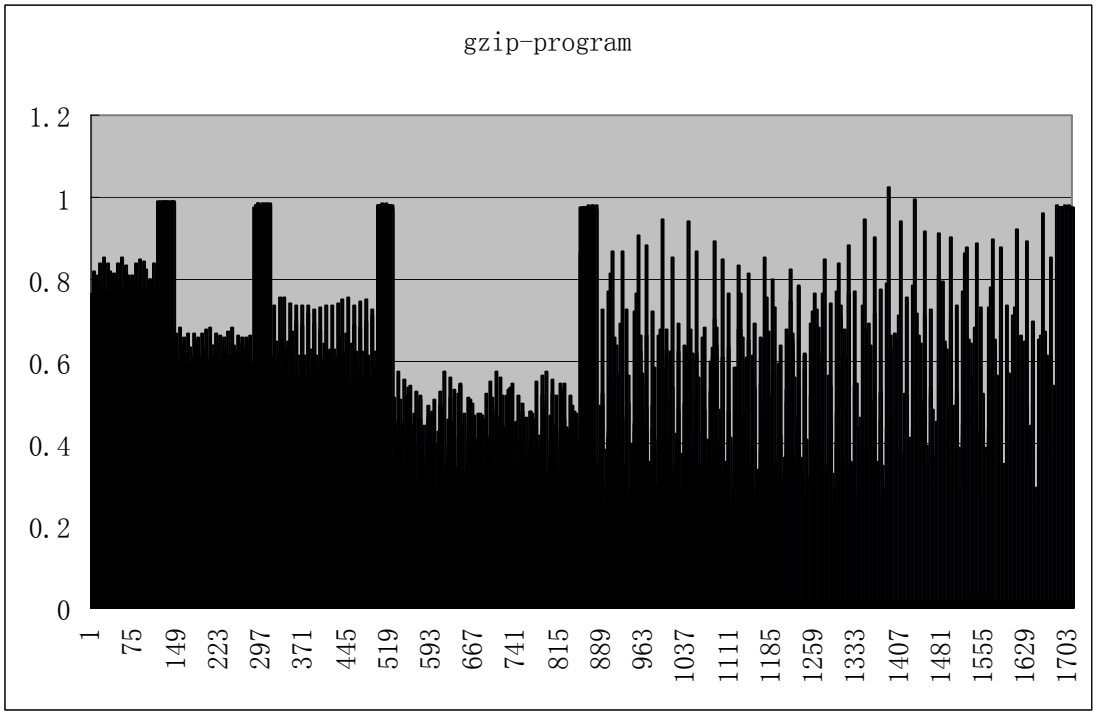


图 7.23 gzip-program 的 IPC 随时间变化情况

7.5 编译器的影响

编译器负责完成从高级语言到机器代码的映射，这项工作的好坏常常对性能有很大的影响。完整的处理器性能分析工作不能缺少对编译影响的分析。我们首先考察了不同的编译器以及同一编译器的不同选项对目标工作负载性能的影响，然后讨论我们对龙芯 2 号编译系统所做的一些优化。

7.5.1 不同编译器和编译器选项的性能

我们在 SGI Origion2000 机器上用不同的编译器和不同选项得到了一组 SPEC CPU2000 分数，如表 7.12 所示。该机器有 7 个 250MHZ R10000 处理器，操作系统为 IRIX6.5，编译器为 MIPSpro-7.3.1 和 gcc-3.2.3。

表 7.12 不同编译器和不同选项的 spec2000 分数

	-O2 -mips2（O32）	***-O2 -mips3***	***-O2 -mips4***	***-O3 -mips4***	***-Ofast =ip27***	***Base: 带反馈***[2]	***peak***	***-O2 Gcc-3.2.3***
164.gzip	87.7	133	138	137	137	157	158	89.5
175.vpr	102	136	148	148	164	177	177	119
176.gcc	130	160	162	163	178	208	207	N.A.
181.mcf	198	218	221	229	240	244	246	212
186.crafty	77.6	180	186	172	171	212	215	136
197.parser	89.4	128	128	128	169	172	181	93.9
252.eon	73.4	122	126	155	165	229	226	N.A.
253.perlbmk	91	113	116	115	122	148	146	101
254.gap	87	108	109	108	117	134	137	94.1
255.vortex	78.9	118	117	121	166	164	264	85.3
256.bzip2	107	135	134	132	191	185	204	117
300.twolf	178	209	233	232	288	300	300	N.A.
整数平均	103	143	147	149	170	189	200	112
168.wupwise	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A	136
171.swim	N.A	136	148	259	269	276	271	156
172.mgrd	N.A.	105	97.7	131	142	168	122	103
173.applu	N.A.	112	118	119	101	149	155	99.1
177.mesa	105	159	164	157	163	172	188	128
178.galgel	N.A	236	234	541	562	568	647	N.A
179.art	385	433	446	445	686	695	672	507
183.equake	126	118	110	109	129	183	185	144
187.facerec	N.A	213	211	232	216	237	242	N.A.

2 PASS1 : -Ofast=ip27 -IPA:use_intrinsic -fb_create /tmp/SPEC2000/FBDIR/base/$(EXEBASE; PASS2 : -Ofast=ip27 -IPA:use_intrinsic -fb_opt /tmp/SPEC2000/FBDIR/base/$(EXEBASE)

	-O2 -mips2（O32）	*-O2 -mips3*	*-O2 -mips4*	*-O3 -mips4*	*-Ofast =ip27*	*Base: 带反馈*	*peak*	*-O2 Gcc-3.2.3*
188.ammp	142	186	188	195	198	200	205	170
189.lucas	N.A.	97.8	99	190	183	196	198	N.A.
191.fma3d	N.A.	149	148	142	151	152	172	N.A.
200.sixtrack	N.A	112	114	116	116	118	118	45.2
301.apsi	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.	N.A	78.9
浮点平均	164	155	156	191	201	222	223	129

表中标记为 N.A.（not available）的数据是由于编译器或者操作系统的问题导致该配置无法正常运行。例如，gcc 编译器不支持 Fortran90 程序，因此 178.galgel、187.facerec、189.lucas、191.fma3d 四个程序无法运行。MipsPro 编译器对 mips2 选项只有有限的支持，有些程序在这种选项下不能正常运行；此外，由于超出操作系统的堆栈限制，MipsPro 编译的 168 和 301 无法运行。

从表 7.12 可以看到，MipsPro 的性能一般优于 gcc，在 peak 优化情况下（没有直接可以使用 gcc peak 优化选项，但一般来说，gcc 的 peak 性能和-02 差距比较小）平均性能高出 70%以上。

MipsPro 不同的优化选项性能差异很大，最好和最差的优化差了几乎一倍。.对定点程序,有三个效果比较明显的优化：

（1） 从-mips2 到-mips3，定点程序整体提高近 40%,浮点中能正常运行的几个程序效果也相当明显。这两个选项所生成的程序 ABI 不同，-mips2 时 ABI 为 032，-mips3 时是 N32。N32 是 032 的改进版本，具有一些性能上的优势。此外，IRIX6.5 操作系统是为 N32 程序优化的。

（2） Ofast 选项：对定点程序，它比-mips4 -03 明显提高（149vs170），其中的优化包括对特定软件硬件系统的优化。如针对特性处理器的优化，使用大页面减小 TLB 失效等。

（3） 反馈优化：使用 train 的数据集训练之后,效果比较明显（>10%）。

对浮点程序，mips4-02 到 mips4-03 有大幅度的提高，可能的原因是-03 多做了一些有效的循环级优化，浮点程序中有很多规则的循环，优化的潜力大。

7.5.2 机器相关优化

针对龙芯 2 号处理器特性，我们在 gcc 编译器上实现了一些机器相关的优化，

并取得了良好的效果。例如，我们为 gcc 提供的龙芯 2 号处理器流水线描述，能够把 SPEC CPU2000 的定点程序性能提高 2%左右。其它机器相关优化包括利用龙芯 2 号提供的扩展指令、针对龙芯 2 号微体系结构调整分支指令和存储分配、针对目标操作系统优化等等。

7.5.3 链接级优化

7.5.1 节中我们比较了 MipsPro 和 Gcc 编译器的性能，发现前者的性能高很多。MipsPro 作为一个 MIPS 平台的商业编译器，一方面它对 MIPS 体系结构和特定目标机器做了更完整、细致的调整和优化，另一方面它比 Gcc 多一些重要的功能，包括过程间优化。过程间优化可以利用函数之间的信息做更多的优化，例如 inline 不同文件里的函数，进行全局常数传播等；结合 profile 反馈的信息，还能做全局的代码重排等优化。目前 Gcc 没有过程间优化的能力，因此我们尝试用链接级优化工具来弥补这个缺憾。为此我们移植了 ALTO 链接级优化器[RDW01]。

ALTO 链接级优化器能够对 gcc 编译的程序进行再优化，并起到明显效果。图 7.24 显示了 ALTO 对 SPEC CPU2000 定点程序的优化效果，使用的处理器为龙芯 2C，优化对象为 gcc 3.4.1 优化编译的可执行代码，图中纵轴为 SPEC CPU2000 分数提高的百分比。

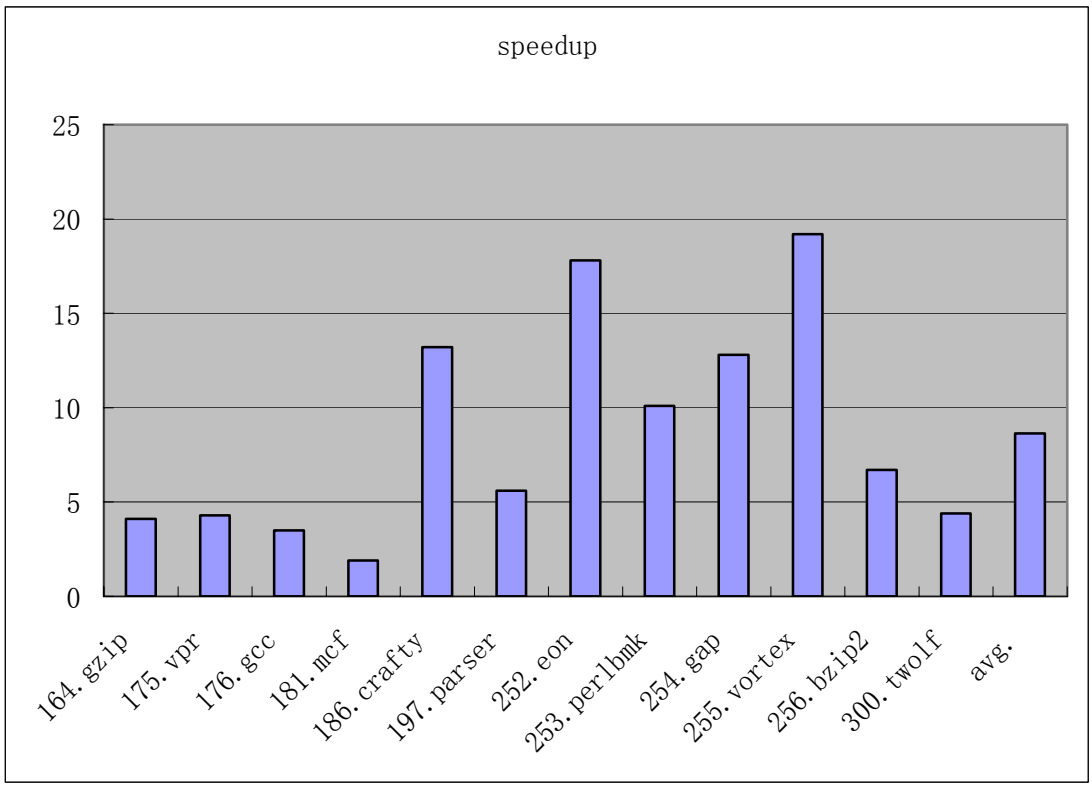


图 7.24 ALTO 的加速效果

7.6 操作系统的影响

这里我们主要考察了操作系统对 SPEC CPU2000 的影响。一般认为，SPEC CPU2000 工作负载对操作系统的考验很小，因此人们常常在研究中忽略操作系统的影响。本文的研究表明，对 SPEC CPU2000 的 test 输入集，用户级模拟得到的性能一般略高于全系统模拟的结果，但是差距不大，平均不到 5%。

我们进一步发现，SPEC CPU2000 的系统态时间，绝大部分花在 TLB 例外处理中。因此，如果用户级模拟正确模拟了 TLB 的代价，应该可以得到更精确的结果。同时，基于这个观察，我们重点对 TLB 处理进行分析和优化，并取得了良好的效果。

7.6.1 操作系统对 SPEC CPU2000 的影响

龙芯 2 号的性能模拟器都同时支持系统态和用户级模拟。前者运行一个操作系统，然后在操作系统中运行程序，后者则直接运行静态链接的程序，用系统调用代理的方法来处理程序中的系统调用。

表 7.13 操作系统对 SPEC CPU2000 的影响

	IPC	*User-IPC*	*OS-IPC*	*OS-rate*	*Usermode-IPC*	*Usermode-err*
Ammp	0.240	0.240	0.200	0.040	0.240	0.001
Applu	0.580	0.600	0.100	0.040	0.605	0.044
Apsi	0.500	0.660	0.080	0.280	0.509	0.017
Art	0.150	0.150	0.210	0.000	0.146	-0.028
Bzip2	0.890	1.000	0.200	0.140	0.964	0.083
Crafty	1.030	1.040	0.150	0.020	1.049	0.018
eon1	0.930	0.960	0.330	0.040	0.966	0.039
eon2	0.940	0.960	0.330	0.030	0.966	0.028
eon3	0.910	0.910	0.310	0.010	0.919	0.010
Equake	0.790	0.820	0.200	0.050	0.817	0.034
Gap	0.720	0.810	0.160	0.140	0.827	0.149
Gcc	0.690	0.710	0.240	0.030	0.711	0.030
Gzip	0.790	0.810	0.200	0.030	0.814	0.030
Mcf	0.410	0.410	0.150	0.010	0.413	0.008
Mesa	1.070	1.110	0.150	0.040	1.088	0.016
Mgrid	0.510	0.540	0.090	0.080	0.516	0.012
Parser	0.730	0.750	0.200	0.040	0.755	0.034
Sixtrack	0.960	1.030	0.240	0.100	1.043	0.086
Swim	0.400	0.460	0.100	0.150	0.453	0.132
Twolf	0.820	0.830	0.220	0.020	0.838	0.022
vpr1	0.960	0.970	0.270	0.010	0.966	0.007
vpr2	0.680	0.680	0.230	0.010	0.684	0.006

Wupwise	1.070	1.230	0.080	0.140	1.173	0.096
avg.	0.729	0.769	0.193	0.063	0.759	0.041

表 7.13 包括两套数据，2-5 列数据是使用 ICT-godson 全系统模拟器得到的结果，第 6 列数据是使用同一模拟器对相同程序进行用户级模拟得出的结果，第 7 列给出用户级模拟相对全系统模拟的误差。在全系统模拟中，我们对处理器在系统态下的运行数据做了单独的统计。OS-IPC 即为系统态下运行的指令数除以系统态时钟周期数的结果，User-IPC 是扣除系统态指令和周期数后计算的结果，OS-rate 则是系统态运行时间和总运行时间的比率。Usermode-IPC 和 Usermode-err 分别是用户级模拟的程序 IPC 以及它和全系统模拟结果相比的误差。运行的程序为 SPEC CPU2000 test 输入集，每个程序最多运行 10 亿拍。统计结果表明，对大部分程序，操作系统态所占用的时间不是很明显。也有少数程序包括 apsi，bzip2, gap, swim, wupwise 等，处于操作系统态的时间超过了 10%。下节中我们可以看到这些程序 TLB 例外花费的时间比较多。但需要说明的是，TLB 例外花费的时间和 TLB 例外的数目并不是一致的，因为 MIPS 中 TLB 例外有几种，不同的例外处理时间差别很大。表 7.13 中用户级模拟的结果一般高于全系统模拟，暗示目前用户级模拟中对 TLB 失效的代价估计过低。

7.6.2 TLB 分析和优化

表 7.14 SPEC CPU2000 的 TLB 例外

Prog.	*TLB 例外总时钟数*	*TLB 例外发生的次数*	*系统态总时钟数*	*TLB 例外在系统态占的比例*	*平均每次 tlb 例外需要花费的时钟*
Ammp	171169272	1130061	189892683	0.901	151
Applu	37716056	149955	40261758	0.937	252
Apsi	639579403	107344	640924840	0.998	5958
Art	18857589	67051	24252920	0.778	281
Bzip2	348453436	2466856	358071251	0.973	141
Crafty	112389176	908559	114868423	0.978	124
Dhryd	340123	46	608678	0.559	7394
Eon1	4805941	2772	12795341	0.376	1734
Eon2	5213104	2366	10595549	0.492	2203
Eon3	5272466	2646	11093329	0.475	1993
Equake	54191156	105712	76578397	0.708	513
Gap	142349586	57987	172782435	0.824	2455
Gcc	68476761	251704	78082060	0.877	272
Gzip	22641260	8187	35167381	0.644	2766
Mcf	68252719	577612	68265740	1.000	118
Mesa	42022638	25701	44627525	0.942	1635
Mgrid	219053155	68165	221562600	0.989	3214
Parser	36396985	103661	59760548	0.609	351
Sixtrack	90779743	56379	175782857	0.516	1610
Swim	223615391	156669	225606229	0.991	1427
Twolf	6740516	621	14707056	0.458	10854
Vortex	162970007	1119861	196920321	0.828	146
vpr1	4251482	382	8773639	0.484	11129
vpr2	46406956	291472	49400738	0.939	159
Whetd	2818749	189	3530996	0.798	14914
Wupwise	198348113	33591	200326359	0.990	5904

从表 7.14 我们可以看到，系统态下的时间大部分都用于处理 TLB 例外，平均高达 77%。此外，平均每次 TLB 处理的时间不同的程序差别比较大。

我们从硬件和软件方面对 TLB 做了进一步的分析和优化。早期龙芯 2 号处理器使用 8 项指令 TLB，统计表明一些程序的指令 TLB 失效比较多。为此我们分析了指令 TLB 项数对性能的影响。表 7.15 给出了 16 项、32 项、64 项指令 TLB 对 SPEC CPU2000 性能影响的情况。其中基本配置为 godson2B1，输入集为 test，最多运行 10 亿个时钟周期。数据表明，指令 TLB 项数对 eon、gap、gcc、crafty、vortex 和 sixtrack 等程序的性能有明显的影响。根据这个分析结果，龙芯 2 号的后续版本将指令 TLB 项数增加到 16 项。

表 7.15 指令 TLB 项数对 SPEC CPU2000 性能的影响

	godson2B1	*itlb16*	*提高的百分比*	*itlb32*	*提高的百分比*	*itlb64*	*提高的百分比*
dhryd	1.467	1.467	0.003	1.467	0.003	1.467	0.004
eon1	0.676	0.709	4.948	0.744	10.064	0.749	10.822
eon2	0.657	0.686	4.462	0.714	8.685	0.719	9.360
eon3	0.597	0.628	5.160	0.661	10.696	0.664	11.204
mcf	0.340	0.341	0.291	0.341	0.279	0.341	0.306
twolf	0.755	0.762	0.896	0.763	0.990	0.763	1.005
gzip	0.812	0.812	0.005	0.812	0.010	0.812	0.010
vpr1	0.875	0.875	-0.003	0.875	-0.034	0.875	-0.042
vpr2	0.652	0.652	0.083	0.653	0.135	0.653	0.118
gap	0.639	0.680	6.331	0.682	6.658	0.682	6.688
gcc	0.414	0.426	3.097	0.437	5.595	0.442	6.986
parser	0.810	0.811	0.144	0.812	0.307	0.812	0.327
Bzip2	0.770	0.770	-0.001	0.769	-0.029	0.770	0.010
Crafty	0.508	0.529	4.032	0.553	8.874	0.554	9.011
Vortex	0.313	0.325	3.893	0.337	7.605	0.350	12.022
Whetd	1.072	1.070	-0.228	1.075	0.261	1.072	0.002
Applu	0.587	0.587	0.024	0.588	0.123	0.589	0.245
Swim	0.412	0.412	0.058	0.412	0.056	0.412	0.056
Equake	0.804	0.823	2.420	0.824	2.590	0.825	2.607
Mesa	1.227	1.229	0.134	1.229	0.143	1.229	0.143
Apsi	0.205	0.205	0.015	0.205	0.015	0.206	0.019
Art	0.152	0.152	-0.007	0.152	0.000	0.152	0.000
Wupwise	1.165	1.165	-0.002	1.165	0.012	1.165	0.004
Ammp	0.419	0.422	0.738	0.424	1.203	0.424	1.213
Mgrid	0.402	0.402	0.030	0.403	0.067	0.402	0.052
Sixtrack	0.766	0.786	2.695	0.797	4.133	0.798	4.173
平均			1.508		2.632		2.936

软件方面我们在操作系统中实现了大页面支持和软 TLB[BKW94]支持。前者利用 MIPS 处理器支持 4k 到 256K 大小的页面的特性，让操作系统支持更大的页面，以提高 TLB 覆盖的程序空间从而减小 TLB 失效率。对 SPEC CPU2000 程序，使用大页面的效果比较显著（见图 7.25）。后者使用软件来预取和缓存常用的页表项，以其降低 TLB 失效时的访存量以及访存失效量。我们实验了缓存最近使用的页表项的效果，发现这种方法对提高一些 SPEC CPU2000 程序的性能有明显效果。图 7.26 显示了一个 4KB 直接相联的缓存的效果。我们可以看到，对 Kandiraju 等人[KaS02]指出的 TLB 失效率比较显著的程序，如 vpr, mcf, vortex, apsi 等，这种方案都能取得明显的效果。

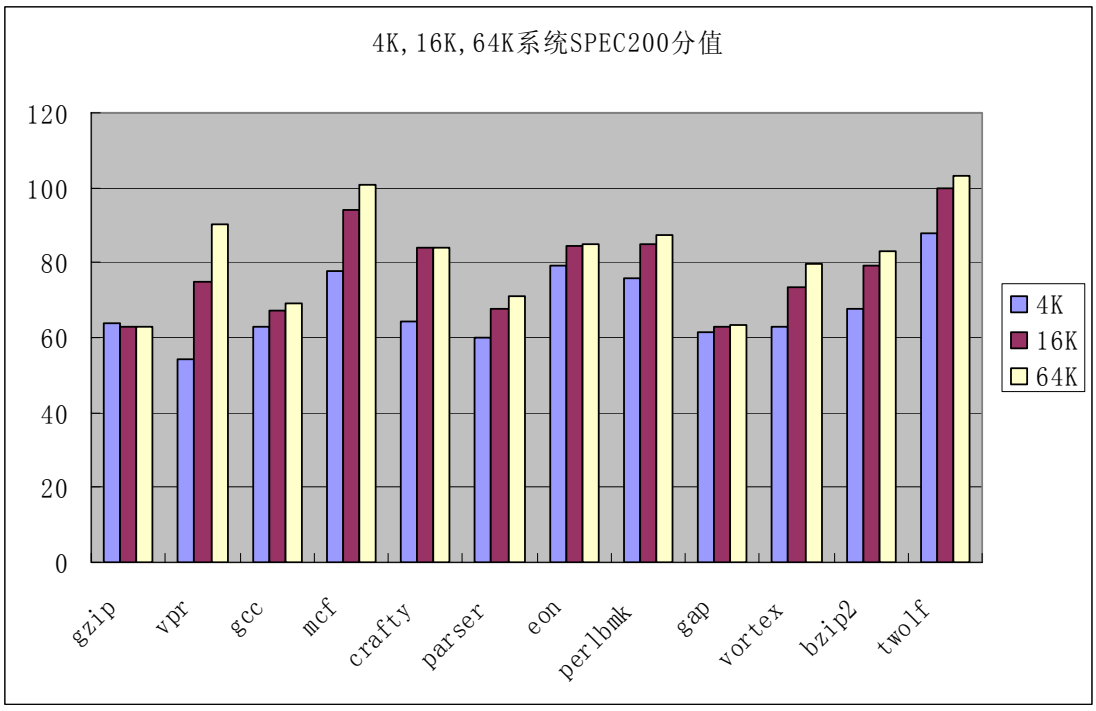


图 7.25 不同 TLB 页大小对 SPEC CPU2000 分值的影响

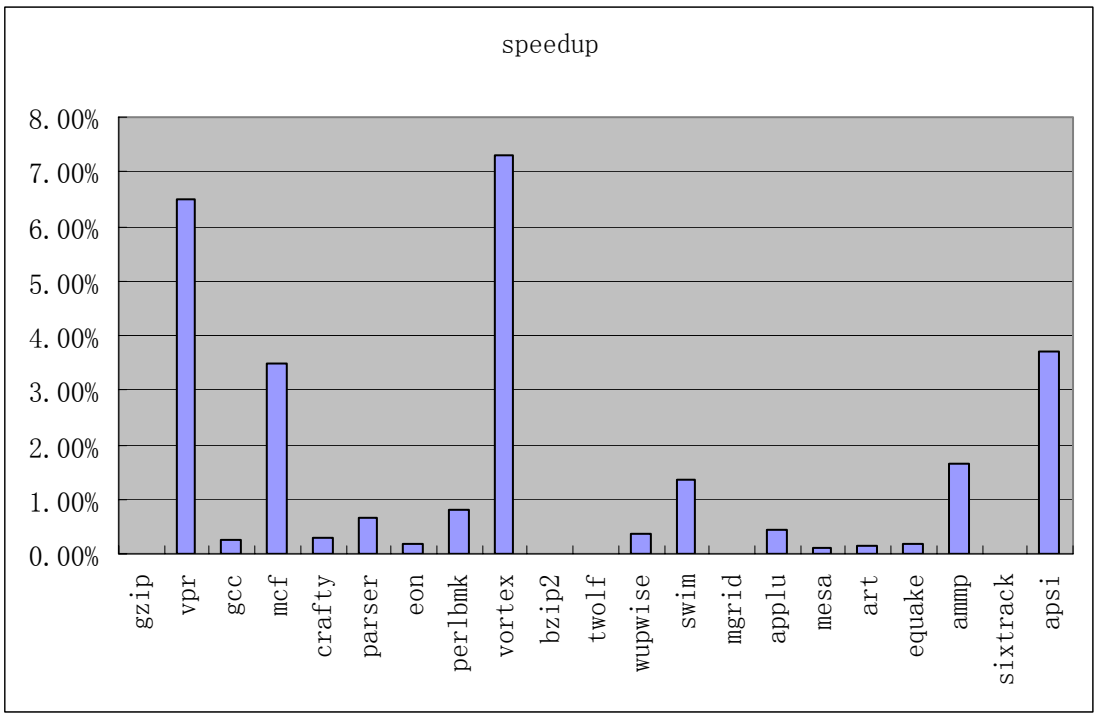


图 7.26 软 TLB 对 SPEC CPU2000 程序的影响

第八章 结束语

面对越来越复杂的处理器和工作负载，如何有效地进行性能分析和优化，是一个急需解决的问题。性能分析工作的核心是各种处理器和工作负载模型的建立和验证。这些模型必须很好地满足准确性、速度和灵活性的要求。本文紧密高性能处理器（龙芯 2 号[HZL05]）的设计，研究了性能分析环境的设计和性能分析方法，并利用它们对龙芯 2 号处理器进行了有效地分析和优化。

8.1 本文工作总结

本文首先对性能分析的相关工作做了一个简要的综述，包括性能分析的关键技术和现有的性能分析环境。接着，本文建立了一个完整的、实用的高性能微处理器分析环境，它包含一组工具，能够很好地满足各种性能分析的需求。

我们提出了一种用 C 语言对硬件建模的方法，高效地建立了信号级模拟 ICT-godson。ICT-godson 具有高度的准确性，能够和 RTL 逐拍验证，甚至可以为 RTL 生成测试向量。为了把 ICT-godson 用于性能分析，我们对它进行了速度优化和相关功能增强。优化之后后 ICT-godson 能够每秒模拟 50-100K 个处理器周期，能够有效地测量和表示处理器行为。

为了弥补 ICT-godson 在速度和灵活性上的不足，我们进一步设计了 Sim-godson 模拟器。我们采取多种技术加速 Sim-godson 模拟器，使它的速度达到每秒 500K 周期以上，和目前同等复杂度的最快的详细模拟器相当。我们还在 Sim-godson 中实现了两种的大程序快速评估方法，并对它们进行了加速比和误差评估。使用其中的 Simpoint 技术和我们所实现的检查点支持，我们能够在一个小时内准确地预测目标处理器运行 SPEC CPU2000 reference 数据集的性能。Sim-godson 结合了 Simplescalar 和 SIMOS 的优点，同时支持用户级模拟和全系统模拟。我们还介绍了 Sim-godson 的验证流程以及相关验证工作的经验。

我们讨论了 RTL 模型和 FPGA 在性能分析中的作用。我们为 RTL 模型和 ICT-godson、Sim-godson 设计了交叉验证的环境。我们考察了利用 FPGA 仿真环境进行性能分析的可行性，指出它能够有效地用于某些性能分析场合，特别是用于验证全系统模拟。

介绍完性能分析环境的工作之后，我们进一步讨论了性能分析方法。我们考察了三种瓶颈分析方法：参数敏感性分析、PB 设计和交互代价分析。这些方法可以帮助我们进行系统的性能分析。

最后，我们使用本文建立的性能分析环境和方法对龙芯 2 号处理器进行了全面的性能分析。我们给出了各种角度的处理器性能分析数据实例以及各种性能瓶颈分析方法的使用和分析。根据性能分析的结果我们提出和评估了一些有效的处理器优化方案。我们分析了 SPEC CPU2000 的一些特性，包括指令局部性和随时间变化的行为。我们讨论了编译器对性能的影响，指出 Gcc 编译器和商业编译器 MipsPro 存在较大的性能差距，并

从机器相关优化和链接级优化方面做了一些尝试，取得了良好的效果。我们还讨论了操作系统对性能的影响。我们比较了用户级模拟和全系统模拟的误差，并指出 SPEC CPU2000 系统态时间多数用于 TLB 例外处理。根据这个结论，我们对 TLB 进行了硬件和软件上的针对性优化，并取得了较好的效果。

8.2 下一步工作

高性能微处理器的性能分析和优化是一个非常艰巨的任务。本文的工作初步建立了一个完整的分析环境和分析方法，然而它仅仅是一个良好的开端。下一步工作中，我们需要从深度和广度上继续完善性能分析环境和分析方法。

我们提出了一种对硬件描述语言建模的方法，比较有效地建立了信号级模拟器 ICT-godson。信号级模拟器可以作为 RTL 实现的直接参考模型，用于及早发现 RTL 实现上的问题；也可以用于验证更抽象的性能模型。它的存在是非常必要的。但是，目前 ICT-godson 模拟器的灵活性是一个突出问题。ICT-godson 作为一种信号级的模拟器，模拟的细节很多，灵活性本身就有限。我们采取的各种加速方法虽然极大地提升了速度，却进一步降低了它的灵活性。因此在 RTL 模型建立之后，设计人员常常更乐于直接在 RTL 模型上做一些小的修改。下一步工作中我们将采取一些有效的方法来解决这个问题。我们将试图把目前建模和加速过程中某些容易出现问题的地方自动化。例如，手工安排模块调用次序是一个很容易出错的过程。我们可以开发一个工具，通过分析模块接口信号的依赖关系来自动生成最优的模块排序。加速过程中，消除拷贝以及冗余调用也有望通过代码分析来完成。事实上，我们正在尝试开发一个 RTL 到 C 的半自动翻译器，它可以利用我们提出的建模方法来直接把 RTL 模型转换为 C 语言模型，然后进行相应的加速工作。

龙芯 2 号性能分析环境中，Sim-godson 是一个比较重要的工具。Sim-godson 模拟器能够比较好地满足灵活性、速度和准确性的要求，可以有效地用于工作负载分析、设计空间探索、性能预测等场合。但是目前 Sim-godson 还有许多值得改进的地方，包括：

- 完善功耗支持。Sim-godson 集成了 WATTCH 功耗模拟支持，但是没有进行深入的分析和验证。我们可以充分利用龙芯 2 号 RTL 模型的功耗分析信息，建立一个能够利用这些反馈信息的功耗模型，实现一个比较准确的高层功耗分析工具。
- 进一步加快速度。模拟器的速度对研究工作有很大的影响。我们可以从几个方向继续提供 Sim-godson 的速度：一、继续改进 Sim-godson 模拟器的结构和算法；二、充分利用编译器优化，例如，修改程序风格帮助编译优化，使用更好的编译器（如 Intel C 编译器对模拟器所运行 P4 平台有一定的性能优势）或者编译选项、三、继续完善大程序快速模拟，包括 Simpoint 和 SMARTS 采样支持；要在实践工作中继续深入分析它们的实际表现，减小误差。对于 SMARTS，进一步加速主要是要提高功能模拟的速度，而提高功能模拟速度的一种方法是采用二进制翻译技术。

- 完善全系统模拟支持。这方面我们还有许多工作要做。首先是完善目标系统的各种模型，包括内存控制器等关键部件的性能模型和一些 I/O 设备模型。我们可以充分利用 FPGA 仿真环境对全系统模拟进行验证。我们还需要完善多处理器评估的机制，支持各种 CACHE 一致性协议的实现和评估、支持 CMP/SMT 等新技术评估。
- 设法降低验证难度。结合人工分析的经验，尽可能增加验证流程的自动化程度。完善微基准程序集，以及 Sim-godson 和 RTL 模型交叉验证的环境。

下一步工作中，我们将加强对 RTL 和 FPGA 仿真环境的建设，充分发挥它们在验证和准确性方面的优势。具体的工作包括：一、完善 RTL 和 ICT-godson 以及 Sim-godson 的验证环境。二、利用 FPGA 研究性能分析硬件支持方案、三、研究在 RTL 或者 FPGA 上使用大程序模拟加速技术。

本文提及了一些硅后优化工具，如 PERFCTR，ALTO 等。目前它们基本上是作为一种点工具使用，尚未完全融入整个性能分析的工作流程中。下一步工作中我们将继续完善这些工具。结合对性能分析硬件支持的研究，我们要开发一个面向一般用户的有效的性能分析工具。ALTO 的工作则可以进一步完善成一个成熟的工具，或者考虑将它集成到 gcc 编译器中。

系统的分析方法能够对性能分析工作起到指导作用。本文考察了三种性能瓶颈分析方法，并在实践中对它们进行了一些分析。但是，我们还需要更多实践以充分认识和改进各种性能分析方法。我们需要寻求一种系统化的手段来进行 PB 设计中的高、低值选择，或者使用另一些试验设计方法来解决 PB 设计的不足。例如，我们可以考虑使用均匀试验[方马 01]方法来替代 PB 设计。均匀试验方法不仅能够解决参数二值限制的问题，还具有许多优秀的统计特性，很可能比 PB 设计更适于用在处理器性能分析领域。我们要考察对实际处理器建立依赖图分析模型的可行性。最后，我们要把研究的结果融入性能分析的工作流程中。

本文工作建立在一个实际高性能处理器性能分析工作的基础之上，其研究目的是为了优化设计。结合处理器技术的发展，我们也将持续地改进性能分析环境和方法。

参考文献

[ABD97] J.M.Anderson, L.M.Berc, J.Dean etc. "Continuous Profiling: Where Have All the Cycles Gone?", Technical Note 1997-016. Digital Equipment Corporation Systems Research Center, July 1997.

[ALE02] T.Austin, E.Larson, D.Ernst. "SimpleScalar: An Infrastructure System Modeling", IEEE Computer, 35(2), Feb. 2002. pp.59-67.

[BCA99] P. Bose, T. M. Conte and T. M. Austin, “Challenges in processor modeling and validation,” IEEE Micro, vol. 19, no. 3, pp. 9-13, May/June 1999.

[BEG04] P. Bohrer, M. Elnozahy, A. Gheith, C. Lefurgy, T. Nakra, J. Peterson, R. Rajamony, R. Rockhold, H. Shafi, R. Simpson, E. Speight, K. Sudeep, E. V. Hensbergen, and L. Zhang. Mambo - a full system simulator for the PowerPC architecture. ACM SIGMETRICS Performance Evaluation Review, 31(4):8-12, Mar. 2004.

[BhD97] D.Bhandarkar, J.Ding. "Performance Characterization of the Pentium Pro Processor" Proceedings of the 3rd IEEE Symposium on High-Performance Computer Architecture, 1997.pp.288.

[BHL96] B.Black, A.S.Huan, M.H.Lipasti, J.P.Shen. "Can Trace-Driven Simulators Accurately Predict Superscalar Performance", Proceedings of International Conference on Computer Design, 1996. pp. 478-485.

[BKO97] P. Bose, S. Kim, F. O'Connell, W. A. Ciarfella. Bounds-Based Loop Performance Characterization: Application to Post-Silicon Analysis and Tuning.IBM Research Report RC21035, Nov. 1997

[BLS02] R. Rakvic, B. Black, D. Limaye, and J. P. Shen. Non-vital loads. In 8th International Symposium on High-PerformanceComputer Architecture, Feb 2002.

[BlS98] B.Black, J.P.Shen. "Calibration of Microprocessor Performance Models", IEEE Computer, 31(5), May 1998. pp. 59-65.

[BoC98] P.Bose, T.M.Conte. "Performance Analysis and Its Impact on Design", IEEE Computer, 31(5), May 1998. pp. 41-49.

[Bos00] P. Bose, “Testing for Function and Performance: Towards an Integrated Processor Validation Methodology,” Journ.of Electronic Testing: Theory and Applications (JETTA),vol. 16, pp. 29-48, 2000.

[Bos01] P.Bose. "Ensuring Dependable Processor Performance: An Experience Report on

Pre-silicon Performance Validation", Proceedings of the 2001 International Conference on Dependable Systems and Networks , 2001. pp. 481-486.

[Bos98] P. Bose, “Performance test case case generation for. microprocessors,” Proc.16th IEEE VLSI Test Symp. April. 1998, pp. 54-59

[Bos99] P.Bose. "Performance Evaluation and Validation of Microprocessor", Proceedings of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems 1999. pp. 226-227.

[BSC04] M. V. Biesbrouck, T. Sherwood, and B. Calder. A co-phase matrix to guide simultaneous multithreading simulation. In Proceedings of theInternational Symposium on the Performance Analysis of Systems and Software, June 2004.

[BTM00] D.Brooks, V.Tiwari, M.Martonosi. "Wattch: A Frame-work for Architectural-Level Power Analysis and Optimizations", Proceedings of the 27th annual international symposium on Computer architecture, 2000.pp.83-94.

[BuA97] D.Burger, T.M.Austin. "The SimpleScalar Tool Set", 1997. http:// www.simplescalar.com.

[CaG00] J. Casmira, D. Grunwald. Dynamic instruction scheduling slack. In Kool Chips Workshop in conjunction with MICRO 33, Dec 2000.

[CaH00] J. F. Cantin, M.D. Hill. Cache Performance for SPEC CPU2000 benchmarks. http://www.cs.wisc.edu/multifacet/misc/spec2000cache-data/.

[Cas98] R. Carl, J. Smith, "Modeling supersclar processors via statisical simulation," Performance Analysis and it's Impact on Design (PAID) Workshop, June 1998.

[CFK98] J.Casmira, J.Fraser, D.Kaeli, W.Meleis. "Operating System Impact on Trace-driven Simulation", the 31st Annual Simulation symposim, Apr. 1998.

[CHH97] T. M. Conte, M. A. Hirsch, WW Hwu: Combining Trace Sampling with Single Pass Methods for Efficient Cache Simulation. IEEE Trans. Computers 47（6).1998.pp. 714-720

[CHM96] T. M. Conte, M. A. Hirsch, and K. N. Menezes. Reducing state loss for effective trace sampling of superscalar processors. In Proceedings of the International Conference on Computer Design, Oct. 1996.

[ChP97] M. J. Charney, T. R. Puzak, “Prefetching and Memory System Behavior of the SPEC95 Benchmark Suite.” IBM. Journal of Research and Development, 41(3), May 1997.

[Cit03] D. Citron. MisSPECulation: partial and misleading use of SPEC CPU2000 in computer

architecture conferences, Proceedings of the 30th annual international symposium on Computer architecture,2003.pp52-61.

[CLS02] H. W. Cain, K. M. Lepak, B. A. Schwartz, and M. H.Lipasti. Precise and Accurate Processor Simulation. InProceedings of the Fifth Workshop on ComputerArchitecture Evaluation Using Commercial Workloads,pages 13-22, Feb. 2002.

[CmK94] R. F. Cmelik and D. Keppel. Shade: A Fast Instruction-Set Simulator for Execution Profiling. In Proceedings of the1994 ACM Sigmetrics Conference on Measurement andModeling of Computer Systems, May 1994.

[CWW76] Curnow, H. J. Wichmann, and B. Wichmann, “A synthetic benchmark,” The Computer Journal, 1976.

[DBK01] R.Desikan, D.Burger,S.W.Keckler. "Measuring Experimental Error in Microprocessor Simulation", Proceedings of the 28th annual international symposium on Computer architecture, 2001. pp.266-277.

[DBK02] R. Desikan, D. Burger, S.W. Keckler, L. Cruz, F. Latorre,A. Gonzalez, and M. Valero. Errata on measuring experimental error in microprocessor simulation. Computer Architecture News, March 2002.

[DeB01] R.Desikan, D.Burger. "Sim-alpha: a Validated, Execution-Driven Alpha 21264 Simulator". The University of Texas at Austin, Department of Computer Sciences. Technical Report TR-01-23. 2001.

[DHW97] J.Dean, J.E.Hicks, C.A.Waldspurger etc. "ProfileMe: Hardware Support for Instruction-Level Profiling on Out-of-Order Processors", Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture, 1997.pp.292-302.

[DJS94] S. Dwarkadas, J. R. Jump, and J. B. Sinclair. Execution-driven simulation of multiprocessors: Address and timing analysis. IEEE Transactions on Modeling and Computer Simulation, Volume 4, No. 4:314-338, Oct. 1994.

[DNS95] T.A.Diep, C.Nelson, J.P.Shen. "Performance Evaluation of the PowerPC 620 Microarchitecture", Proceedings of the 22nd annual international symposium on Computer architecture, 1995. pp. 163-174.

[DPA99] M. Durbhakula, V. S. Pai, and S. Adve. Improving the accuracy vs.speed tradeoff for simulating shared-memory multprocessors with ILP processors. In Proceedings of the International Symposium on High-Performance Computer Architecture, Jan. 1999.

[EAB02] J.Emer, P.Ahuja, E.Borch etc. "Asim: A Performance Model Framework", IEEE Computer, 35(2), Feb. 2002.pp.68-76.

[EJS04] L. Eeckhout, R. B. Jr., B. Stougie, K. D. Bosschere, and L. K. John. Control flow modeling in statistical simulation for accurate and efficient processor design studies. In Proceedings of the International Symposium on Computer Architecture, June 2004.

[ESD03] L.Eeckhout, D.Stroobandt,K.D.Bosschere. "Efficient Microprocessor Design Space Exploration through Statistical Simulation", Proceedings of the 36th annual symposium on Simulation, 2003. pp. 233.

[FBH02] B. Fields, R. Bodik, M. D. Hill. Slack: Maximizing performance under technological constraints. In 29th International Symposium on Computer Architecture, May 2002.

[FBH04] B. Fields, R. Bodik, M. D. Hill, C. J. Newburn.Interaction cost and shotgun profiling,ACM Trans. archit. Code Optim. 1(3), 2004. pp.272-304

[FiB99] B. R. Fisk, R. I. Bahar. The non-critical buffer: Using load latency tolerance to improve data cache effciency. Oct 1999.

[FRB01] B.Fields, S.Rubin, and R.Bodik. Focusing processor policies via critical-path prediction. In 28th International Symposium on Computer Architecture, Jun 2001.

[GHP93] J. D. Gee,M. D. Hill, D. N. Pnevmatikos, A. J. Smith. Cache Performance of the SPEC92 Benchmark Suite. Cache Performance of the SPEC92 Benchmark Suite, IEEE Micro. Aug. 1993. pp. 17-27.

[GKO00] J. Gibson, R. Kunz, D. Ofelt, M. Horowitz, J. Hennessy, M. Heinrich. Flash vs. (simulated) Flash: Closing the simulation loop. In Proceedings of the 9th International Symposium on Architectural Support for Programming Languages and Operating Systems, November 2000.

[GMC03] S.Girbal, G.Mouchard, A.Cohen, O.Temam. "Dist, A Simple,Reliable and Scalable Method to Significantly Reduce Processor Architecture Simulation Time",Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems.2003. pp.1-12.

[HaP02] A. Hartstein and T. R. Puzak. The optimum pipeline depth for a microprocessor. In 29th International Symposium on Computer Architecture, 2002.

[HaS01] J. W. Haskins and K. Skadron. Minimal subset evaluation: Rapid warm-up for simulated hardware state. In Proceedings of the International Conference on Computer Design, Sept. 2001.

[HaS03] J. W. Haskins and K. Skadron. Memory reference reuse latency: Accelerated warmup for sampled microarchitecture simulation. In Proceedings of the International Symposium on the Performance Analysis of Systems and Software, Mar. 2003.

[Hen00] J. L. Henning. SPEC CPU2000: Measuring CPU performance in the new millennium. Computer, 33(7):28-35, 2000.

[HeP02] J. L. Hennessy and D. A. Patterson. Computer Architecture:A Quantitative Approach. Morgan Kaufmann Publishers, Los Altos, CA, 3rd edition, 2002.

[HJF02] M. S. Hrishikesh, N. P. Jouppi, K. I. Farkas, D. Burger, S. W.Keckler, and P. Shivakumar. The optimal logic depth perpipeline stage is 6 to 8 FO4 inverter delays. In 29th International Symposium on Computer Architecture, 2002.

[HPC04] G.Hamerly, E. Perelman, B. Calder, How to Use SimPoint to Pick Simulation Points ACM SIGMETRICS Performance Evaluation Review, 2004.

[HPR02] C. Hughes, V. Pai, P. Ranganathan, S.Adve. "Rsim: Simulating Shared-Memory Multiprocessors with ILP Processors", IEEE Computer 35(2), 2002. pp.40-49.

[HSU01] G.Hinton, D.Sager, M.Upton etc. "The Microarchitecture of the Pentium 4 Processor". Intel Technology Journal Q1, 2001. in International Symposium on Microarchitecture MICRO27, ACM, November 1994.

[HZL05] Weiwu Hu, Fuxin Zhang, Zusong Li. "The Microarchitecture of Godson-2 processor", Journal of Computer Science And Technology, 20(2), Mar. 2005. pp. 243-249

[Int03-1] Intel Corporation. Intel Pentium 4 processor manual. In http://developer.intel.com/design/ pentium4/manuals/, 2003.

[Int03] Intel Corporation. Intel Itanium 2 processor reference manual for software development and optimization. Apr 2003.

[IntV] Intel VTune performance analyzers.http://www.intel.com/software/products/vtune.

[Jai01] R. K. Jain. The Art of Computer Systems Performance Analysis : Techniques for Experimental Design, Measurement, Simulation, and Modeling.Wiley, 2001.

[JEJ04] R. B. Jr., L. Eeckhout, and L. K. John. DeConstructing And Improving statistical simulation in HLS. In Proceedings of the Workshop on Duplicating, Deconstructing, and Debunking, June 2004.pp.2-12

[Jou89] N. P. Jouppi, "The nonuniform distribution of instruction-level and machine parallelism and its effect on performance," IEEE Transactions on Computers, December 1989.

[KaS02] G. B. Kandiraju, A. Sivasubramaniam. Characterizing the d-TLB behavior of SPEC CPU2000 benchmarks,Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems,2002.pp. 129-139.

[Kes99] R. E. Kessler. "The Alpha 21264 Microprocessor", IEEE Micro, 19(2), Mar. 1999. pp.24-36.

[KFM01] A. J. KleinOsowski, J. Flynn,N. Meares,D. J. Lilja. Adapting the SPEC 2000 benchmark suite for simulation-based computer architecture research. http://www.arctic.umn.edu/papers/spec2000-wwc.pdf.

[Kha00] H.Khalid. "Validating Trace-Driven MicroArchitectural Simulations", IEEE Micro, 20(6), Dec. 2000.pp.76-82.

[KHW91] R. E. Kessler, M. D. Hill, and D. A. Wood. A comparison of trace-sampling techniques for multi-megabyte caches. IEEE Transactions on Computers, 1991.

[KlL02] A.KleinOsowski, D.J.Lilja. "MinneSPEC: A New SPEC BenchmarkWorkload for Simulation-Based Computer Architecture Research". Computer Architecture Letters, Volume 1, June, 2002.

[KrT98] V. Krishnan and J. Torrellas. A direct-execution framework for fast and accurate simulation of superscalar processors. In Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, Oct. 1998.

[LaS00] T. Lafage and A. Seznec. Choosing representative slices of program execution for microarchitecture simulations: A preliminary application to the data stream. In IEEE Workshop on Workload Characterization, ICCD, Sept. 2000.

[Lau94] G. Lauterbach. Accelerating architectural simulation by parallel execution of trace samples. In Hawaii International Conference on System Sciences, Volume 1: Architecture, pages 205-210, Jan. 1994.

[LCA01] E.Larson, S.Chatterjee, T. Austin. "MASE: A Novel Infrastruture for Detailed Microarchitectural Modeling" Proc. IEEE Int'l Symp. Performance Analysis of Systems and Software , 2001.

[LiH04] W. Liu and M. Huang. Expert: Expedited simulation exploiting program behavior repetition. In Proceedings of the International Conference on Supercomputing, June 2004.

[LPI99] S. Laha, J. H. Patel, and R. K. Iyer. Accurate low-cost methods for performance evaluation of cache memory systems. IEEE Transactions on Computers, Volume C-37(11):1325-1336, Feb. 1988.

[MAA02] S.S.Mukherjee, S.V.Adve, T.Austin, J.Emer etc.. "Performance Simulation Tools", IEEE Computer, 35(2), Feb. 2002.pp38-39.

[Mat70] R. Mattson et al., “Evaluation Techniques for Storage Hierarchies,” IBM Systems J., Vol. 9, No. 2, 1970, pp.78-117

[McC99] J. McCalpin. Stream: Sustainable memory bandwidth in high performance computers.

http://www.cs.virginia.edu/stream/, 1999.

[MCE02] P.S. Magnusson, M.Christensson, J.Eskilison etc. Simics: A Full System Simulation Platform, IEEE Computer, 35(2), Feb. 2002. pp. 50-58. http://www.simics.net.

[McS03] C.McNairy, D.Soltis. "Itanium 2 Processor Microarchitecture", IEEE Micro, 23(2), Mar. 2003. pp. 44-55.

[MGA93] M.Martonosi, A.Gupta, T.Anderson. "Effectiveness of trace sampling for performance debugging tools", Proceedings of the ACM SIGMETRICS conference on Measurement and modeling of computer systems, 1993. pp. 248-259.

[MoB99] M.Moudgill, P.Bose. "Validation of Turandot, a Fast Processor Model for Micro－architecture Exploration"，Proc. IEEE Int'l Performance, Computing and Communication Conf., Feb. 1999, pp. 451-457.

[Mon91] D. Montgomery, “Design and Analysis of Experiments” (Third Edition), Wiley 1991.

[Mou98] M.Moudgill. "Techniques for Implementing Fast Processor Simulators", Proceedings of the The 31st Annual Simulation Symposium, 1998. pp. 83.

[MoW98] M.Moudgill, J.D.Wellman. "An approach for quantifying the impact of not simulating mispredicted paths", Workshop Digest of the PAID workshop held in conjunction with ISCA98.

[MoW99] M.Moudgill, J.D.Wellman. "Environment for PowerPC Microarchitecture Exploration", IEEE Micro, 19(3), 1999. pp. 15-25.

[MSB05] M. M. Martin, D.J. Sorin,B.M Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood. Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset,Submitted to Computer Architecture News (CAN). http://www.cs.wisc.edu/gems

[NMN97] A. Nguyen, M. Michael, A. Nanda, K. Ekanadham, and P. Bose. Accuracy and speed-up of parallel trace-driven architectural simulation. In Proc. Int’l Parallel Processing Symp., April 1997.pp.39-44

[NoS94] D. B. Noonburg, J. P. Shen, "Theoretical modeling of superscalar processor performance," in International Symposium on Microarchitecture MICRO27, ACM, November 1994.

[NoS97] D. B. Noonburg, J. P. Shen, "A framework for statistical modeling of superscalar processor performance," International Symposium on High-Performance Computer Architecture HPCA3, ACM, 1997.

[NuS01] S.Nussbaum, J.E.Smith. "Modeling superscalar processors via statisitcal simulation".

In International Conference on Parallel Architectures and Compilation Techniques (PACT), 2001.pp. 15-24.

[OCF00] M. Oskin, F. T. Chong, M. Farrens, "Hls: Cmobining statistical and symbolic execution to guide microprocessor design," in Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA'00), (Vancouver, Canada), 2000.

[Pai94] A.Paithankar. "AINT: A Tool for Simulation of Shared-Memory Multiprocessors", master's thesis, Univ. of Rochester,1993; revised August 1994.

[Pet02] Linux Performance-Monitoring Driver. http://user.it.uu.se/~mikpe/linux/ perfctr/.

[PHB03] Erez Perelman, Greg Hamerly, Michael Van Biesbrouck, Timothy Sherwood, and Brad Calder, Using SimPoint for Accurate and Efficient Simulation, , ACM SIGMETRICS the International Conference on Measurement and Modeling of Computer Systems, June 2003.

[PHC03] Erez Perelman, Greg Hamerly, and Brad Calder, Picking Statistically Valid and Early Simulation Points , International Conference on Parallel Architectures and Compilation Techniques, September 2003.

[PlB46] R. Plackett and J. Burman, “The Design of Optimum Multifactorial Experiments”, Biometrika, Vol. 33, Issue 4, June 1946, Pages 305-325.

[PRA97] V. S. Pai, P. Ranganathan, and S. V. Adve. The impact of instruction-level parallelism on multiprocessor performance and simulation methodology. In 3rd International Symposium on High Performance Computer Architecture, Feb 1997.

[PVA04] D. A. Penry, M. Vachharajani, and D. I. August. Rapid development of flexible validated processor models. Technical Report 04-03, Liberty Research Group, Princeton University, Nov. 2004.

[RBH95] M. Rosenblum, E. Bugnion, S. A. Herrod, E. Witchel, and A. Gupta. The impact of architectural trends on operating system performance. In 15th Symposium on Operating Systems Principles, Dec 1995.

[RDW01] R. Muth, S. Debray, S. Watterson, K. D. Bosschere. Alto: A Link-Time Optimizer for the Compaq Alpha. Software Practice and Experience 31:67-101, Jan. 2001.

[ReE98] M.Reilly, J.Edmondson. "Performance Simulation of an Alpha Microprocessor." IEEE Computer, 31(5), May 1998. pp. 50-58.

[REL00] J.A.Redstone, S.J.Eggers and H.M.Levy. "An Analysis of Operating System Behavior on a Simultaneous Multithreaded Architecture". Proceedings of the 9th International

Conference on Architectural Support for Programming Languages and Operating Systems, November 2000. pp. 245-256.

[RGA98] P. Ranganathan, K. Gharachorloo, S. V. Adve, and L. A. Barroso. Performance of database workloads on shared-memory systems with out-of-order processors. Oct 1998.

[RHL93] S. K. Reinhardt, M. D. Hill, J. R. Larus, A. R. Lebeck, J. C. Lewis, and D. A. Wood. The Wisconsin Wind Tunnel: Virtual prototyping of parallel computers. In Proceedings of the International Conference on Measurement and Modeling of Computer Systems, May 1993.

[RoH95] M.Rosenblum, S.A.Herrod. "Complete Computer System Simulation:The SimOS Approach", IEEE Parallel & Distributed Technology: Systems & Technology, 3(4), Dec. 1995.

[SaC00] S. Sair,M. Chamey, "Memory Behavior of the SPEC2000 Benchmark Suite." IBM Thomas J. Waston Research Center Technical Report RC-21852, October 2000.

[SAM99] K. Skadron, P. Ahuja, M. Martonosi, and D. Clark, “Branch Prediction, Instruction-Window Size, and Cache Size: Performance Trade-Offs and Simulation Techniques”, IEEE Transactions on Computers, Vol. 48, No. 11, November 1999, Pages 1260-1281.

[SaS95] R.H. Saavedra, A.J. Smith. Measuring Cache and TLB Performance and Their Effect on Benchmark Run Times. IEEE Trans. on Computers, Vol. 44, No. 10, October 1995, pp. 1223-1235.

[ScL98] E. Schnarr and J. Larus. Fast out-of-order processor simulation using memoization. In Proceedings of the International Conference on ArchitecturalSupport for Programming Languages and Operating Systems,Oct. 1998.

[SGI] http://www.sgi.com.

[SHA02] R. Sasanka, C. J. Hughes, and S. V. Adve. Joint local and global hardware adaptations for energy. In 10th International Conferenceon Architectural Support for Programming Languages andOperating Systems, Oct 2002.

[ShC99] Timothy Sherwood and Brad Calder, Time Varying Behavior of Programs, UC San Diego Technical Report UCSD-CS99-630, August 1999.

[Simca] http://www.cs.umn.edu/Research/Agassiz/Tools/SIMCA/simca.html.

[Sin04] R.Singhal etc. "Performance Analysis and Validation of the Intel Pentium4 Processor on 90nm Technology". Intel Technology Journal, 8(1), Feb. 2004.

[SJW01] S. T. Srinivasan, R. Dz ching Ju, A. R. Lebeck, and C. Wilkerson. Locality vs. criticality. In 28th International Symposium on Computer Architecture, Jun 2001.

[SKL03] M.Sakamoto. A.Katsuno, A.Lnoue etc. "Microarchitecture and Performance Analysis of SPARC-V9 Microprocessor for Enterprise Server Systems", Proceedings of the The Ninth International Symposium on High-Performance Computer Architecture, 2003. pp. 141.

[SMA03] K.Skadron, M. Martonosi, D. August, M. D. Hill, D. J. Lilja and V. S. Pai.Challenges in Computer Architecture Evaluation, IEEE Computer, 36(8), Aug. 2003.pp. 30-36.

[SMB02] G. Semeraro, G. Magklis, R. Balasubramonian, D.H. Albonesi, S. Dwarkadas, and M.L. Scott. Energy-effcient processor design using multiple clock domains with dynamic voltage and frequency scaling. In 8th International Symposium on High-Performance Computer Architecture, Feb 2002.

[Smi91] M.D.Smith. "Tracing with pixie", Stanford Univ. Tech. Rep. CSL-TR-91-497. 1991.

[SoS97] A. Sodani and G. S. Sohi. Dynamic instruction reuse. In 24th International Symposium on Computer Architecture, 1997.

[SPC01] Timothy Sherwood, Erez Perelman and Brad Calder, Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications , International Conference on Parallel Architectures and Compilation Techniques, September 2001.

[SpC02] E. Sprangle and D. Carmean. Increasing processor performance by implementing deeper pipelines. In 29th International Symposium on Computer Architecture, 2002.

[SPEC04] Standard Performance Evaluation Corporation. SPEC CPU2004 submission requirements. http://www.spec.org/cpu2004/, 2004.

[SPH02] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In Proceedings of theTenth International Conference on Architectural Support for Programming Languages and Operating Systems, Oct. 2002.

[SPH03] Timothy Sherwood, Erez Perelman, Greg Hamerly, Suleyman Sair, and Brad Calder, Discovering and Exploiting Program Phases, IEEE Micro: Micro's Top Picks from Computer Architecture Conferences, December 2003.

[Spr02] B. Sprunt. Pentium 4 performance-monitoring features. IEEE Micro, Jul 2002.

[SrE94] A.Srivastava, A. Eustace. "ATOM: A System for Building Customized Program Analysis Tools," Proceedings of the ACM SIGPLAN 1994 conference on

Programming language design and implementation, 1994. pp. 196-205.

[SrL98] S. T. Srinivasan and A. R. Lebeck. Load latency tolerance in dynamically scheduled processors. In 31st International Symposium on Micro-architecture, Nov 1998.

[SSC03] Timothy Sherwood, Suleyman Sair, and Brad Calder, Phase Tracking and Prediction, 30th International Symposium on Computer Architecture, June 2003.

[SPEC] Standard Perfomrance Evaluation Corperative. http://www.spec.org/.

[STT01] J. S. Seng, E. S. Tune, and D. M. Tullsen. Reducing power with dynamic critical path information. In 34th International Symposium on Microarchitecture, Dec 2001.

[SVB03] B. D. Sutter, H. Vandierendonck, B. D. Bus, K. D. Bosschere. On The Side-Effects of Code Abstraction. Proceedings of the 2003 ACM SIGPLAN Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES`03). 2003. pp. 244-253

[TDF02] J.M.Tendler, J.S.Dodson, J.S.Fields, J.H.Le etc. "POWER4 system microarchitecture". IBM Journal of Research & Development. 46(1), Jan. 2002.

[TLT01] E. Tune, D. Liang, D. M. Tullsen, and B. Calder. Dynamic prediction of critical path instructions. In 7th International Symposium on High-Performance Computer Architecture, Jan 2001.

[TTC02] E. Tune, D. Tullsen, and B. Calder. Quantifying instruction criticality. In 11th International Conference on Parallel Architectures and Compilation Techniques, Sep 2002.

[VaB04] H. Vandierendonck, K. D. Bosschere. Eccentric and Fragile Benchmarks. Proceedings of the 2004 IEEE International Symposium on Performance Analysis of Systems And Software. 2004. pp. 2-11.

[VaV01] M.Vachharajani, N.Vachharajani. "Microarchitecture Exploration with Liberty". Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture. 2002. pp. 271-282.

[VeF94] J. E. Veenstra, R. J. Fowler. MINT: A Front End for E_cient Simulation of Shared-Memory Multiprocessors. In Proceedings of the Second International Workshop on Modeling. Analysis, and Simulation of Computer and Telecommunication Systems, January 1994.pp204-207.

[Wei94] R. P. Weicker, "Dhrystone: A synthetic systems programming benchmark," Comm. ACM, October 1994.

[WiR96] E. Witchel, M. Rosenblum. Embra: Fast and Flexible Machine Simulation. The

proceedings of ACM SIGMETRICS '96: Conference on Measurement and Modeling of Computer Systems, Philadelphia, 1996

[WLY02] Chulho Won, Ben Lee, Chansu Yu, Sangman Moh, Yong-Youn Kim, Kyoung Park. "Linux/SimOS - A Simulation Environment for Evaluating High-Speed Communication Systems".Proceedings of the 2002 International Conference on Parallel Processing, 2002. pp.193

[WWF03] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. SMARTS:Accelerating microarchitecture simulation via rigorous statistical sampling. In Proceedings of the International Symposium on Computer Architecture, June 2003.

[WWF04] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. An evaluation of stratified sampling of microarchitecture simulations. In Third Annual Workshop on Duplicating, Deconstructing, and Debunking, ISCA,June 2004.

[Yea96] K. Yeager. "The MIPS R10000 Superscalar Microprocessor",IEEE Micro, 16(2), Apri. 1996. pp.28-40.

[YLH03] J. J. Yi, D. J. Lilja, and D. M. Hawkins. A statistically rigorous approach for improving simulation methodology. In 9thInternational Symposium on High Performance Computer Architecture , Feb 2003.

[YLS05] J. J. Yi, D. J. Lilja, R. Sendag, S. V. Kodakara, and D. M. Hawkins.Characterizing and comparing prevailing simulation methodologies. In Proceedings of the International Symposium on High-Performance Computer Architecture, Feb. 2005.

[ZLS96] M.Zagha, B.Larson, S.Tuner, M.Ltzkowitz. "Performance Analysis Using the MIPS R10000 Performance Counters", Proceedings of the 1996 ACM/IEEE conference on Supercomputing. 1996.

中文参考文献

[刘张王 0 4] 刘玲，张福新，王剑。“基于profiling反馈信息的链接后优化技术在龙芯上的实现”，计算机工程，已录用。

[方马 02] 方开泰，马长兴。《正交与均匀试验设计》，科学出版社，2001。

致　谢

在本文的写作过程中，许多的人为我提供了各种帮助。我很高兴能在此对他们表示感谢。

首先，我要感谢我的家人，没有他们就没有我的今天。为了供养我上学，父母早早地累弯了腰，累白了头，姐姐初中没有毕业就去打工挣钱；而不论怎么样的逆境中，他们从来就没让我受一点委屈。我也要感谢我新婚的妻子，感谢她容忍我天天深夜回家，感谢她细心地照顾我的生活。

我要特别感谢我的导师胡伟武，他对我的帮助和影响是全方位的。胡老师以身作则，教我做人、做事、做学问。他教我学会多思考再动手，他教我读文章、写文章。他敏捷的思维、踏实的工作态度和敢于创新的精神对我影响至深。胡老师营造了一个非常难得的科研环境，使我能够有机会学习高性能微处理器设计，并最终选择了处理器性能分析和优化这一研究领域。此外，在生活上胡老师给了我许多实质的帮助，使我能够全心投入科研工作。

本文的工作很多地方都得到 CPU 组同事的帮助。胡伟武老师、李祖松同学和我一起设计了最初的 ICT-godson 模拟器，其中大量的设计思想出自胡老师。李祖松帮助我对 ICT-godson 进行了速度优化。设计在 SIMOS 模拟器相关工作中，林伟、马可、伍鸣、高翔等同学做了许多贡献，在 Sim-godson 设计中，黄昆、肖俊华、高翔和汤彦等同学也给予了各种帮助。汤彦在 simpoint 收集和分析，smarts 分析以及机群运行环境维护等方面为我提供了许多帮助。范宝霞帮我实现了 FPGA 的接口改造的想法，李晓钰同学则帮我进行 FPGA 代码综合。软件工具 ALTO 的开发则得到了刘玲和陈瑜等同学的帮助，文中引用的加速比数据来源于陈瑜。TLB 分析引用了李建松、林伟、伍鸣的一些数据。系统结构小组许多同事对龙芯 2 号的性能分析工作作出了贡献。胡伟武老师、唐志敏研究员、章隆兵副研究员审阅了本文的初稿并提出了宝贵意见。此外，本文的很多想法都来源于平时与同事们，尤其是系统结构小组的同事们的讨论，在此一并感谢。我也要特别感谢王剑副研究员。为了让我专心进行论文工作，他承担了许多额外工作。

我还要感谢研究室的秘书刘凤勤老师、杨欣等老师，以及所里研究生部和各后勤部门老师多年来对我的关心和照顾。

作者简历

姓名：张福新　　性别：男　　出生日期：1976.6.30　　籍贯：福建永定

2000.9 – 2005.7　　中科院计算所计算机系统机构专业硕博连读生
1995.9 – 2000.7　　中国科学技术大学计算机软件专业本科生

【攻读博士学位期间发表的论文】

- Microarchitecture of the Godson-2 processor. Weiwei Hu, Fuxin Zhang, Zusong Li. Journal of Computer Science and Technology, 2005.2.
- Dynamic Data Prefetching in Home-Based Software DSMs. Weiwu Hu, Fuxin Zhang, Haiming Liu. J. Comput. Sci. Technol. 16(3): 231-241 2001.
- Running Real Applications on Software DSMs. Weiwu Hu, Fuxin Zhang, Li Ren, Weisong Shi, Zhimin Tang. The Fourth International Conference on High-Performance Computing in the Asia-Pacific Region, Volume 1. 2000.
- A New Home-Based Software DSM Protocol for SMP Clusters. Weiwu Hu,Fuxin Zhang,Haiming Liu. 2000. Euro-Par, pp 1132-1142.
- 龙芯 2 号性能分析。张福新，李祖松，马可，林伟，章隆兵，胡伟武。《计算机研究与发展》，已录用。
- 可恢复的软件 DSM 系统 JIACKPT。张福新，章隆兵，胡伟武，唐志敏。《软件学报》，2005 年 2 月.
- 解咏梅，张珩，张福新，“基于覆盖率的功能验证方法”，计算机应用研究，Vol.22, No.1, pp. 23-24,28, 2005 年 1 月。
- 章隆兵，吴少刚，张福新，“软件 DSM 系统中的动态数据竞争检测”，小型微型计算机系统,25(12):2070-2074，2004 年 12 月。
- 基于锁集合的动态数据竞争检测方法。章隆兵，张福新，吴少刚，陈意云。2003《计算机学报》，2003 年 10 月。
- 胡明昌，史岗，胡伟武，唐志敏，张福新，“PC 机群上 JIAJIA 和 MPI 的比较”，软件学报，第 14 卷第 7 期，pp. 1187-1194，2003 年 7 月。
- 章隆兵，张福新，陈意云，“软件 DSM 系统的重放”，小型微型计算机系统，24(3):340-343，2003.3。
- 软件 DSM 机群上并行大规模地理图像处理系统 ParGIP。史 岗、张福新、胡伟武、韩承德。《计算机研究与发展》，2003 年 1 月。
- 庄泗华，王剑，张福新，“检查 linux 下的 VFS 型内核后门软件”，计算机应用研究，已录用。
- 张逸微，王剑，张福新，“基于滤波的动态向量的快速估计算法”，计算机应用研究，已录用。
- 刘玲，张福新，王剑，“基于 profiling 反馈信息的链接后优化技术中 profiling 在龙芯上的实现”,计算机工程，已录用。

【攻读博士学位期间参加的科研项目】

【攻读博士学位期间的获奖情况】

[1] 2003 中科院计算所“所长奖学金优秀奖”
[2] 2004 中科院计算所“所长奖学金特别奖”
[3]2004 年中国科学院杰出科技成就奖