

Solaris

应用程序设计

Solaris Application Programming



(英) Darryl Gove 著

张伟 刘子锐 董峻峰 等译

```
public void init() { . . . }  
.  
.  
public static void main(String args[])  
{  
    AppletFrame frame = new AppletFrame(new MyAppletApplica  
());  
    frame.setTitle("MyAppletApplication");  
    frame.setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);  
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOS  
    frame.setVisible(true);  
}  
ml version="1.0" encoding="UTF-8"?>  
CTYPE preferences SYSTEM "http://java.sun.com/dtd/preferen  
>  
erences EXTERNAL_XML_VERSION="1.0">  
ot type="user">  
>  
node name="com">  
map/>  
node name="horstmann">  
map/>  
<node name="corejava">  
<map>  
<entry key="left" value="11"/>
```



机械工业出版社
China Machine Press

Sun 公司核心技术丛书

TP316.81/167

2008

Solaris

应用程序设计

Solaris Application Programming

(英) Darryl Gove 著

张伟 刘子锐 董峻峰 等译

```
public void init() { . . . }

public static void main(String args[])

AppletFrame frame = new AppletFrame(new MyAppletApp

frame.setTitle("MyAppletApplication");
frame.setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_C
frame.setVisible(true);

version="1.0" encoding="UTF-8"?>
preferences SYSTEM "http://java.sun.com/dtd/pref

EXTERNAL_XML_VERSION="1.0">
type="user">

name="com">

name="horstmann">

me="corejava">

ey="left" value="11"/>
```



机械工业出版社
China Machine Press

本书介绍如何在 Solaris 操作系统上获得程序的最佳运行性能。全书共分五部分 13 章,包括常规处理器、SPARC、x64 处理器、信息工具、编译器、浮点数优化、库与链接、性能分析工具、校正与调试、性能计数器度量、源代码优化、多核多进程多线程以及性能分析。实用性是本书的最大特点,全书覆盖了目前所有在 Solaris 上可以使用的开发工具,并介绍了它们的用法,而且书中还提供了大量实例工具的用法并解释其输出数据的含义。

本书内容丰富,层次分明,不仅适合软件开发新手阅读,也可供有一定经验的开发者参考。

Simplified Chinese edition copyright © 2008 by Pearson Education Asia Limited and China Machine Press.

Original English language title: *Solaris Application Programming* (ISBN 978-0-13-813455-6) by Darryl Gove, Copyright © 2008 Sun Microsystems, Inc.

All rights reserved.

Published by arrangement with the original publisher, Pearson Education, Inc., publishing as Sun Microsystems, Inc..

本书封面贴有 Pearson Education(培生教育出版集团)激光防伪标签,无标签者不得销售。

版权所有,侵权必究。

本书法律顾问 北京市展达律师事务所

本书版权登记号:图字:01-2008-2136

图书在版编目(CIP)数据

Solaris 应用程序设计/(英)格夫(Gove, D.)著;张伟,刘子锐,董峻峰译. —北京:机械工业出版社,2008.5

ISBN 978-7-111-23878-2

I. S… II. ①格… ②张… ③刘… ④董… III. 操作系统(软件), Solaris - 程序设计 IV. TP316.89

中国版本图书馆 CIP 数据核字(2008)第 047390 号

机械工业出版社(北京市西城区百万庄大街 22 号 邮政编码 100037)

责任编辑:李南丰

北京牛山世兴印刷厂印刷 · 新华书店北京发行所发行

2008 年 5 月第 1 版第 1 次印刷

186mm × 240mm · 21 印张

标准书号:ISBN 978-7-111-23878-2

定价:49.00 元

凡购本书,如有倒页、脱页、缺页,由本社发行部调换
本社购书热线(010)68326294

译者序

由 Prentice Hall 出版的《Solaris™ 应用程序设计》(作者 Darryl Gove) 是 Solaris 系列丛书中的一本。

本书的多语言翻译工作正在进行中, 包括荷兰语、法语、德语、印度语、日语以及韩语等。当然, 本书的中文版也即将与广大读者见面。

此书之所以受到如此的追捧, 主要原因是 Solaris 操作系统正在被越来越多的用户所接受。随着 Solaris 操作系统的开源, Open Solaris 依靠其优异的性能和稳定的特性在全球迅速升温, 官方网站的下载量节节攀升, 各种有关 Open Solaris 的社区如雨后春笋般涌现出来, Solaris 操作系统越来越受到人们的青睐。用户在享受 Solaris 操作系统带来的种种好处的同时, 还热衷于为 Solaris 的发展贡献自己的力量: 积极参与社区活动, 为扩展 Solaris 的功能贡献源代码。

但问题也随之而来: 如何在 Solaris 上编写出性能出众而且稳定的应用程序呢? 本书的推出恰逢其时地解决了这个棘手问题。引用一位读者的话, “这本书对 Solaris 平台上的编程技术做了详细的总结和分类, 并告诉读者如何使用这些技术。我非常喜欢这本书并把它作为最新的 Solaris 参考手册, 如果你对应用程序性能十分挑剔, 是性能至上的狂热份子, 那么这将是一本必备图书。”

由此可见, 本书的适用读者对象十分广泛。无论是对应用程序性能分析一无所知的初学者, 还是性能分析的专家, 本书都同样具有指导和参考意义。即使在非 Solaris 平台上优化应用程序性能, 本书的许多概念和方法仍然可以被借鉴和使用。

本书的翻译团队成员全部来自 Sun 中国工程研究院, 包括: 董峻峰、段少婷、焦梦葳、胡惠文、李志宏、刘子锐、吕华锋、孙勇、王秀燕、王宇博、许闽、薛东、薛晓舟、颜强、姚延栋、于继军、张伟、张扬眉、张羽和郑艺丁。为了使本书以最快的速度呈现在广大中文读者面前, 翻译团队的每位成员都付出了艰苦的劳动。借此机会也要特别感谢翻译项目的发起方 Sun 中国技术社区和发起人蒋清野先生。

由于本书的翻译进度要求高, 涉及技术面广, 翻译难度较大; 加之译者水平有限, 书中难免有疏漏或错误, 欢迎广大读者批评指正。

张 伟

2008 年 4 月

前 言

关于本书

本书旨在介绍如何在 Solaris 操作系统上获得程序的最佳运行性能。本书适用的读者是那些对 Solaris 上的可用工具感兴趣的开发者和软件架构师，以及那些关心如何充分利用系统中每一种性能的人员。

本书不仅适合性能分析和优化的新手阅读，也适合在这个领域有一定经验的开发者阅读。为了满足不同层次读者的需求，本书在介绍各种工具并进行深入讨论之前，首先综述了处理器的基本概念。

与其他同类书籍相比，本书是一本实用性很强的入门书籍。人们在开发过程中经常会遇到两类问题。第一类问题是应该使用什么工具。针对这一问题，本书覆盖了目前所有在 Solaris 上可用的开发工具，并且介绍了它们的常见用法。第二类问题是如何理解这些工具的输出。书中提供了大量的例子，来演示工具的用法并解释其输出数据的含义。

对于编译器可以自动执行的优化技术，本书将避免解释手动启用它的方法。本书着重关注于如何利用适当的工具来定位问题，以及如何利用最简单的方法来解决问题。有时候，解决的方法是使用不同的编译选项优化应用程序，从而消除导致问题产生的热点代码。而有时候，解决的方法是修改编译器无法执行优化的代码；我会详细地解释编译器无法优化这些代码的原因。

目标和前提

本书要实现的目标如下：

- 全面地介绍影响处理器性能的部件。
- 介绍用于性能分析和性能提高的工具，包括操作系统所提供的工具和编译器所提供的工具。
- 介绍编译器以及编译器支持的提高性能的优化技术。
- 讨论 SPARC 和 x64 处理器家族的特征，并示范如何利用这些特征来提高应用程序的性能。
- 介绍利用多处理器或多线程来提高性能的可能性，以及有效地利用计算机资源的可能性。

本书假定读者可以熟练地使用 C 编程语言。书中的大多数例子都采用了这种语言。本书还假定读者了解一些处理器的基本原理和指令集的相关知识。本书不会详细地阐述处理器的内部细节，但是会介绍一些现代处理器的特征，这些特征可能影响应用程序的性能。

本书假定读者有 Sun Studio 编译器和相关的工具。这些工具都是可以免费下载的。本书

使用 Sun Studio 12 编译大多数例子。新版本的编译器将产生相似的结果。编译器通常安装在 /opt/SUNWspro/bin/ 目录下，并且假定该路径已经设置在读者的环境变量中。

本书是基于 Solaris 10 的。在此之前的操作系统支持书中涉及的大多数工具。如果某个工具需要更新版本的 Solaris 的支持，那么我将在书中给出说明。

本书章节综述

第一部分 处理器综述

- 第 1 章：常规的处理器的
- 第 2 章：SPARC 家族
- 第 3 章：x64 处理器家族

第二部分 开发工具

- 第 4 章：信息工具
- 第 5 章：编译器的使用
- 第 6 章：浮点数优化
- 第 7 章：库与链接
- 第 8 章：性能分析工具
- 第 9 章：校正与调试

第三部分 优化技术

- 第 10 章：性能计数器度量
- 第 11 章：源代码优化

第四部分 线程化和吞吐量

- 第 12 章：多核、多进程与多线程

第五部分 总述

- 第 13 章：性能分析

致谢

在本书的编写过程中，很多人都做出了贡献。Ross Towle 提供了多线程编程的章节大纲，并且对这章的内容做出了有价值的评论。Joel Williamson 多次阅读了早期的草稿，每次都给出了详细的评论和改进方法。我的同事 Boris Ivanovski、Karsten Gutheridge、John Henning、Miriam Blatt、Linda Hsi、Peter Farkas、Greg Price 和 Geetha Vallabhenini 也阅读了草稿，并提出了修改意见。在此，我要特别感谢 John Henning 对本书做了许多细致的改进。

我还要感谢所有花时间阅读本书并提供有效建议的各个领域的专家，他们是链接方面的 Rod Evans，调试器方面的 Chris Quenelle，提供了一些有用工具名字和评论的 Brian Whitney，提供建议的 Brendan Gregg，对编译器和源代码优化技术部分进行审阅的 Jian-Zhong Wang，提供关于浮点优化技术详细建议的 Alex Liu，对性能分析和多线程章节进行评论的 Marty Izkowitz，对多线程相关章节提出建议的 Yuan Lin、Ruud van der Pas、Alfred Huang 和 Nawal Copty，

对 MPI 进行评论的 Josh Simmons, 了解 SPARC 处理器历史的 David Weaver, 对 x64 处理器相关章节进行审阅的 Richard Smith, 以及对本书提供总体建议的 Richard Friedman。

在本书的出版过程中，很多人都提供了帮助，他们是 Yvonne Prefontaine、Ahmed Zandi 和 Ken Tracton。尤其感谢 Richard McDougall 在项目最后阶段提供的帮助。

特别感谢 Prentice Hall 的工作人员，包括编辑 Greg Doench 和产品经理 Julie Nahil。还要感谢 Techné 小组的产品项目经理 Dmitri Korzh。

最重要的是，感谢我的家人对我的支持和鼓励。Jenny 的冷静以及有效的建议帮助我完成了书中困难部分的编写；Aaron 即使对最现实的问题也抱有丰富的想象力，他成功地感染了我；Timothy 对分享生活中的乐趣有很大的热情，这对我的影响是无处不在的；Emma 出生在我完成本书的时候，她是上天赐给我的最珍贵的礼物。

目 录

译者序 前言

第一部分 处理器综述

第1章 常规的处理器的	1
1.1 本章目标	1
1.2 处理器的组成	1
1.3 时钟速率	2
1.4 乱序执行处理器	2
1.5 芯片多线程	3
1.6 执行管道	3
1.6.1 指令时延	4
1.6.2 装入/存储管道	4
1.6.3 整型操作管道	4
1.6.4 分支管道	5
1.6.5 浮点管道	5
1.7 高速缓存	6
1.8 系统交互	8
1.8.1 带宽与时延	8
1.8.2 系统总线	8
1.9 虚拟内存	9
1.9.1 概述	9
1.9.2 TLB 和页面大小	10
1.10 内存的索引和标记	10
1.11 指令集架构	11
第2章 SPARC 家族	12
2.1 本章目标	12
2.2 UltraSPARC 家族	12
2.2.1 SPARC 体系结构的历史	12
2.2.2 UltraSPARC 处理器	12
2.3 SPARC 指令集	13
2.3.1 SPARC 指令集简介	13
2.3.2 整数寄存器	15

录

2.3.3 寄存器窗	16
2.3.4 浮点寄存器	17
2.4 32 位和 64 位代码	18
2.5 UltraSPARC III 系列处理器	18
2.5.1 CPU 的核心	18
2.5.2 与内存的通信	18
2.5.3 预取	18
2.5.4 数据高速缓存不命中时装入操作 的停顿	21
2.5.5 基于 UltraSPARC III 的系统	21
2.5.6 全存储顺序	22
2.6 UltraSPARC T1	22
2.7 UltraSPARC T2	23
2.8 SPARC64 VI	23
第3章 x64 处理器家族	24
3.1 本章目标	24
3.2 x64 处理器家族	24
3.3 x86 处理器: CISC 和 RISC	25
3.4 字节顺序	25
3.5 处理器指令格式	26
3.6 寄存器	27
3.7 指令集扩展与浮点计算	29
3.8 内存操作顺序	29

第二部分 开发工具

第4章 信息工具	31
4.1 本章目标	31
4.2 报告系统配置的工具	31
4.2.1 简介	31
4.2.2 报告一般系统信息	31
4.2.3 启用虚拟处理器	32
4.2.4 通过处理器的集合或者绑定来 控制处理器的使用	33
4.2.5 报告硬件支持的指令集	33
4.2.6 报告硬件支持的 TLB	33

页面大小	34	4.4.12 用 dtrace 探究用户代码和 内核活动	54
4.2.7 报告 SPARC 硬件特性摘要	35	4.5 应用相关信息	56
4.3 报告当前系统状态的工具	35	4.5.1 报告库链接(ldd)	56
4.3.1 简介	35	4.5.2 报告文件内容的类型(file)	57
4.3.2 报告虚拟内存的利用 情况(vmstat)	35	4.5.3 报告文件中的符号(nm)	58
4.3.3 报告交换文件使用 情况(swap)	37	4.5.4 报告库的版本信息(pvs)	58
4.3.4 报告进程资源利用 情况(prstat)	37	4.5.5 检查应用、库或者目标文件的 反汇编(dis)	59
4.3.5 列出进程(ps)	39	4.5.6 报告应用、库或者目标文件中 各种段的大小(size)	59
4.3.6 定位应用的进程 ID(pgrep)	39	4.5.7 报告文件中的元数据(dumpstabs、 dwarfdump、elfdump、dump 和 mcs)	60
4.3.7 报告所有处理器的 活动(mpstat)	40	第 5 章 编译器的使用	62
4.3.8 报告内核统计(kstat)	41	5.1 本章目标	62
4.3.9 生成系统活动报告(sar)	42	5.2 三类编译选项	62
4.3.10 报告 I/O 活动(iostat)	44	5.3 在 x86 平台上使用 -xtarget = generic	63
4.3.11 报告网络活动(netstat)	45	5.4 优化	64
4.3.12 snoop 命令	46	5.4.1 优化级别	64
4.3.13 报告硬盘空间利用情况(df)	46	5.4.2 -O 选项的使用	65
4.3.14 报告文件占用的硬盘 空间(du)	47	5.4.3 -fast 选项的使用	65
4.4 进程和处理器的专有工具	47	5.4.4 在使用 -fast 的同时指定 体系结构	65
4.4.1 简介	47	5.4.5 解构 -fast	66
4.4.2 进程执行时间(time、timex 和 ptime)	47	5.4.6 -fast 对性能的优化(针对 Sun Studio 12)	67
4.4.3 报告系统级硬件计数器的 活动(cpustat)	48	5.5 生成调试信息	67
4.4.4 为单个进程报告硬件性能计数器 的活动(cputrack)	49	5.5.1 调试信息选项	67
4.4.5 报告总线活动(busstat)	50	5.5.2 调试和优化	68
4.4.6 报告陷阱活动(trapstat)	50	5.6 为应用程序选择目标机器的类型	68
4.4.7 报告进程的虚拟内存映射 信息(pmap)	51	5.6.1 在 32 位和 64 位之间选择	68
4.4.8 检测传递给进程的命令行 参数(pargs)	52	5.6.2 目标机器的通用类型 (generic)	69
4.4.9 报告进程打开的文件(pfiles)	52	5.6.3 使用 -xcache 指定缓存配置	69
4.4.10 检查进程的当前栈(pstack)	52	5.6.4 使用 -xchip 来设置指令调度	70
4.4.11 跟踪应用执行(truss)	53	5.6.5 -xarch 和 -m32/-m64	70
		5.7 代码布局优化	71

5.7.1 简介	71	5.10 其他针对 C 和 C++ 的优化	89
5.7.2 文件间优化	72	5.11 针对 Fortran 程序的优化	90
5.7.3 映射文件	73	5.11.1 对齐变量来优化布局	90
5.7.4 程序剖析反馈	73	(-xpad)	90
5.7.5 链接时优化	76	5.11.2 用栈来存放局部变量	90
5.8 通用的编译器优化	77	(-xstackvar)	90
5.8.1 预取指令	77	5.12 编译器指示词 (pragmas)	91
5.8.2 启用预取指令的生成	78	5.12.1 简介	91
(-xprefetch)	78	5.12.2 指明变量的对齐位数	91
5.8.3 控制激进的预取	79	5.12.3 指明函数访问全局变量的情况	91
(-xprefetch_level)	79	5.12.4 指明函数没有副作用	92
5.8.4 启用依赖分析 (-xdepend)	79	5.12.5 指明函数很少被调用	93
5.8.5 处理 SPARC 上非对齐的存取操作	80	5.12.6 指明针对特定循环流水线的 安全级别	94
(-xmemalign/-dalign)	80	5.12.7 指明循环的单次迭代内没有 访存依赖	95
5.8.6 使用 -xpagesize = <size> 设置页大小	81	5.12.8 指明循环展开的程度	95
5.9 C 和 C++ 的指针别名	82	5.13 针对 C 程序更好地控制别名的 指示词	95
5.9.1 关于指针的问题	82	5.13.1 断言变量间别名的程度	96
5.9.2 诊断别名问题	84	5.13.2 断言变量是别名的	97
5.9.3 使用受限指针来减少 C 和 C++ 程序中的别名问题	84	5.13.3 断言非指针变量是别名的	98
5.9.4 使用 -xalias_level 来指定指针 别名的级别	85	5.13.4 断言变量不会别名	98
5.9.5 C 的选项 -xalias_level	85	5.13.5 断言非指针变量不会别名	99
5.9.6 C 的选项 -xalias_level = any	85	5.14 与 GCC 的兼容	99
5.9.7 C 的选项 -xalias_level = basic	86	第 6 章 浮点数优化	100
5.9.8 C 的选项 -xalias_level = weak	86	6.1 本章目标	100
5.9.9 C 的选项 -xalias_level = layout	88	6.2 浮点数优化标记	100
5.9.10 C 的选项 -xalias_level = strict	88	6.2.1 标记 -fast 中的运算优化	100
5.9.11 C 的选项 -xalias_level = std	88	6.2.2 IEEE-754 与浮点运算	100
5.9.12 C 的选项 -xalias_level = strong	88	6.2.3 矢量化浮点运算	101
5.9.13 C++ 的选项 -xalias_level	89	(-xvector)	101
5.9.14 C++ 的选项 -xalias_level = simple	89	6.2.4 使用 SIMD 指令进行矢量运算	102
5.9.15 C++ 的选项 -xalias_level = compatible	89	(-xvector = simd) (仅适用于 x64 平台)	102
		6.2.5 次正规数	103
		6.2.6 将次正规数清为 0 (-fns)	104
		6.2.7 处理非数字的值	104

6.2.8 启用浮点表达式的简化 (-fsimple)	105	7.2.11 使用调试接口	130
6.2.9 消除比较	106	7.2.12 使用审计接口 (Audit Interface)	131
6.2.10 消除不必要的计算	106	7.3 其他一些有趣的库	132
6.2.11 重排运算顺序	107	7.3.1 C 运行时库 (libc 和 libc_psr)	132
6.2.12 Kahan 方程	109	7.3.2 内存管理库	132
6.2.13 提升除法	110	7.3.3 libfast	134
6.2.14 不同浮点简化级别对括号的 遵从情况	111	7.3.4 Performance 程序库	135
6.2.15 使用 -fast 对 error 的影响	112	7.3.5 STLport4	136
6.2.16 指定浮点消息的触发自陷 (-ftrap)	112	7.4 程序库调用	136
6.2.17 浮点异常标记	113	7.4.1 用于计时的程序库例程	136
6.2.18 C99 中的浮点异常	114	7.4.2 选择最适合的程序库例程	138
6.2.19 使用内联浮点函数模板 (-xlibmil)	115	7.4.3 SIMD 指令以及媒体库	139
6.2.20 使用优化的数学库 (-xlibmopt)	115	7.4.4 使用 VIS 指令搜索数组	139
6.2.21 不要把单精度提升为双精度 (C 的 -fsingle)	116	第 8 章 性能分析工具	143
6.2.22 使用单精度存储浮点常数 (适用于 C 的 -xsfpcnst)	116	8.1 简介	143
6.3 浮点数乘法加速指令	117	8.2 Sun Studio 性能分析器	143
6.4 整数数学	118	8.3 收集分析器	144
6.5 使用 SPARC V8 代码传递浮点 参数	121	8.4 为性能分析器编译应用程序	145
第 7 章 库与链接	123	8.5 使用 GUI 查看性能分析数据	145
7.1 简介	123	8.6 Caller-Callee 信息	147
7.2 链接	123	8.7 使用命令行工具进行性能分析	148
7.2.1 链接的概述	123	8.8 分析器详解	149
7.2.2 动态和静态链接	123	8.9 UltraSPARC III/IV 处理器的 分析器详解	150
7.2.3 链接程序库	124	8.10 使用性能计数器的分析	151
7.2.4 创建一个静态库	124	8.11 调用栈详解	152
7.2.5 创建一个动态库	125	8.12 生成映射文件	154
7.2.6 指定库的位置	126	8.13 使用 spot 工具生成性能报告	155
7.2.7 库的延时加载	127	8.14 内存访问模式分析器	157
7.2.8 程序库的初始化和终止代码	127	8.15 er_kernel	163
7.2.9 符号作用域	128	8.16 尾部调用 (Tail-Call) 优化和 调试	164
7.2.10 程序库间入	129	8.17 使用 gprof 收集分析信息	165
		8.18 用 tcov 得到代码覆盖信息	167
		8.19 用 dtrace 去收集分析数据和 覆盖信息	169
		8.20 编译器注解	171

第9章 校正与调试	173	10.3.1 指令与时钟周期	196
9.1 简介	173	10.3.2 数据高速缓存事件	198
9.2 编译时间检查	173	10.3.3 指令缓存事件	199
9.2.1 简介	173	10.3.4 二级高速缓存事件	200
9.2.2 C 语言程序的编译时间检查	173	10.3.5 高速缓存未命中事件所耗费的 时钟周期	200
9.2.3 使用 lint 检查 C 程序代码	173	10.3.6 高速缓存访问度量示例	201
9.2.4 C 和 C++ 编译器中常用的源程序 处理选项	175	10.3.7 延时的综合度量	203
9.2.5 C++	176	10.3.8 内存带宽消耗综合度量	204
9.2.6 Fortran	177	10.3.9 预读高速缓存事件	205
9.3 运行时检查	178	10.3.10 性能计数器预读和未预读的 比较	207
9.3.1 边界检查	178	10.3.11 写高速缓存事件	208
9.3.2 watchmalloc	180	10.3.12 处理器阻塞事件所消耗的 时钟周期	209
9.3.3 其他 mallocs 下的调试选项	180	10.3.13 分支预测失误	210
9.3.4 Fortran 中运行时数组 边界检查	181	10.3.14 内存控制器事件	210
9.3.5 运行时堆栈溢出检查	181	10.4 UltraSPARC IV 和 UltraSPARC IV + 上的性能计数器	211
9.3.6 使用 discover 检测内存 存取错误	182	10.4.1 简介	211
9.4 使用 dbx 调试程序	183	10.4.2 UltraSPARC IV + 上的三级 高速缓存	211
9.4.1 调试编译器标识位	183	10.4.3 内存控制器事件	212
9.4.2 调试和优化	183	10.5 UltraSPARC T1 上的性能计数器	212
9.4.3 调试信息格式	184	10.5.1 硬件性能计数器	212
9.4.4 调试和 OpenMP	184	10.5.2 UltraSPARC T1 时钟 周期预算	214
9.4.5 x86 上的帧指针优化	184	10.5.3 内核的性能计数器	215
9.4.6 在信息转储文件上运行 调试程序	185	10.5.4 计算系统带宽消耗	215
9.4.7 调试应用程序的例子	185	10.6 UltraSPARC T2 性能计数器	216
9.4.8 在 dbx 下运行一个应用 程序	187	10.7 SPARC64VI 性能计数器	216
9.5 使用 ATS 定位优化缺陷	189	10.8 Opteron 性能计数器	217
9.6 使用 mdb 调试	191	10.8.1 简介	217
 第三部分 优化技术		10.8.2 指令	218
第10章 性能计数器度量	195	10.8.3 指令高速缓存事件	218
10.1 本章目标	195	10.8.4 数据高速缓存事件	219
10.2 读取性能计数器	195	10.8.5 TLB 事件	220
10.3 UltraSPARC III 和 UltraSPARC IV 性能计数器	196	10.8.6 分支事件	221
		10.8.7 阻塞所消耗的时钟周期	221

第 11 章 源代码优化	223
11.1 概述	223
11.2 传统优化方法	223
11.2.1 简介	223
11.2.2 循环展开和流水线作业	223
11.2.3 循环剥离、融合及分割	224
11.2.4 循环交换和分块	225
11.2.5 循环不变量提升	227
11.2.6 公共子表达式消除	227
11.2.7 强度削弱	227
11.2.8 函数克隆	228
11.3 数据的局部性、带宽以及延时	228
11.3.1 带宽	228
11.3.2 整数数据	229
11.3.3 存储流	230
11.3.4 手动预取	231
11.3.5 延时	233
11.3.6 拷贝和移动内存	237
11.4 数据结构	238
11.4.1 结构重组	238
11.4.2 结构体预取	241
11.4.3 对结构体优化性能的考虑	244
11.4.4 矩阵及其访问	244
11.4.5 多个流	246
11.5 抖动	246
11.5.1 概述	246
11.5.2 数据 TLB 性能计数器	248
11.6 写后读	249
11.7 存储队列	251
11.7.1 停顿	251
11.7.2 检测存储队列停顿	251
11.8 If 语句	253
11.8.1 简介	253
11.8.2 条件移动	253
11.8.3 SPARC 处理器上的未对齐 内存访问	256
11.9 32 位应用程序中的文件处理	259
11.9.1 文件描述符的限制	259
11.9.2 在 32 位应用程序中	259

处理大文件	260
-------------	-----

第四部分 线程化和吞吐量

第 12 章 多核、多进程与多线程	263
12.1 简介	263
12.2 进程、线程、处理器、内核与 芯片多线程	263
12.3 虚拟化	265
12.4 横向以及纵向扩展	266
12.5 并行计算	266
12.6 用多进程扩展处理性能	267
12.6.1 多进程	267
12.6.2 多进程协同	268
12.6.3 使用 MPI 的并行计算	271
12.7 多线程应用程序	274
12.7.1 使用 Pthread 进行并行处理	274
12.7.2 线程局部存储	275
12.7.3 互斥锁	277
12.7.4 使用原子操作	281
12.7.5 伪共享	283
12.7.6 一个多线程程序的内存分布	285
12.8 使用 OpenMP 并行化应用程序	287
12.9 用 OpenMP 编译指令来并行化 循环代码	288
12.10 使用 OpenMP API	290
12.11 代码段并行化	290
12.12 应用程序的自动并行化	291
12.13 对多线程应用程序进行性能 分析	293
12.14 检测多线程应用程序中的数据 竞争	295
12.15 调试多线程代码	296
12.16 将一个顺序执行程序并行化	298
12.16.1 程序实例	298
12.16.2 优化对顺序执行程序的 性能影响	300
12.16.3 对顺序执行程序进行 性能分析	300

12.16.4	展开关键循环	301
12.16.5	用 Pthreads 实现并行化	303
12.16.6	使用 OpenMP 并行化	304
12.16.7	自动并行化	305
12.16.8	用 OpenMP 进行负载均衡 ...	308
12.16.9	线程间共享数据	308
12.16.10	使用 OpenMP 在线程之间 共享变量	310

第五部分 总 述

第 13 章	性能分析	313
13.1	简介	313
13.2	算法和复杂度	313
13.3	串行代码调优	316
13.4	并行性的探索	318
13.5	为 CMT 处理器进行优化	319

第一部分 处理器综述

第1章 常规的处理器

1.1 本章目标

简单来说,处理器的功能就是从内存中取出指令,并执行该指令,如果需要的话,还将从内存取出数据,或将结果送入内存。但是,这样的描述遗漏了决定应用程序性能的许多重要细节。本章描述的是“常规的”处理器;也就是说,本章所描述的是,通常情况下处理器是如何工作的以及它由哪些部件构成。阅读完本章,读者将会了解有关处理器的一些术语,并且会理解一些应用在处理器设计方面的方法。

1.2 处理器的组成

每一台计算机的“心脏”是一个或多个中央处理器(CPU)。图1-1是UltraSPARC T1 CPU的照片。CPU是计算机中执行计算的部件,组成计算机的其他部件包括内存芯片、硬盘、电源、风扇(用来冷却计算机),以及其他使得计算机可以与外界通信的芯片(如显卡芯片及网卡芯片)。CPU的底部有成百的“引脚”;^①如图所示,它们组成了交织状的纹样。每一个引脚都是CPU和系统之间的一个连接。

在CPU封装的内部是一个称为“内核”的小硅片。一个CPU包含一个或者多个用于计算的内核,本机上的或者是片上的一些存储器,这称为“高速缓存”(用于保存指令和数据),以及系统接口(使得处理器可以和系统的其他部件进行通信)。

有些处理器只有一个内核。而图1-1中的处理器UltraSPARC T1有8个内核,每一个内核都可以同时运行4个线程。对于这个系统的用户来说,这看起来就像有32个虚拟的处理器。

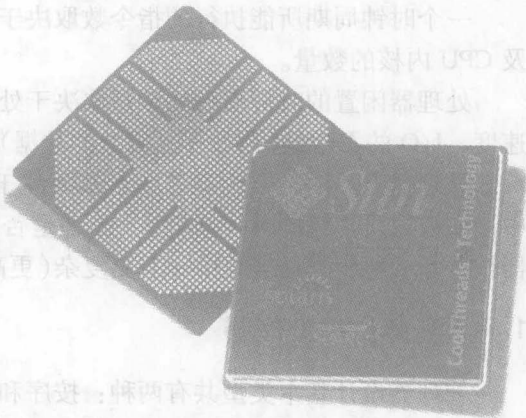


图1-1 UltraSPARC T1 处理器

① 过去,引脚设计成连接在处理器基底上真正的插针。这种封装方式的问题是,引脚很容易被弄弯或者弄坏。近年来,芯片封装技术把引脚设计成球状或者片状。

器。每一个虚拟处理器对操作系统来说就像是一个完整的处理器，都能执行一条指令流。图 1-2 所示的是 UltraSPARC T1 处理器的核心，该图标示了 CPU 每个区域所执行的功能。

1.3 时钟速率

所有的处理器都以某种特定的时钟速率来执行。时钟速率可以从 MHz 到 GHz^① 级别。时钟速率越高，通常意味着功耗越高。一条或者多条指令能够在一个时钟滴答内执行，所以，每秒被执行的指令数能够达到数百万条甚至数 10 亿条之多。每个时钟滴答称为一个“周期”。

时钟速率通常作为衡量一台处理器能力的显著特征，但时钟速率不足以衡量一台处理器的工作能力。

把时钟速率作为衡量标准常称为“兆赫神话”。处理器每秒能够处理的工作量取决于许多因素，时钟速率只是其中之一。其他因素还包括每个时钟周期处理器能够发出多少条指令，以及由于没有发出指令，而导致有多少个时钟周期被错失，这是一种令人惊讶的常见情况。处理器的性能还取决于处理器的设计以及当时它所负载的工作量。

一个时钟周期所能执行的指令数取决于可用的执行流水线的数量(将在第 1.6 节谈到)以及 CPU 内核的数量。

处理器闲置的时钟周期的数量取决于处理器的设计和所提供的高速缓存的容量、内存的速度、I/O 的吞吐量(比如写入磁盘的数据)以及具体的应用。

处理器设计的一个关键选择点常关注于高速缓存是否需要增加(这将会减少等待从内存中读取数据所消耗的时钟周期)，或者是否将这些核心空间用于其他的功能，如更多的处理器核，是否每个处理器核中具有更复杂(更高的性能)的电路。

1.4 乱序执行处理器

处理器设计基本类型共有两种：按序和乱序执行处理器。乱序执行处理器一般在给定的时钟速率下能够提供更高的性能，但也更复杂并且消耗更多的能量。

在一个按序执行的处理器中，指令按照它所出现的顺序执行，如果前一条指令的结果还不可用，处理器将会等待(或说“停滞”)直到结果可用。这种方式依赖编译器来做好调度指令的工作以避免停滞。但这种方式并不是总有效，所以按序执行处理器会有一些停滞的时钟周期，不能执行新的指令。

一种减少停滞时钟周期数量的方法是允许处理器乱序执行指令。处理器会试图从当前的指令流中找到那些与当前停滞的指令相独立并能与之并行执行的指令。x64 系列处理器都是乱序执行处理器。乱序执行的弊端是随着乱序程度的增加，处理器急剧地变得更加复杂。

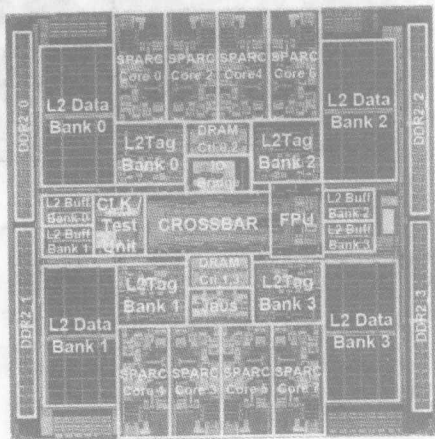


图 1-2 UltraSPARC T1 的核心

① 兆赫(MHz) = 每秒 100 万个周期。千兆赫(GHz) = 每秒 10 亿个周期。

当指令流存在小间隙时,乱序执行是保持处理器利用率的好方法。但是,如果指令流中存在一个较大的间隙,比如在处理器等待数据从内存中取出时,那么乱序执行处理器将不会比按序执行处理器好多少。

1.5 芯片多线程

芯片多线程(CMT)是对复杂的乱序执行处理器的一种替代技术。按序执行处理器设计更简单并且比乱序执行处理器消耗更少的电能,但它存在更多等待前一条指令的结果变为可用的时间。CMT被用在了UltraSPARC T1处理器中,有多个线程的指令在同一个核上执行。当一个线程被阻塞时,一个或更多个线程将准备好执行。因此,处理器的每一个核几乎在每一个时钟周期都可以执行一条指令——处理器的利用率提高了。

以往,强调的是尽可能地提高一个线程的性能,但CMT强调的是单位时间(吞吐量)能够做多少工作,而不是每份工作需要多长时间(响应时间)。

Web服务器是一个很适合在CMT系统上运行的应用的例子。Web服务器的性能通常以每秒钟它能够处理的页面数量来衡量,即以吞吐量为衡量标准。有多个可用的硬件线程来处理页面请求以提高系统的性能。另一方面,服务器的“响应性”通常取决于它花了多少时间将页面通过网络送出去,而不是它花了多少时间让Web服务器准备这个页面,所以处理器准备这个页面的响应时间和它将页面通过网络送出去的时间相比是很短的。

1.6 执行管道

为了能在每个时钟周期内执行大量的指令,处理器会有多个“管道”,每一个都可以处理一种特定类型的指令。这种类型的处理器被称为是超标量处理器。一般来说,它有内存管道(用来处理有关内存的操作,如装入和存储),浮点管道(用来处理浮点算术),整型管道(用来处理整型算术,如加法和减法),以及分支管道(用来处理分支以及调用指令)。多执行管道的一个例子如图1-3。

另外一种提高处理器时钟速率的方法是让指令执行流水线化,这意味着每条指令实际上花了多个时钟周期来完成,而且在每一个时钟周期内处理器只执行了整条指令的一小步。

流水线的例子是把执行一条指令的过程分为取指令(把下一条指令从内存中取出来)、解码(确定指令要处理器做什么)、执行(进行工作)和送回结果(提交指令执行的结果),这是一个四级流水线,这种流水线如图1-4所示。这样做的优点是当一条指令正处于取指令逻辑阶段,第二条指令可以处于解码逻辑阶段,第三条可以处于执行逻辑阶段,第四条可以处于送回逻辑阶段。流水线执行的速度

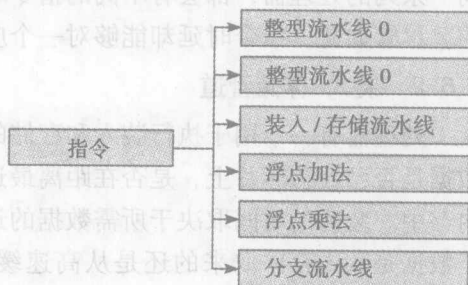


图1-3 多执行管道示例

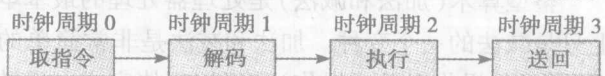


图1-4 四级流水线