深入 C++ 系列

Parallel and Distributed Programming Using C++

C++ 并行与分布式编程

[美] Cameron Hughes / Tracey Hughes 著
肖和平 张杰良 等 译

中国电力出版社
www.infopower.com.cn

深入 C++ 系列

Parallel and Distributed Programming Using C++

C++ 并行与分布式编程

[美] Cameron Hughes / Tracey Hughes 著
肖和平 张杰良 等 译

中国电力出版社
www.infopower.com.cn

Parallel and Distributed Programming Using C++ (ISBN 0-13-101376-9)
Cameron Hughes & Tracey Hughes

北京市版权局著作权合同登记号 图字：01-2004-1298 号

图书在版编目（CIP）数据
C++并行与分布式编程 /（美）休斯（Hughes,C.），（美）休斯（Hughes,T.）著；肖和平等译.
—北京：中国电力出版社，2004
ISBN 7-5083-2281-9
I.C... II.①休...②休...③肖... III.C 语言－程序设计 IV.TP312

中国版本图书馆 CIP 数据核字（2004）第 053589 号

丛 书 名：深入 C++系列
书　　名：C++并行与分布式编程
编　　著：（美）Cameron Hughes & Tracey Hughes
翻　　译：肖和平 张杰良 等
责任编辑：陈维宁
出版发行：中国电力出版社
　　地址：北京市三里河路 6 号　　邮政编码：100044
　　电话：（010）88515918　　传　　真：（010）88518169
印　　刷：汇鑫印务有限公司
开　　本：787×1092　1/16　　印　　张：30.5　　字　　数：666 千字
书　　号：ISBN 7-5083-2281-9
版　　次：2004 年 8 月北京第 1 版　　2004 年 8 月第 1 次印刷
定　　价：59.80 元

译 者 序

为了使用多线程技术在分布式环境下进行项目开发，我们阅读过很多并行与分布式编程方面的书籍，这些书籍确实也让我们学到不少如何进行并行与分布式编程的技能，学会了使用一些工具进行并行与分布式编程，比如使用 CORBA 进行分布式编程等等。但是，我们一直没有完整地认识并行与分布式编程中所固有的问题，更不可能对这些问题提出非常漂亮的解决方案，因此，在项目的实际开发过程中，总是“只见树木，不见森林”，不能“由此及彼，由表及里”。尤其是在进行分布式系统建模时，有无从下手的感觉。看完本书后，我们豁然开朗，对并行与分布式编程有了崭新的认识，解决了一些实际开发过程中一直困扰我们的疑团。

与其他很多有关并行与分布式编程的书籍相比，本书不是详细地介绍用于并行与分布式编程的工具的使用，而是从分析并行与分布式编程中固有的问题入手，并紧紧围绕这些问题，从程序体系结构的角度系统地提出了多种解决方案及模型。它除了能够帮助我们了解用于并行与分布式编程的一些技术和开发工具之外，更是提出了支持并行与分布式编程的软件组织方法，使我们能够从整体上把握并行与分布式编程的关键。这一点无论对编程人员还是软件设计师都是至关重要的。

本书的翻译人员由国防科技大学计算机学院并行与分布式计算方向的研究人员组成，他们都参加过国家“863”重点课题的研究工作，具备丰富的理论和实践经验。

非常感谢 Cameron Hughes 和 Tracey Hughes 为我们带来一本如此优秀的著作，我们从本书中学到很多以前想学而没有学到的东西，极大地拓宽了我们的视野。如果你正在进行并行或分布式软件开发，或者希望进行并行或分布式软件开发，那么，本书必定是一本不可多得的参考书籍。

本书主要由肖和平、张杰良翻译。参加本书翻译的还有方小永、张绪冰、曲向丽、李磊、李蕾等。全书最后由张杰良统稿。**Be Flying 工作室**负责人肖国尊负责本书的译员组织与翻译质量和进度的控制。由于我们的水平有限，错误与不到位之处在所难免。敬请广大读者提供反馈意见，读者可以将意见 E-mail 至 be_flying@sohu.com，我们会仔细查阅读者发来的每一封邮件，以求进一步提高今后翻译的质量。

本书谨献给所有崇尚代码的人、熬白头发的黑客、夜间加班的工程师，以及无数不知疲倦地贡献他们的技能、智慧、经验和时间，使开源运动成为现实并且使 Linux 变革成为可能的志愿者们。没有他们出色的奉献，开发集群编程、MPP 编程、SMP 编程和分布式编程所需的软件不可能像现在这样在世界范围内被所有人广泛地接受和利用。

序　言

本书为使用 C++语言的分布式和并行编程提供了体系结构上的方法，对 C++语言的标准库、算法以及容器类在分布式和并行环境中如何运转进行了特别关注，对通过类库和函数库扩展 C++语言，以实现分布式和并行编程任务的方法进行了解释，强调了对于多线程 C++语言如何与新的 POSIX 和单一 UNIX 标准协同工作。同样本书也讨论了将 C++语言可执行代码与其他语言的可执行代码结合起来，实现分布式或者并行编程问题多语言的解决方法，并且介绍了几种支持并行和分布式编程的软件组织方法。

本书示范了如何排除并行操作中的基本障碍，探讨了新兴的并行化思想。本书中我们关注的核心不是优化技术、硬件细节、性能比较或者试图将并行编程技术应用到复杂的科学或数学算法中；而是关注于如何组织计算机程序和软件系统以充分发挥并行化的优势。此外，我们还会让读者熟悉解决某些分布式或并行编程固有问题的多范例（multiparadigm）方法。有效解决这些问题通常要求综合采用几种软件设计和工程方法。例如，我们采用面向对象编程技术来解决数据竞争和同步问题。我们使用面向 agent 的体系结构来处理多进程和多线程的管理。黑板（blackboard）方法则用来使流出的通信量最小。除了面向对象、面向 agent 和面向人工智能的编程技术外，我们还使用参数化编程技术来实现适合于需要并发操作的通用算法（generalized algorithm）。我们开发不同规模和形式的软件的经历使我们相信：成功的软件设计和实现要求通用性。本书中我们提出的建议、思想和解决方法正是我们这些经历的体现。

并行和分布式编程的挑战

编写并行和分布式程序存在三个基本的挑战，它们分别是：

（1）确认问题领域的环境中存在的固有并行性。

（2）将软件适当地分成两个或多个任务，这些任务可以在同一时刻执行，以完成所需的并行性。

（3）协调上述过程中所分隔的任务，使软件正确高效地运行，从而达到预期目的。

对于并行操作，这三个挑战中会有下述障碍：

数据竞争	死锁检测
部分失效	延迟
死锁	通信失效
终止检测	缺少全局状态
多时钟问题	协议不匹配
定位错误	缺少集中资源分配

本书解释了这些障碍是什么、它们为什么发生以及如何管理它们。

最后，我们在并发操作中采用的几种机制使用 TCP/IP 作为协议。特别是 MPI（Message Passing

Interface，消息传递接口）库、PVM（Parallel Virtual Machine，虚拟并行计算机）库和 MICO（CORBA）库，这些机制都使用 TCP/IP 作为协议。因此，这允许我们的方法可以用于 Internet/Intranet 环境，这意味着并行协同运转的程序可以在 Internet 或者企业的 Intranet 中的不同站点上执行，并且通过消息传递通信。关于并发操作的思想，有许多是 Web 服务基础结构的基石。除了 MPI 和 PVM 例程外，我们使用的 CORBA 对象可以跨越 Internet 与不同的服务器进行通信。这些构件可以用来提供多种 Internet/Intranet 服务。

并行和分布式编程的方法

我们提倡对分布式和并行编程中存在的挑战和障碍使用组件方法。我们主要的目标是使用框架类作为并发的构建块。框架类通过面向对象的互斥锁、信号量、管道和套接字支持。通过使用接口类，任务同步和通信的复杂度大大降低了。我们采用 agent 驱动的线程和进程来促进线程和进程的管理。对于全局状态及其相关问题，我们的主要方法包括黑板模型的使用。我们综合面向 agent 和面向对象的体系结构来实现多范例解决方法。通过使用支持面向对象编程、参数化编程和结构化编程的 C++语言，使多范例的方法成为可能。

为什么选用 C++?

实际上每个平台和操作环境都有可以利用的 C++编译器。ANSI（American National Standards Institute，美国国家标准化协会）和 ISO（International Standard Organization，国际标准化组织）已经为 C++语言及其库定义了标准。C++语言有与商业实现一样健壮的开源实现。C++语言在世界范围内已经广泛地被研究人员、设计者和专业开发人员采用。C++语言已经用于解决不同规模和形式（从驱动程序到大规模的工业应用程序）的问题。C++语言支持软件开发的多范例方法，并且已经可以使用增加了并行和分布式编程能力的库。

并行和分布式编程的库

MPICH（MPI 的一种实现）、PVM 库和 Pthread（POSIX 线程）库用来实现使用 C++的并行编程；MICO（CORBA 标准的 C++实现）用来实现分布式编程。C++标准库与 CORBA 和 Pthread 库结合在一起，为本书讨论的面向 agent 和黑板编程概念提供了支持。

新的单一 UNIX 规范标准

新的单一 UNIX 规范标准，版本号为 3，于 2001 年发布，是 IEEE 和 Open Group 合作的结晶。新的单一 UNIX 规范包括 POSIX 标准，并且为应用程序员提供了可移植性。该标准为软件开发人员提供一个单独的、所有 UNIX 系统都支持的 API 集合。它为需要编写多任务和多线程应用程序的程序员提供了一个可靠的标准路线图。本书中我们关于进程创建、进程管理、Pthread 库、新的 posix_spawn()例程、POSIX 信号量和 FIFO 的讨论都基于单一 UNIX 规范标准。本书中的附录 B 包含

的单一 UNIX 规范标准的摘录可以用作我们所提供的资料的引用。

本书面向的读者

本书为软件设计人员、软件开发人员、应用程序员、研究员、教育工作者以及使用 C++语言、需要并行和分布式编程导论的学生编写。学习本书需要具有初步的 C++语言和标准 C++类库的知识。本书不是 C++编程或者面向对象编程的指南，假定了读者对面向对象技术（例如封装、继承和多态）已有基本的了解。本书介绍了 C++语言环境中并行和分布式编程的基本知识。

支持的开发环境

本书中所提供的例子和程序都是在 Linux 和 UNIX 环境，具体说就是 Solaris 8、Aix 和 Linux（SuSE，Red Hat）下开发和测试的。PVM 和 MPI 代码是在基于 Linux、具有 32 个节点的集群上开发和测试的。本书中的大多数程序在 SUN 公司的 Enterprise 450 上进行了测试。我们使用了 SUN 公司的 C++ Workshop、Portland Group 的 C++编译器以及 GNU C++。本书中的大多数例子都可以在 UNIX 和 Linux 两个环境中运行，对于在这两种环境下都不能运行的例子，本书在为完整的程序例子提供的程序说明（Program Profile）中进行了注释。

补充

UML 图

本书中的许多图使用了 UML（Unified Modeling Language，统一建模语言）标准，特别是用来描述重要的并行操作的体系结构和类关系的活动图、部署图、类图、状态图。尽管在本书中对 UML 的了解不是必需的，但是精通 UML 的话会有很大的帮助。附录 A 包含了本书中所使用的 UML 符号和语言的解释和描述。

程序说明

本书中每个完整的程序都有一个程序说明。说明中包含有诸如所需的头文件、所需的库、编译指令和链接指令的实现的详细说明。说明同样也包括一个“注意”说明段，该段中包含了执行该程序时需要注意的特殊考虑事项。对于没有说明的代码则意味着该代码仅限于讲解目的。

补充内容

我们尽量避免在像本书一样的入门教材中采用过于理论的表示法。但是，在某些情况下理论或者数学表示法是不可避免的。在这种情况下，我们使用了理论或者数学表示法，但是我们在补充内容中为该表示法提供了详细的解释。

测试和代码可靠性

尽管本书中所有的代码和应用程序为了保证正确性已经进行了测试，但是我们不保证本书中所包含的程序没有缺陷或者错误，与任何特定标准或者商品一致或者对于任何特定应用程序满足你的需

求。我们不应依赖这些程序来解决问题，因为不正确的解决方法将导致人身的伤害或者财产的损失。作者和出版商拒绝承担你使用本书中的例子、程序或者应用程序所造成的直接或间接伤害的所有责任。

致谢

如果没有许多朋友和同事的帮助、建议、建设性批评以及他们提供的资源，我们不可能成功地完成本书的编写。特别要指出的是，我们要感谢 OSC（Ohio Super-Computing）公司的 Terry Lewis 和 Doug Johnson，他们为我们提供了基于 Linux、具有 32 个节点的集群的完全访问；感谢 YSU 大学的 Mark Welton，他为我们配置支持 PVM 和 MPI 程序的集群提供了帮助，提出了专家意见；感谢 YSU 大学的 Sal Sanders，他允许我们访问运行 Mac OSX 和 Adobe Illustrator 的 Power PC；感谢 YSU 大学的 Brian Nelson 允许我们在 SUN 公司的 E-250 和 E-450 多处理机上测试我们大部分的多线程和分布式程序；我们也受到 YSU 大学 MAAG 的 Mary Ann Johnson 和 Jeffrey Trimble 的帮助，他们为我们查找和保存我们所需的技术引用；感谢 IEEE 标准、认证和合约办公室的 Claudio M. Stanziola、Paulette Goldweber 和 Jacqueline Hansson，从他们那里我们获得授权复印了新的单一 UNIX/POSIX 标准的部分内容；Open Group 组织的 Andrew Josey 和 Gene Pierce 也在这方面提供了帮助；感谢 Z-Group 公司的 Trevor Watkins，他为我们测试了程序例子，他的多 Linux 分布式环境对于测试过程非常重要；特别要感谢的是 Steve Tarasweki，他对本书的草稿进行了技术上的审阅。感谢 Eugene Santos 博士在我们探讨如何将数据结构分类以便用于 PVM 中时为我们指明了方向；感谢 YSU 大学高级计算工作组的 Mike Crescimanno 博士允许我们引用一些 ACWG 会议的资料；最后，感谢 Prentice Hall 出版社的 Paul Petrlia 及其制作组（特别是 Gail Cicker-Bogusz）容忍我们超过了最终期限以及古怪的 UNIX/Linux 文件格式——我们特别要感谢他们的耐心、勇气、激情和专业。

目　录

1 并发编程的乐趣

我认为最好用一种库来支持并发性，并且该库不需要主语言扩展就能够实现。

——C++之父 Bjarne Stroustrup

本章内容

什么是并发？ •并行编程的优点 •分布式编程的优点 •最少的工作需求 •软件并发性的基本层次 •C++中没有关键字支持并行性 • 并行和分布式编程的编程环境 • 小结——关于并发

现在进行软件开发需要具备并行和分布式编程的工作经验。软件通常需要在 Internet、intranet 或某些几乎遍及全球的网络上正确运行，一旦一个软件在一种或多种这样的环境中部署，那么其性能就要受到最严格的要求。用户需要迅速而可靠的结果，很多情况下，用户希望软件能够同时满足很多需求，从 Internet 上同时对软件和数据进行多路下载的能力就是用户对软件性能需求的一个典型例子。将软件设计成能够播放视频，也就是软件要能渲染图像和对声音进行无衰减的量化处理，并且要没有间断。Web 服务器软件每天要受到几十万次访问，在营业时间，频繁使用的电子邮件服务器要被迫承受上百万次发送和接收消息，这点并不罕见。需要进行大量工作的不只是很多消息，还包括内容。例如，假如服务器软件设计不正确，那么包含数字化音乐、电影或图像的传输数据就会耗尽网络带宽，并可能造成服务器软件性能下降。典型的计算环境是网络化的，并且计算环境中的计算机通常有多个处理器。软件能做得越多，对它的需求就越多。为了满足最低程度的用户需求，当前的软件必须功能强大而且小巧。软件必须设计成能充分利用拥有多处理器计算机的优点。由于网络化的计算机比非网络化的计算机更常见，所以所设计的软件必须能够正确有效地运行，并且它的一些程序段能同时在不同的计算机上执行。某些情况下，不同的计算机拥有完全不同的操作系统和网络协议！为了适应这些现实状况，软件开发指令系统中必须包含用并行和分布式编程实现并发操作的技术。

1.1 什么是并发？

如果两个事件在同一时间间隔内发生就称这两个事件是并发的，两个或多个任务在同一时间间隔内执行叫做并发执行。对于我们而言，并发并不一定就表示在同一精确时刻执行，例如，两个任务可能在同一秒发生，但是每个任务在该秒的不同时间片内执行。第一个任务可能执行第一个十分之一秒后暂停，第二个任务执行第二个十分之一秒后暂停，然后第一个任务可能继续开始执行第三个十分之

一秒，如此类推，每个任务交替执行。但是一秒的时间非常短，所以看起来两个任务好像是同时执行。我们可以将这个概念推广到更长的时间间隔，同一时间段内执行某项任务的两个程序在该时间段内连续推进任务的完成，尽管它们可能在也可能不在同一精确时刻执行，但我们认为这两个程序在该时段并发执行。处于同一时间的多项任务在同一时间间隔内执行就是并发，并发任务能够在单道处理或多道处理环境中执行。在单道处理环境中，处于同一时间的并发任务通过上下文切换在同一时段内执行；在多道处理环境中，如果有足够多的空闲处理器，那么并发任务可以在同一时间间隔内的同一时刻执行。多长的时间作为一个可以接受的并发时间段，其决定因素与应用程序有关。

并发技术使得计算机程序能够在同一时间间隔或同一时限内做更多的工作，与其把程序设计成在某一时刻只能做一项任务，还不如将计算机程序分段成多项任务，这样，其中的一些任务能够并发执行。某些情况下，在同一时间间隔做更多的工作并不是目的所在，简化程序解决方案才是真正目的。有时把问题的解决方案看作一组并发执行的任务更合理，例如，最好把减肥问题的解决方案看作是并发执行的任务：节食和锻炼，也就是说饮食的改良和正规的身体训练在同一时间间隔内（并不一定是在同一时刻）发生。通常，在一段时间间隔内做一件事，而在完全不同的一段时间间隔内做另一件事不是很好，两个过程的并发执行才是解决方案的自然方式。有时采用并发性旨在使软件运行得更快或者更迅速地完成作业，有时对于效能而言速度处于次要地位，采用并发性则是为了使软件在相同的时间间隔内做更多的工作。例如有些 Web 站点希望只要有可能用户都保持登录状态，它们并不关心用户登录或离开站点是否迅速，而是关心站点能够同时支持多少用户，所以这种软件设计的目的是在尽可能长的时间间隔内处理尽可能多的连接事件。最后，采用并发性可简化软件，通常一系列较短的并发执行操作比一个较长的复杂的顺序操作更容易实现。并发使软件运行速度更快，处理更多的负载或简化程序的设计方案，其主要目标是软件优化，即采用并发性使软件性能更佳。

1.1.1 实现并发的两种基本途径

并行和分布式编程是达到软件并发的两种基本途径，它们是两种不同的、有时又相互交叉的编程范例。并行编程技术将程序必须处理的作业分配给一个物理或虚拟计算机内的两个或多个处理器，分布式编程技术将程序必须处理的作业分配给两个或多个处理器，这些处理器可以也可以不在同一个计算机中，也就是说分布式程序的各部分通常在不同的由网络连接的计算机上运行，或者至少在不同的处理器上运行。包含并行性的程序在同一个物理或虚拟计算机上执行，程序内的并行性可分成进程或线程。我们在第 3 章中讨论进程，在第 4 章中讨论线程。在我们看来，分布式程序仅能分成进程，多线程仅限于并行性。在技术上，并行程序有时候是分布式的，例如 PVM（Parallel Virtual Machine，虚拟并行计算机）编程；分布式编程有时用于实现并行性，例如 MPI（Message Passing Interface，消息传递接口）编程。但是并非所有的分布式程序都包括并行性，分布式程序的各部分可以在不同时间间隔内的不同时刻执行。例如一个日历程序软件可被分成两部分，一部分给用户提供日历和记录重要约会的方法，另一部分给用户提供针对不同类型约会的一组闹钟。用户用软件的一部分来安排约会，而软件的另外一部分可以在不同的时间分开执行，闹钟和行程安排部分组成了一个应用程序，但是它们被分成两个分开执行的部分。而在纯粹的并行程序中，并发执行部分都是同一个程序中的部分，而在分布式程序中，这些并发执行的部分通常实现成分离的程序。图 1-1 中显示了并行和分布式程序的典型结构。

图 1-1 中的并行应用程序由一个分成四项任务的程序组成，每个任务在一个不同的处理器上执

行，因此所有任务都能够同时执行，任务能够由一个进程或者一个线程实现。另一方面，图 1-1 中的分布式应用软件由三个分离的程序组成，每个程序在不同的计算机上执行。程序 3 由同一计算机上执行的两个分离部分组成。尽管程序 3 的任务 A 和任务 D 在相同的计算机上，但是它们是分布式的；因为它们由两个分离的处理器执行。并行程序中的任务比分布式程序中的任务结合得更紧密，通常与分布式程序关联的处理器在不同的计算机上，但是与并行程序关联的处理器在相同的计算机上。当然，也有既具有并行性又具有分布式的混合程序，这些混合的组合程序正成为一种规范。

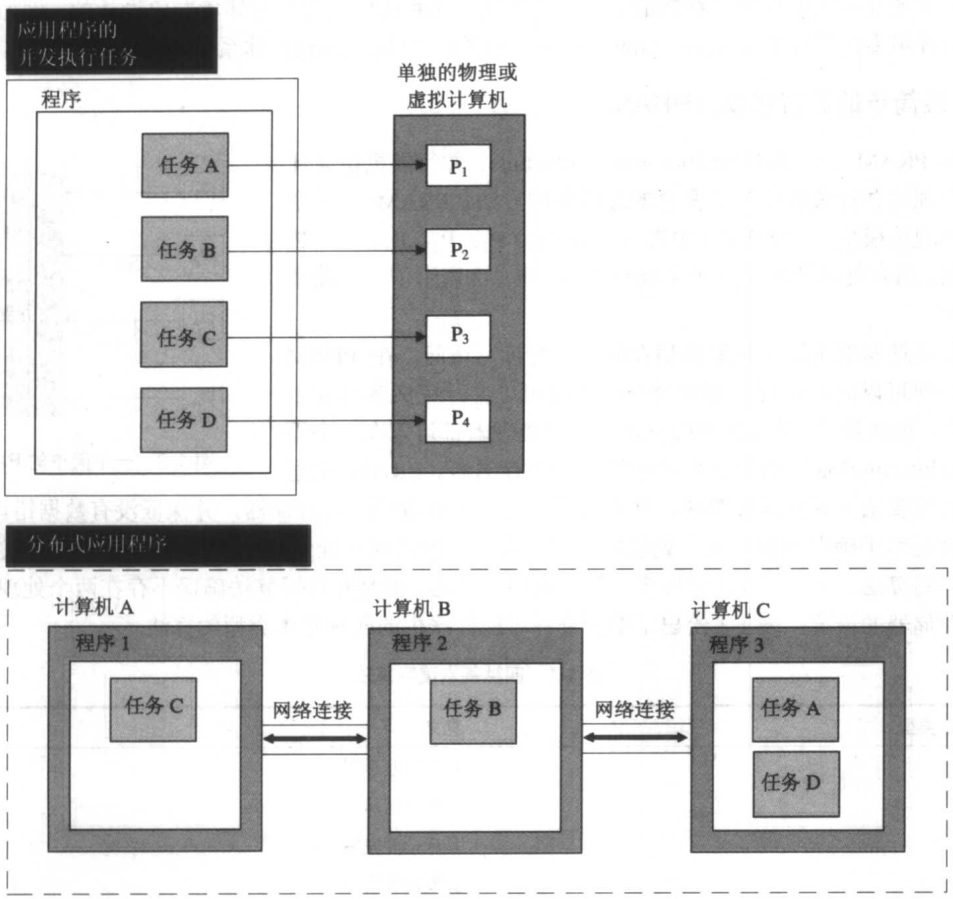

图 1-1　并行和分布式程序的典型体系结构

1.2　并行编程的优点

正确设计的程序利用并行性能够比与其相当的串行程序执行得更快，这就是并行程序的市场优势。在某些情形下，速度用于拯救生命，更快等同于更好。把某些问题的解决方案描述成一组同时执行的任务集合更为自然，特别是在许多科学、数学和人工智能编程领域中更是如此。这意味着，并行编程技术允许软件开发者可以直接实现由研究人员开发的数据结构、算法和搜索法；从而在某些情况

下，可以免除软件开发者的一些工作。有时可能要使用专门的硬件，例如，在高端多媒体程序中，为了增加性能，程序逻辑可分配给专门的处理器，诸如：专用图形芯片、数字声音处理器以及专用算术处理器。通常，这些处理器可被同时访问。MPP（Massively Parallel Processor，大规模并行处理器）计算机拥有数百，有时候达到数千个处理器，可以用来解决实际上用串行方式完全不能解决的问题。对于 MPP 计算机，是快速与纯粹的强力（brute force）的结合才使得不可能变为可能。在大规模并行计算范畴中有环境建模、空间探索以及诸如人类基因工程的几个生物研究领域。更深一层的并行编程技术打开了通往某些软件体系结构的大门，这些软件体系结构是为并行环境专门设计的。例如：针对并行处理环境专门设计的多 agent（multiagent）和黑板（blackboard）体系结构。

1.2.1 最简单的并行模型（PRAM）

使用 PRAM（Parallel Random Access Machine，并行随机访问计算机）是理解并行编程中基本概念的最简单的方法。PRAM 是一个简化了的理论模型，在 PRAM 中有 n 个标识为 P_1，P_2，P_3，…，P_n 的处理器，所有处理器共享一个全局存储器。图 1-2 表示了一个简单的 PRAM。

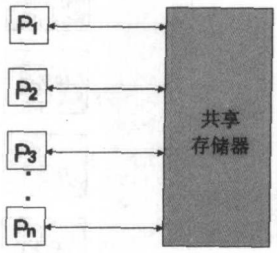

图 1-2 一个简单的 PARM

所有处理器都可以对共享全局存储器进行读写访问，在 PRAM 模型中访问可以同时进行，假定每个处理器可以并行完成各种算术逻辑操作，也就是说，图 1-2 中所有理论上的处理器都可以在一个不间断（uninterruptible）的时间单元中访问全局存储器。PRAM 模型既有并发读算法又有互斥读算法，并发读算法允许同时读同一片存储器，并保证没有数据错误发生；互斥读算法用于确保不存在两个处理器同时读取存储器的同一处地址。PRAM 模型既有并发写算法又有互斥写算法，并发写算法允许多个处理器写存储器，但是互斥写算法确保不存在两个处理器同时写同一存储器的情况。表 1-1 给出了根据读写可能性派生的四种基本类型的算法。

表 1-1 四种基本读写算法

读写算法类型	意义
EREW	互斥读互斥写
CREW	并发读互斥写
ERCW	互斥读并发写
CRCW	并发读并发写

本书中当我们讨论并发体系结构的实现方法时会经常引用到上述算法类型。黑板体系结构是我们用 PRAM 模型实现的最重要的体系结构之一，它将在第 13 章中讨论。需要注意的一点是：尽管 PRAM 是简化了的理论模型，但是 PRAM 用于开发实用程序，并且用 PRAM 开发的程序在性能上完全可以和那些用更复杂的并行模型开发的程序媲美。

1.2.2 最简单的并行分类

PRAM 给我们提供了一个关于考虑计算机怎样划分处理器和存储器的简单模型，并且提出了关于