

Java 程序性能优化

——让你的Java程序更快、更稳定

葛一鸣 等编著

一个优秀的程序员，不仅要会编写程序，更要会编写高质量的程序
感受Java开发中的大智慧，让你的Java程序更优美

专注于Java应用程序的优化方法、技巧和思想

深入剖析软件设计层面、代码层面、JVM虚拟机层面的优化方法
理论结合实际，使用丰富的示例帮助读者理解理论知识

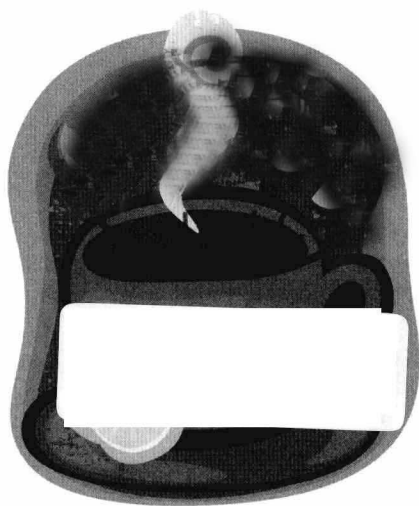


清华大学出版社

Java 程序性能优化

——让你的Java程序更快、更稳定

葛一鸣 等编著



清华大学出版社
北 京

内 容 简 介

Java 是目前应用最为广泛的软件开发平台, 学习针对 Java 程序的优化方法有重要的现实意义。本书以 Java 性能调优为主线, 系统地阐述了与 Java 性能优化相关的知识与技巧。

本书共 6 章, 先后从软件设计、软件编码、JVM 调优以及程序故障排除等方面介绍针对 Java 程序的优化方法。第 1 章介绍性能的基本概念、定律、系统调优的过程和注意事项; 第 2 章从设计层面介绍与性能相关的设计模式、组件。第 3 章从代码层面介绍如何编写高性能的 Java 程序; 第 4 章介绍并行开发和如何通过多线程提高系统性能; 第 5 章立足于 JVM 虚拟机层面, 介绍如何通过设置合理的 JVM 参数提升 Java 程序的性能; 第 6 章为工具篇, 介绍获取和监控程序或系统性能指标的各种工具, 包括相关的故障排查工具。

本书适合所有 Java 程序员、软件设计师、架构师以及软件开发爱好者, 对于有一定经验的 Java 工程师, 本书更能帮助突破技术瓶颈, 深入 Java 内核开发!

本书封面贴有清华大学出版社防伪标签, 无标签者不得销售。

版权所有, 侵权必究。侵权举报电话: 010-62782989 13701121933

图书在版编目 (CIP) 数据

Java 程序性能优化: 让你的 Java 程序更快、更稳定 / 葛一鸣等编著. —北京: 清华大学出版社, 2012.10
ISBN 978-7-302-29625-6

I. ①J… II. ①葛… III. ①JAVA 语言—程序设计 IV. ①TP312

中国版本图书馆 CIP 数据核字 (2012) 第 184193 号

责任编辑: 夏兆彦

封面设计: 欧振旭

责任校对: 徐俊伟

责任印制: 张雪娇

出版发行: 清华大学出版社

网 址: <http://www.tup.com.cn>, <http://www.wqbook.com>

地 址: 北京清华大学学研大厦 A 座 邮 编: 100084

社 总 机: 010-62770175

邮 购: 010-62786544

投稿与读者服务: 010-62776969, c-service@tup.tsinghua.edu.cn

质 量 反 馈: 010-62772015, zhiliang@tup.tsinghua.edu.cn

印 刷 者: 清华大学印刷厂

装 订 者: 北京国马印刷厂

经 销: 全国新华书店

开 本: 185mm×260mm

印 张: 26

字 数: 649 千字

版 次: 2012 年 10 月第 1 版

印 次: 2012 年 10 月第 1 次印刷

印 数: 1~5000

定 价: 59.00 元

前 言

关于 Java

Java 是目前应用最为广泛的软件开发平台之一。随着 Java 以及 Java 社区的不断壮大, Java 早已不再是简简单单的一门计算机语言了, 它更是一个平台、一种文化、一个社区。

作为一个平台, JVM 虚拟机扮演着举足轻重的作用。除了 Java 语言, 任何一种能够被编译成字节码的计算机语言都属于 Java 这个平台。Groovy、Scala、JRuby 等都是 Java 平台的一个部分, 它们依赖于 JVM 虚拟机, 同时, Java 平台也因为它们变得更加丰富多彩。

作为一种文化, Java 几乎成为了“开源”的代名词。在 Java 程序中, 有着数不清的开源软件和框架, 如 Tomcat、Struts、Hibernate、Spring 等。就连 JDK 和 JVM 自身也有不少开源的实现, 如 OpenJDK、Harmony。可以说, “共享”的精神在 Java 世界里体现得淋漓尽致。

作为一个社区, Java 拥有无数的开发人员, 数不清的论坛、资料。从桌面应用软件、嵌入式开发到企业级应用、后台服务器、中间件, 都可以看到 Java 的身影。其应用形式之复杂、参与人数之庞大也令人咋舌。可以说, Java 社区已经俨然成为了一个良好而庞大的生态系统。

此外, 纯粹作为一门软件开发语言, Java 非常容易学习, 其学习曲线较 C++ 等老牌计算机语言相比, 也比较平缓。因为它尽力简化或去除了 C++ 中许多晦涩、多余和难以理解的部分, 如指针、虚函数、多继承等。

本书架构

本书主要介绍 Java 应用程序的优化方法和技巧, 总共分为 6 章。

第 1 章是综述, 介绍了性能的基本概念、两个重要的定律(木桶原理和 Amdahl 定律), 以及系统调优的一般过程与注意事项。

第 2 章从设计层面, 介绍与性能相关的设计模式、组件以及有助于改善性能的软件设计思想。

第 3 章从代码层面介绍如何编写高性能的 Java 代码。涉及的主要内容有字符串的优化处理、文件 I/O 的优化、核心数据库结构的使用、Java 的引用类型以及一些常用的惯例。

第 4 章介绍并行程序开发的相关内容, 以及如何通过多线程提高系统性能。先后介绍了并发设计模式、线程池、并发数据结构的使用、并发控制方法、“锁”的优化、无锁的使用以及协程。

第 5 章立足于 JVM 虚拟机层面,介绍如何通过设置合理的 JVM 参数提升 Java 程序的性能。

第 6 章为工具篇,主要介绍获取和监控程序或系统性能指标的各种工具,以及 Java 应用程序相关的故障排查工具。

本书特点

本书的主要特点有:

- ☐ 专注于介绍 Java 应用程序的优化方法、技巧和思想,并深度剖析 JDK 的部分实现。
- ☐ 具有较强的层次性和连贯性,依次介绍了在软件设计层面、代码层面、JVM 虚拟机层面的优化方法。
- ☐ 理论结合实际,使用丰富的示例帮助读者理解理论知识。

阅读人群

要通读本书并取得良好的学习效果,要求读者具备 Java 的基本知识。本书不是一本帮助初学者入门的书籍。因此,本书适合以下读者:

- ☐ 拥有一定开发经验的 Java 开发人员;
- ☐ Java 软件设计师、架构师;
- ☐ 系统调优人员;
- ☐ 有一定的 Java 基础并希望更进一步的程序员。

本书的约定

本书在叙述过程中,有如下约定:

- ☐ 本书中所述的 JDK 1.5、JDK 1.6 分别等同于 JDK 5、JDK 6;
- ☐ 如无特殊说明,JVM 虚拟机均指 Hot Spot 虚拟机;
- ☐ 如无特殊说明,本书的程序、示例均在 JDK 1.6 环境中运行。

下载提示

本书涉及的源程序请读者直接登录清华大学出版社网站(<http://www.tup.com.cn>),搜索到本书页面后按照提示进行下载。

本书作者

本书由葛一鸣编著。其他参与编写和资料整理的人员有武冬、郅晓娜、孙美芹、卫丽行、尹翠翠、蔡继文、陈晓宇、迟剑、邓薇、郭利魁、金贞姬、李敬才、李萍、刘敬、陈慧、刘艳飞、吕博、全哲、余勇、宋学江、王浩、王康、王楠、杨宗芳、张严虎、周玉。

本书的写作过程远比我想象中的艰辛。为了让全书能够更清楚、更正确地表达和论述，笔者经历了无数个不眠之夜。即使现在回想起来，也忍不住让我打个寒战。但由于写作水平和写作时间的限制，书中难免会有不妥之处。为此，读者可以通过邮箱 bookservice2008@163.com 与笔者联系。

致谢

在本书的写作过程中，我充满着感激之情。首先是对我的家人，在本书完稿前，父亲病重，但我由于工作上的繁忙未能抽出太多时间照顾他，幸好得到了母亲的大力支持和父亲的谅解，我才能够鼓足勇气，全身心投入到写作之中。同时，母亲对我的悉心照料也让我能够更加专注到工作之中。

同时，我要感谢我的工作单位 UT 斯达康以及两位前辈 Rex Zhu 和 Tao Tao。正是他们在平时工作中对我的细心指导，才能让我有所进步和积累。而这一切，正是本书的基础。

最后，再次感谢我的母亲，祝她身体健康。

葛一鸣

目 录

第 1 章	Java 性能调优概述	1
1.1	性能概述	1
1.1.1	看懂程序的性能	1
1.1.2	性能的参考指标	2
1.1.3	木桶原理与性能瓶颈	2
1.1.4	Amdahl 定律	3
1.2	性能调优的层次	5
1.2.1	设计调优	5
1.2.2	代码调优	5
1.2.3	JVM 调优	6
1.2.4	数据库调优	6
1.2.5	操作系统调优	6
1.3	基本调优策略和手段	7
1.3.1	优化的一般步骤	7
1.3.2	系统优化注意事项	8
1.4	小结	8
第 2 章	设计优化	10
2.1	善用设计模式	10
2.1.1	单例模式	10
2.1.2	代理模式	15
2.1.3	享元模式	24
2.1.4	装饰者模式	27
2.1.5	观察者模式	33
2.1.6	Value Object 模式	37
2.1.7	业务代理模式	40
2.2	常用优化组件和方法	43
2.2.1	缓冲 (Buffer)	43
2.2.2	缓存 (Cache)	46
2.2.3	对象复用——“池”	50
2.2.4	并行替代串行	56
2.2.5	负载均衡	56
2.2.6	时间换空间	62
2.2.7	空间换时间	63

2.3 小结	65
第 3 章 Java 程序优化	66
3.1 字符串优化处理	66
3.1.1 String 对象及其特点	66
3.1.2 subString()方法的内存泄漏	68
3.1.3 字符串分割和查找	71
3.1.4 StringBuffer 和 StringBuilder	74
3.2 核心数据结构	79
3.2.1 List 接口	79
3.2.2 Map 接口	86
3.2.3 Set 接口	97
3.2.4 优化集合访问代码	99
3.2.5 RandomAccess 接口	101
3.3 使用 NIO 提升性能	102
3.3.1 NIO 的 Buffer 类族和 Channel	103
3.3.2 Buffer 的基本原理	104
3.3.3 Buffer 的相关操作	107
3.3.4 MappedByteBuffer 性能评估	114
3.3.5 直接内存访问	116
3.4 引用类型	118
3.4.1 强引用	119
3.4.2 软引用	120
3.4.3 弱引用	121
3.4.4 虚引用	122
3.4.5 WeakHashMap 类及其实现	125
3.5 有助于改善性能的技巧	127
3.5.1 慎用异常	127
3.5.2 使用局部变量	128
3.5.3 位运算代替乘除法	128
3.5.4 替换 switch	129
3.5.5 一维数组代替二维数组	130
3.5.6 提取表达式	131
3.5.7 展开循环	132
3.5.8 布尔运算代替位运算	133
3.5.9 使用 arrayCopy ()	134
3.5.10 使用 Buffer 进行 I/O 操作	135
3.5.11 使用 clone()代替 new	137
3.5.12 静态方法替代实例方法	139
3.6 小结	140
第 4 章 并行程序开发及优化	141
4.1 并行程序设计模式	141
4.1.1 Future 模式	141
4.1.2 Master-Worker 模式	148

4.1.3	Guarded Suspension 模式	153
4.1.4	不变模式	160
4.1.5	生产者-消费者模式	162
4.2	JDK 多任务执行框架	166
4.2.1	无限制线程的缺陷	166
4.2.2	简单的线程池实现	167
4.2.3	Executor 框架	171
4.2.4	自定义线程池	173
4.2.5	优化线程池大小	177
4.2.6	扩展 ThreadPoolExecutor	178
4.3	JDK 并发数据结构	179
4.3.1	并发 List	179
4.3.2	并发 Set	182
4.3.3	并发 Map	182
4.3.4	并发 Queue	183
4.3.5	并发 Deque	186
4.4	并发控制方法	187
4.4.1	Java 内存模型与 volatile	187
4.4.2	同步关键字 synchronized	190
4.4.3	ReentrantLock 重入锁	192
4.4.4	ReadWriteLock 读写锁	195
4.4.5	Condition 对象	196
4.4.6	Semaphore 信号量	198
4.4.7	ThreadLocal 线程局部变量	200
4.5	“锁”的性能和优化	201
4.5.1	线程的开销	201
4.5.2	避免死锁	202
4.5.3	减小锁持有时间	206
4.5.4	减小锁粒度	207
4.5.5	读写分离锁来替换独占锁	208
4.5.6	锁分离	209
4.5.7	重入锁 (ReentrantLock) 和内部锁 (synchronized)	210
4.5.8	锁粗化 (Lock Coarsening)	211
4.5.9	自旋锁 (Spinning Lock)	212
4.5.10	锁消除 (Lock Elimination)	212
4.5.11	锁偏向 (Biased Lock)	214
4.6	无锁的并行计算	214
4.6.1	非阻塞的同步/无锁	214
4.6.2	原子操作	215
4.6.3	Amino 框架介绍	217
4.6.4	Amino 集合	218
4.6.5	Amino 树	222
4.6.6	Amino 图	222
4.6.7	Amino 简单调度模式	223

4.7	协程	226
4.7.1	协程的概念	226
4.7.2	Kilim 框架简介	226
4.7.3	Task 及其状态	227
4.7.4	Fiber 及其状态	228
4.7.5	Kilim 开发环境配置	228
4.7.6	Kilim 之 Hello World	230
4.7.7	多任务通信	232
4.7.8	Kilim 实例及性能评估	233
4.8	小结	236
第 5 章	JVM 调优	237
5.1	Java 虚拟机内存模型	237
5.1.1	程序计数器	237
5.1.2	Java 虚拟机栈	238
5.1.3	本地方法栈	243
5.1.4	Java 堆	244
5.1.5	方法区	245
5.2	JVM 内存分配参数	249
5.2.1	设置最大堆内存	249
5.2.2	设置最小堆内存	250
5.2.3	设置新生代	251
5.2.4	设置持久代	252
5.2.5	设置线程栈	253
5.2.6	堆的比例分配	254
5.2.7	堆分配参数总结	256
5.3	垃圾收集基础	257
5.3.1	垃圾收集的作用	257
5.3.2	垃圾回收算法与思想	258
5.3.3	垃圾收集器的类型	262
5.3.4	评价 GC 策略的指标	263
5.3.5	新生代串行收集器	264
5.3.6	老年代串行收集器	265
5.3.7	并行收集器	265
5.3.8	新生代并行回收 (Parallel Scavenge) 收集器	266
5.3.9	老年代并行回收收集器	267
5.3.10	CMS 收集器	267
5.3.11	G1 收集器 (Garbage First)	270
5.3.12	Stop the World 案例	270
5.3.13	收集器对系统性能的影响	272
5.3.14	GC 相关参数总结	273
5.4	常用调优案例和方法	275
5.4.1	将新对象预留在新生代	275

5.4.2	大对象进入老年代	278
5.4.3	设置对象进入老年代的年龄	279
5.4.4	稳定与震荡的堆大小	280
5.4.5	吞吐量优先案例	281
5.4.6	使用大页案例	282
5.4.7	降低停顿案例	282
5.5	实用 JVM 参数	283
5.5.1	JIT 编译参数	283
5.5.2	堆快照 (堆 Dump)	284
5.5.3	错误处理	285
5.5.4	取得 GC 信息	285
5.5.5	类和对象跟踪	287
5.5.6	控制 GC	288
5.5.7	选择类校验器	289
5.5.8	Solaris 下线程控制	289
5.5.9	使用大页	289
5.5.10	压缩指针	289
5.6	实战 JVM 调优	290
5.6.1	Tomcat 简介与启动加速	290
5.6.2	Web 应用程序介绍	292
5.6.3	JMeter 介绍与使用	293
5.6.4	调优前 Web 应用运行状况	296
5.6.5	调优过程	297
5.7	总结	298
第 6 章	Java 性能调优工具	299
6.1	Linux 命令行工具	299
6.1.1	top 命令	299
6.1.2	sar 命令	301
6.1.3	vmstat 命令	302
6.1.4	iostat 命令	304
6.1.5	pidstat 工具	305
6.2	Windows 工具	309
6.2.1	任务管理器	309
6.2.2	perfmon 性能监控工具	311
6.2.3	Process Explorer	313
6.2.4	pslist 命令行	315
6.3	JDK 命令行工具	317
6.3.1	jps 命令	317
6.3.2	jstat 命令	318
6.3.3	jinfo 命令	322
6.3.4	jmap 命令	323

6.3.5	jhat 命令	324
6.3.6	jstack 命令	326
6.3.7	jstatd 命令	329
6.3.8	hprof 工具	330
6.4	JConsole 工具	332
6.4.1	JConsole 连接 Java 程序	332
6.4.2	Java 程序概况	333
6.4.3	内存监控	333
6.4.4	线程监控	335
6.4.5	类加载情况	335
6.4.6	虚拟机信息	336
6.4.7	MBean 管理	337
6.4.8	使用插件	338
6.5	Visual VM 多合一工具	339
6.5.1	Visual VM 连接应用程序	339
6.5.2	监控应用程序概况	342
6.5.3	Thread Dump 和分析	343
6.5.4	性能分析	344
6.5.5	快照	346
6.5.6	内存快照分析	347
6.5.7	MBean 管理	349
6.5.8	TDA 使用	349
6.5.9	BTrace 介绍	350
6.6	Visual VM 对 OQL 的支持	356
6.6.1	Visual VM 的 OQL 基本语法	356
6.6.2	内置 heap 对象	357
6.6.3	对象函数	359
6.6.4	集合/统计函数	362
6.6.5	程序化 OQL	366
6.7	MAT 内存分析工具	368
6.7.1	初识 MAT	368
6.7.2	浅堆和深堆	371
6.7.3	支配树 (Dominator Tree)	374
6.7.4	垃圾回收根	375
6.7.5	内存泄露检测	376
6.7.6	最大对象报告	378
6.7.7	查找支配者	378
6.7.8	线程分析	379
6.7.9	集合使用情况分析	380
6.7.10	扩展 MAT	381
6.8	MAT 对 OQL 的支持	385

6.8.1	Select 子句	385
6.8.2	From 子句	387
6.8.3	Where 子句	389
6.8.4	内置对象与方法	389
6.9	JProfile 简介	393
6.9.1	JProfile 使用配置	393
6.9.2	内存视图	394
6.9.3	堆快照	394
6.9.4	CPU 视图	395
6.9.5	线程视图	397
6.9.6	JVM 统计信息	397
6.9.7	触发器	398
6.10	小结	400

第 1 章 Java 性能调优概述

本章对性能优化技术进行整体性的介绍，让读者了解性能的概念和性能优化的基本思路与方法。掌握这些内容，有助于读者对性能问题进行系统性的分析。

本章涉及的主要知识点有：

- ☐ 评价性能的主要指标；
- ☐ 木桶原理的概念及其在性能优化中的应用；
- ☐ Amdahl 定律的含义；
- ☐ 性能调优的层次；
- ☐ 系统优化的一般步骤和注意事项。

1.1 性能概述

为什么程序总是那么慢？它现在到底在干什么？时间都花到哪里去了？也许，你经常会抱怨这些问题。如果是这样，那说明你的程序出了性能问题。和功能性问题相比，性能问题在有些情况下，可能并不算什么太大的问题，将就将就，也就过去了。但是，严重的性能问题会导致程序瘫痪、假死，直至奔溃。本节就先来认识性能的各种表现和指标。

1.1.1 看懂程序的性能

对客户端程序而言，拙劣的性能会严重影响用户体验。界面停顿、抖动、响应迟钝等问题会遭到用户不停的抱怨。一个典型的例子就是 Eclipse IDE 工具在 Full GC 时会出现程序假死现象，相信一定被不少开发人员所诟病。对于服务器程序来说，性能问题则更为重要，相信不少后台服务器软件都有各自的性能目标。以 Web 服务器为例，服务器的响应时间、吞吐量就是两个重要的性能参数。当服务器承受巨大的访问压力时，可能出现响应时间变长、吞吐量下降，甚至是抛出内存溢出异常而崩溃。这些问题，都是性能调优需要解决的。

一般来说，程序的性能通过以下几个方面来表现：

- ☐ 执行速度：程序的反映是否迅速，响应时间是否足够短。
- ☐ 内存分配：内存分配是否合理，是否过多地消耗内存或者存在泄漏。
- ☐ 启动时间：程序从运行到可以正常处理业务需要花费多长时间。
- ☐ 负载承受能力：当系统压力上升时，系统的执行速度、响应时间的上升曲线是否平缓。

1.1.2 性能的参考指标

为了能够科学地进行性能分析，对性能指标进行定量评测是非常重要的。目前，一些可以用于定量评测的性能指标有：

- ❑ 执行时间：一段代码从开始运行到运行结束，所使用的时间。
- ❑ CPU 时间：函数或者线程占用 CPU 的时间。
- ❑ 内存分配：程序在运行时占用的内存空间。
- ❑ 磁盘吞吐量：描述 I/O 的使用情况。
- ❑ 网络吞吐量：描述网络的使用情况。
- ❑ 响应时间：系统对某用户行为或者事件做出响应的的时间。响应时间越短，性能越好。

1.1.3 木桶原理与性能瓶颈

木桶原理又称“短板理论”，其核心思想是：一只木桶盛水的多少，并不取决于桶壁上最高的那块木块，而是取决于桶壁上最短的那块，如图 1.1 所示。



图 1.1 木桶原理示意图

将这个理论应用到系统性能优化上，可以这么理解，即使系统拥有充足的内存资源和 CPU 资源，但是如果磁盘 I/O 性能低下，那么系统的总体性能是取决于当前最慢的磁盘 I/O 速度，而不是当前最优越的 CPU 或者内存。在这种情况下，如果需要进一步提升系统性能，优化内存或者 CPU 资源是毫无用处的。只有提高磁盘 I/O 性能才能对系统的整体性能进行优化。而此时，磁盘 I/O 就是系统的性能瓶颈。

 **注意：**根据木桶原理，系统的最终性能取决于系统中性能表现最差的组件。因此，为了提升系统整体性能，必须对系统中表现最差的组件进行优化，而不是对系统中表现良好的组件进行优化。

根据应用的特点不同,任何计算机资源都有可能成为系统瓶颈。其中,最有可能成为系统瓶颈的计算资源如下。

- ❑ 磁盘 I/O: 由于磁盘 I/O 读写的速度要比内存慢很多,程序在运行过程中,如果需要等待磁盘 I/O 完成,那么低效的 I/O 操作会拖累整个系统。
- ❑ 网络操作: 对网络数据进行读写的情况与磁盘 I/O 类似。由于网络环境的不确定性,尤其是对互联网上数据的读写,网络操作的速度可能比本地磁盘 I/O 更慢。因此,如不加特殊处理,也极可能成为系统瓶颈。
- ❑ CPU: 对计算资源要求较高的应用,由于其长时间、不间断地大量占用 CPU 资源,那么对 CPU 的争夺将导致性能问题。如科学计算、3D 渲染等对 CPU 需求旺盛的应用。
- ❑ 异常: 对 Java 应用来说,异常的捕获和处理是非常消耗资源的。如果程序高频率地进行异常处理,则整体性能便会有明显下降。
- ❑ 数据库: 大部分应用程序都离不开数据库,而海量数据的读写操作可能是相当费时的。而应用程序可能需要等待数据库操作完成或者返回请求的结果集,那么缓慢的同步操作将成为系统瓶颈。
- ❑ 锁竞争: 对高并发程序来说,如果存在激烈的锁竞争,无疑是对性能极大的打击。锁竞争将会明显增加线程上下文切换的开销。而且,这些开销都是与应用需求无关的系统开销,白白占用宝贵的 CPU 资源,却不带来任何好处。
- ❑ 内存: 一般来说,只要应用程序设计合理,内存在读写速度上不太可能成为性能瓶颈。除非应用程序进行了高频率的内存交换和扫描,但这些情况比较少见。使内存制约系统性能的最可能的情况是内存大小不足。与磁盘相比,内存的大小似乎小的可怜,这意味着应用软件只能尽可能将常用的核心数据读入内存,这在一定程度上降低了系统性能。

1.1.4 Amdahl 定律

Amdahl 定律是计算机科学中非常重要的定律,它定义了串行系统并行化后加速比的计算公式和理论上限。

加速比定义: 加速比=优化前系统耗时/优化后系统耗时

所谓加速比,就是优化前的耗时与优化后耗时的比值。加速比越高,表明优化效果越明显。

Amdahl 定律给出了加速比与系统并行度和处理器数量的关系。设加速比为 *Speedup*, 系统内必须串行化的程序比重为 *F*, CPU 处理器数量为 *N*, 则有:

$$Speedup \leq \frac{1}{F + \frac{1-F}{N}}$$

根据这个公式,如果 CPU 处理器数量趋于无穷,那么加速比与系统的串行化率成反比,如果系统中必须有 50%的代码串行执行,那么系统的最大加速比为 2。

假设有一程序分为以下步骤执行，每个执行步骤花费 100 个时间单位。其中，只有步骤 2 和步骤 5 可以进行并行，步骤 1、3、4 必须串行，如图 1.2 所示。在全串行的情况下，系统合计耗时 500 个时间单位。

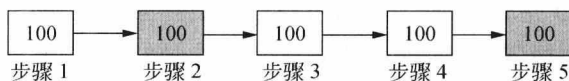


图 1.2 串行工作流程

若将步骤 2 和步骤 5 并行化，假设在双核处理上，则有如图 1.3 所示的处理流程。在这种情况下，步骤 2 和步骤 5 的耗时将为 50 个是单位。故系统整体耗时为 400 个时间单位。根据加速比的定义有：

$$\text{加速比} = \text{优化前系统耗时} / \text{优化后系统耗时} = 500 / 400 = 1.25$$

或者前文中给出的加速比公式。由于 5 个步骤中，3 个步骤必须串行，因此其串行化比重为 $3/5=0.6$ ，即 $F=0.6$ ，且双核处理器的处理器个数 N 为 2。代入公式得：

$$\text{加速比} = 1 / (0.6 + (1 - 0.6) / 2) = 1.25$$

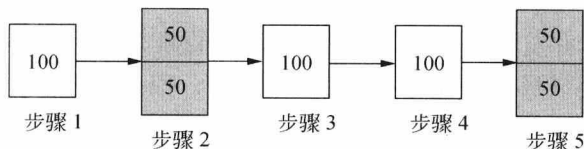


图 1.3 双核处理上的并行化

在极端情况下，假设并行处理器个数为无穷大，则有如图 1.4 所示的处理过程。步骤 2 和步骤 5 的处理时间趋于 0。即使这样，系统整体耗时依然大于 300 个时间单位。即加速比的极限为 $500/300=1.67$ 。

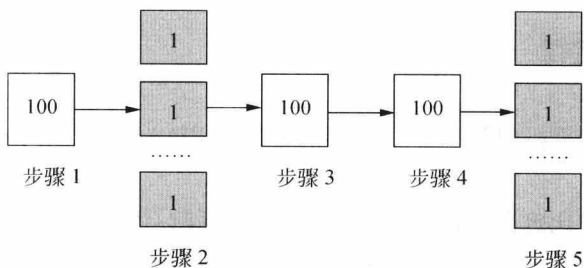


图 1.4 极端情况下的并行化

使用加速比计算公式， N 趋于无穷大，有 $\text{Speedup} = 1/F$ ，且 $F=0.6$ ，故有 $\text{Speedup}=1.67$ 。

由此可见，为了提高系统的速度，仅增加 CPU 处理器的数量并不一定能起到有效的作用，需要从根本上修改程序的串行行为，提高系统内可并行化的模块比重，在此基础上，合理增加并行处理器数量，才能以最小的投入，得到最大的加速比。

注意：根据 Amdahl 定律，使用多核 CPU 对系统进行优化，优化的效果取决于 CPU 的数量以及系统中的串行化程序的比重。CPU 数量越多，串行化比重越低，则优化效果越好。仅提高 CPU 数量而不降低程序的串行化比重，也无法提高系统性能。