沈绪榜 编著

MPP 嵌入式计算机设计

清 华 大 学 出 版 社
http://www.tup.tsinghua.edu.cn

MPP 嵌入式计算机设计

沈绪榜　编著

清华大学出版社

(京)新登字 158 号

内 容 简 介

本书介绍当今世界上先进的图象理解等方面的 MPP 嵌入式计算机设计。

全书共分七章。第一章讨论并行模型;第二章讨论体系结构;第三章讨论基本 MPP 微处理器、SIMD 计算机及其应用;第四章讨论浮点 MPP 微处理器与 MIMD 计算机;第五章讨论设计的验证模型与并行验证技术;第六章讨论芯片的工艺测试与设计测试;第七章讨论 MPP 系统的冗余设计与容错设计以及 MCM、WSI 与 3D 等集成技术。

本书可供计算机、电路与系统、超大规模集成系统设计以及图象理解等专业的工作者及研究生参考。

图书在版编目(CIP)数据

MPP 嵌入式计算机设计/沈绪榜编著. —北京:清华大学出版社,1999

ISBN 7-302-03318-8

Ⅰ.M… Ⅱ.沈… Ⅲ.电子计算机-设计 Ⅳ.TP302

中国版本图书馆 CIP 数据核字(1999)第 00835 号

出版者:清华大学出版社(北京清华大学校内,邮编 100084)

http://www.tup.tsinghua.edu.cn

印刷者:北京市清华园胶印厂

发行者:新华书店总店北京发行所

开 本:787×1092 1/16 印张:30 字数:704 千字

版 次:1999 年 4 月第 1 版 1999 年 4 月第 1 次印刷

书 号:ISBN 7-302-03318-8/TP・1784

印 数:0001～1000

定 价:40.00 元

前　言

回顾短暂但很丰富的计算技术发展历史，从真空管到晶体管，到微电子时代以及相应的装配技术的进步，都说明激励这种发展的一个简单目的就是要不断提高单位体积内的计算处理能力。例如，1958 年的晶体管尺寸大约为 1cm，1970 年为 10μm，1985 年为 1μm。到下世纪初，预计则会只有 0.1μm。到 2010 年每块芯片将能容纳 10 亿个器件。概括地说，芯片的集成度大致是每 18 个月就翻一番。为什么计算机要越做越小？原因是只有这样，才能使计算机越来越经济、可靠、高速、方便与智能，从而才能使计算技术的应用达到"无孔不入"的普及地步。正如我们今天所看到的那样，芯片变成了科技发展的媒介，所有高科技都是芯片带给我们的。所以，为了在电子工业及其它工业领域取得领先地位，一些发达国家都争取要在芯片制作与设计上取得领先地位。在国际上，计算机的设计总是与 IC 的发展紧密联系在一起的，大约从 20 世纪 80 年代初开始，就提倡将计算机的设计工作从 PCB 设计向芯片的 MASK 设计转移了。现在看来，这是符合 IC 设计这种事物发展的客观规律的。

VLSI 技术的惊人发展，对计算机本身带来的巨大影响，就是 RISC 技术与并行处理技术已变得越来越现实普及。200MHz ALPHA ，500MHz Super Sparc …… 这些数字指出了一个明显的趋势，亚微米 IC 技术的进步使得微处理器的性能达到了几年之前系统设计者不可想象的地步。支持适度并行处理与大规模并行处理的计算机，经过三十多年来的发展，到今天也成为一种重要的机型了。在并行处理计算机中，通常把包含 1000 个处理器的机器叫做 MPP(massively parallel processors)计算机，而把含有 10 到 100 个处理器的机器叫做 MP (multiprocessors) 计算机，也叫做适度并行处理(moderately parallel processing)计算机。MPP 计算机可以用来达到两个不同的目的，一是可以作为一种以更低价格提供给定计算能力的重要手段；二是可以不管价格如何，可以用来扩充计算机解决问题的范围。针对第一个目的，已有许多图象理解应用的 MPP 嵌入式计算机；针对第二个目的，人们估计 2005 年的目标是在新的电子学、体系结构和软件技术的基础上使 MPP 计算机达到每秒 1000 万亿次的运算能力。MPP 计算机是针对特定应用的，一般应用是不需要这么多处理器的，而大量应用的还是 MP 计算机。我国的 VLSI 技术与国际水平相比虽然差距不小，但也远远超过国外早期研制 MPP 嵌入式计算机芯片的水平。至少可以利用它研制国产芯片 MPP 嵌入式计算机作为更经济地提供给定计算能力的重要手段。例如，完全可以用来研制针对航空航天图象处理嵌入式应用的 MPP 计算机。现代计算机总体积的百分之九十差不多都是由硅电路之外的其它材料占用的，目前主要是靠提高芯片

集成度与装配密度来缩小这个不合理的体积百分比的。而 MPP 计算机体系结构的规则性是非常适合于提高芯片集成度与装配密度的。例如，休斯公司用于航空航天图象处理的三维 MPP 计算机就达到了总体积的百分之九十是由硅电路占用的。所以，当我们要大跨度发展国产芯片计算机的时候，着重考虑了计算机已从 SISD 型机的串行计算步入 SIMD 或 MIMD 型机的 MPP 与 MP 并行计算机这样一个新变化以及如何发展国产并行计算机来加大我们的发展跨度。这一点是非常重要的，因为改变一个体系结构意味着用户的应用软件将必须改变。而一个体系结构下的一个给定实现的改变则只意味着改变硬件及可能必要的操作系统软件，这不需用户来完成。只有体系结构的选择才是最困难的，因为它要包括当前的实现与未来的实现。

随着特征尺寸的减少，芯片集成度是按 $O(n^2)$ 上升的，而器件的速度是按 $O(n)$ 上升的，那么，如何选择计算机的结构才能使它的性能是按 $O(n^3)$ 上升的呢？为此，并行计算成了研究的热点。从微处理器的设计角度来说，就是要从算法级、指令级与进程级等方面来提高处理的并行度。VLSI 算法已是目前从逻辑上提高运算速度的常用技术，使过去要用子程序实现的功能变成了“硅程序”。从概念上讲这一点也不新鲜，但 VLSI 技术所能实现的规模已从量变产生了质变，它不仅在提高计算的速度上，而且也在提高它的智能性上起作用。从人工智能的角度来看，人的思维从行为与功能上讲是串行的“一心不可二用的”，但从结构上讲则是采用大规模并行处理的。所以，利用 VLSI 技术从算法级上提高并行处理能力，也合乎人工智能的特点。有效的支持流水线模型是 RISC 微处理器从指令级并行计算上取得成功的诀窍之一。由此不难看出，实质上并行计算技术的发展与芯片设计是不可分割的，MPP 计算机的研制就更是如此，例如，用于图象理解的 MPP 计算机就是在图象处理并行算法与 SIMD(或 MIMD)体系结构的交错研究过程中发展起来的。

总之，人们在微型化技术方面，取得的成果是特别惊人的。甚至有人预言微型化可能引发新一轮工业革命。对计算机来说，我们正面临着微型化制造技术与设计技术的挑战。基于这样的认识，我们在国产芯片 RISC 微计算机研制的基础上，针对嵌入式应用又进行了国产芯片图象理解 MPP 计算机的设计。但是，就其指令集合而言，也是支持 64 位高性能 MPP 微处理器设计的。虽然我们目前的设计与制作条件还不成熟，但自然希望今天的图象理解 MPP 计算机的设计，也能为明天的高性能 MPP 计算机的设计打下必要的基础。应当指出，只有 MP 计算机才会是一种大量应用的机型，因此，LS MPP 微处理器的设计是考虑了支持 MP 计算机的实现的。本书是以 LS MPP 嵌入式计算机的设计工作为基础的，同时也介绍了其它的 MPP 计算机与 MP 计算机。换句话说，本书所要讨论的是以 VLSI 芯片设计为基础的嵌入式并行计算机。实际上，将并行计算机按其所用处理器多少再细分，并不一定是科学的，因为并行计算机的体系结构是可以做到从几个处理器到几千个处理器的大范围变化的。所以，在本书中 MPP 的意思是 massively/moderately parallel processing。因此，书名为《MPP 嵌入式计算机设计》。内容包括并行模型、体系结构、程序设计、芯片(硬件)设计、VLSI 设计验证、可测试性、容错设计以及应用举例等。其中第一章讨论了 MPP 的技术模型、语义模型与算法模型等三种并行模型的选择，以便为 LS MPP的设计提供必要的基础知识。第二章体系结构主要讨论了将串行计算的 SISD 结构与并行计算的 SIMD 和 MIMD 三种结构能有机地联系在一起的 LS MPP 的指令集合

以及芯片与程序设计。第三、四章讨论了两种 LS MPP 微处理器芯片的设计以及相应的两种 MPP 计算机的设计。其中第三章的基本 MPP 微处理器主要是支持中低层图象处理应用的 MPP 微处理器，是作为协处理器而用来构成 SIMD 计算机的。其中操作描述与图象理解部分实质上也是讨论这种 SIMD 计算机在这方面的应用的。而第四章的 32 位浮点 MPP 微处理器主要是支持中高层的图象处理计算任务的 MPP 微处理器。它实现了 32 位浮点 LS RISC 微处理器的指令集合，是用来构成 MIMD 计算机的。第五章讨论设计验证。设计验证可分为功能验证与性能验证两个方面。在各个层次的设计过程中，这两种验证都将会用到。由于 VLSI 芯片日益复杂，其设计综合与验证越来越需要更快的 VLSI-CAD 支持。因此，VLSI 芯片设计的验证技术一节实质上也是讨论 MIMD 计算机在这方面的应用。本章讨论了验证模型与验证技术两个方面。第六章讨论芯片测试。芯片的好坏取决于制作工艺与设计两个方面。因此，本章是按工艺测试与设计测试两方面讨论的。第七章讨论系统集成的两个问题，一是系统设计中的冗余设计与容错设计。二是集成技术中的 MCM 技术、WSI 技术与 3D 技术，它们都是为提高单位体积内计算处理能力而正在迅速发展的新技术。

沈绪榜

1997 年元月 10 日

目 录

第一章 并行模型

自现代计算机科学的历史开始以来，抽象机或虚拟机的概念就已广泛使用。它的第一个好处是可使开发者不必了解真实计算机体系结构的细节；而第二个好处是能提高软件对计算机体系结构的可移植性。许多程序试图反映或处理真实世界的不同部分，无疑真实世界是高度并行的。所以，采用并行性的可能性一般会使反映真实世界现象与自然求解策略更容易些。对于传统的串行程序设计，不同程序设计策略中的哪一种在提供舒适的程序设计方面是最成功的还远未取得任何共识，相应的并行程序设计更是如此。当处理并行计算时，硬件的具体特征对软件开发的影响，比处理串行计算时更大。软件并发部分在不同硬件层上的布局、负载平衡以及通讯优化成了开发的主要问题。实现限制妨碍纯粹的算法考虑，不能保证软件的可重用性。所以，在并行计算系统的设计中，显然希望能找到强有力的抽象模型，以实现通用并行计算的两个重要目标：一是可剪裁的并行性能；二是体系结构无关的并行软件。因此，在讨论 LS MPP 计算机的体系结构问题之前，先讨论并行计算的技术模型、语义模型以及算法模型，以便为并行体系结构的设计提供必要的基础知识。

LS MPP 是我们针对嵌入式图象理解应用自行设计的一种大规模并行计算系统，可以看作是由语言、算法、体系结构以及它们的相互作用组成的。图象理解大规模并行系统设计与分析的复杂性要求在各个抽象层上采用模型(model)，以提供一个系统的简化看法。模型就是系统的一个抽象看法，或者更合适地说，是去掉细节后的系统的一部分，使人们了解与运用其基本原理。模型可以粗略地划分为三层：高层的程序设计(语义)模型，中层的计算(算法)模型以及低层的体系结构(技术)模型。低层模型注重硬件的具体物理、技术与/或经济性质，而不太注重问题描述的技术；高层模型十分注重人们能最好地描述与解决问题的方法，但不太注重实现细节。只有在中间层上，一个好的模型将能兼顾描述与实现。所以中间层上的模型又称作桥梁模型(bridging model)，可为建立软件与硬件工业所需的公共环境提供一个基础。

在串行计算中，高层有 HOL 程序设计模型，中层有 RAM 计算模型以及低层有 1945 年 Von Neumann 按照 Turing 于 1936 年提出的基本原理，在通用存储程序串行计算机的实际设计中导出的 Von Neumann 体系结构模型。但 RAM 模型只是体系结构模型的一个简单描述，没有真正的差异，所以，一般认为串行计算中只有两层模型。串行计算机的 Von Neumann 模型就是一个桥梁模型。尽管硬件与软件工业不断地在发展，但从 20 世纪 40 年代到现在的长时间内，变化都是在该模型的抽象程度之内，只是在最近，变化才大到不能用该模型概括了。一是技术进步带来的大规模并行体系结构(看来不可能用一种有效

的方法自动地将串行程序并行化)；二是问题的大小已达到一个门限，使 I/O 时间经常必须看得比计算时间更重要(过去用层次存储器隐藏的办法，变得更困难了)，以及新的重要的交互或实时问题要求更多的 I/O。由于并行计算的概念复杂性远远超过串行计算，虽然已经提出了许许多多的模型，但至今还没有通用的统一模型。尽管并行处理技术内在的性能好处早已知道，但模型上的这种多样性，妨碍了并行处理技术的更加迅速的开发与应用。并行计算只是并行活动的一个特例。理想的情况是：并行系统模型应当能适用于模仿并行系统与模访真实世界，即体系结构模型表示必须用到的材料与支配世界的物理定律；计算(算法)模型能描述与设计世界以及分析设计性能的各种抽象；程序设计(语义)模型是了解与论证世界的基础构架(framework)。

1.1 技术模型

体系结构模型又称作技术模型，它是机器的硬件与操作系统的抽象。并行处理由来已久，计算机用户与设计者常把它分为大规模并行(massively parallel)与适度并行(moderate parallel)两种。前者是指采用 1000 个以上处理器结构的并行处理，后者是指采用 10 到 100 个处理器结构的并行处理。并行计算机通常又分为 SIMD 与 MIMD 两种体系结构。它们又可以进一步分为共享存储器的体系结构或分布存储器的体系结构。在共享存储器的多处理机与分布存储器多处理机(更确切地说为多计算机)中，不同的处理器都是通过互连网络来连接的。典型的互连网络有总线、交叉开关、多级网络、环、网格、树以及超立方体等。体系结构模型可以比这更抽象一些，它可以不具体指明网络，而仅仅指明其性能特征(以达到某种体系结构或实现的无关性)由于这个原因，一个体系结构模型可以用作一类网络机器之上的一个桥梁模型。更抽象的体系结构模型基本上抛弃局部性(或接近性)表示(以便通过一个总体地址空间得到使用简单性)以及某种网络体系结构的无关性(严格地说，可以允许区分处理机上与处理机外通讯的有限好处，但这是一种非常有限制的局部性形式)。

在 20 世纪 80 年代的大部分时间内，并行计算的主要动力(driving force)是从硬件方面考虑的。因为 VLSI 技术的进步可以开发许多分布存储器的多计算机体系结构，这些系统由通用的微处理器，通过一个稀疏网络，比如阵列、蝶式或超立方等互连组成。在这种系统中算法效率的关键是细心地利用网络的局部性，通过使消息传递经过最少的节点数来提高效率。然而，在消息传递的办法中，为了提高效率，软件开发的努力多在低层的进程映射活动上，除了麻烦之外，也缺乏可移植性。在并行体系结构迅速变化的情况下，这种办法的体系结构相关性是一个主要弱点。第二个办法是以软件为动力，体现在许多程序设计语言上，比如函数的、单赋值、逻辑的等。为了体系结构无关性，函数语言自然就导致采用图归约作为并行计算的模型。不幸的是，图归约、数据流等并行体系结构的开发没有取得明显的进展。第三个办法是以并行计算模型为动力，串行计算的 Von Neumann 模型为此提供了很好的先例。但并行计算要复杂得多，还没有一个并行计算模型能支配并行计算的发展。已有的 VLSI 系统、脉动阵列与分布存储器多计算机的模型、算法效率的关键都是细心地利用网络的局部性。这些并行计算只能算作专用的，对其设计、分析、实现与验证已有许多研究。当前计算机科学的主要挑战是确定通用并行计算所能达到的程度。目标就是

提供可剪裁的并行性能与体系结构无关的并行软件。但是，当前在MPP中的大量工作是科学应用软件的开发，还没有对上述挑战具体关注，尚没有公认的模型。但也没有充分证据说明通用并行计算是不可能达到的。并行计算机可以是同步的或异步的。一个处理器在与另外一个处理器同步或者数据通讯之前执行的指令数目叫做粒度大小(grain size)。有每几条指令之后使处理器同步的细粒度(fine-grain)并行处理；每几千条指令执行之后使处理器同步的粗粒度(coarse-grain)并行处理；同步之间执行几百条指令的中粒度(medium-grain)并行处理。因此，作为具体实用的技术模型，这里将从同步技术与通讯技术两方面来讨论。

1.1.1 同步技术

同步技术涉及到时钟模型的同步还是异步实现问题。MPP机的时钟方案可以有三种选择。一是对所有处理器采用一个统一的时钟，多用于SIMD结构中，当然也可以用于MIMD结构中。这种方法有时钟偏斜(cLock skew)处理问题。二是每个处理器采用单独的时钟。于是当两个时钟区之间传送数据时，就有时钟之间的裁决问题。三是通讯链路是由数据线与选通脉冲线组成时，尽管不同的数据路径有不同的时钟，但每个通讯路径是同步操作的。这样就避免了前两种选择的问题，但需要额外的选通脉冲线。同步模型要考虑最坏执行时间，而异步模型的不确定性带来了程序设计的困难。两者之间的折衷就是所谓SC(synchronized communication)模型。在SC模型中，算法是由完成计算或者通讯的相继阶段所组成。这两个阶段是明确区分的，计算与通讯从不在同一时间内发生。如果处理器的控制是分布的，则在开始一个通讯阶段之前，某些同步是必要的。通讯由处理元PE通过执行其局部存储器中的指令或者由一个主控器的播送来处理。因为处理器仅借助通讯与其它处理器交互，SC模型是确定的，并保证给定一个单一的输入就有同样的输出。SC模型可以用于共享或分布存储器的体系结构，所有的SIMD结构，必须是SC模型的，但是，MIMD-SC模型也是可能的。许多SIMD计算的算法是属于SC模型的，值得惊奇的是，为MIMD计算机开发的许多算法，也是属于SC模型的。在本书第三章所讲的协处理MPP系统中采用了同步时钟模型，由于只在播送指令中出现最坏执行时间，而且是与阵列的大小相关，因此，在实现播送指令时，采用了可根据阵列大小改变频率的方法，从而使最坏执行时间对其它指令的执行没有影响。

对MIMD体系结构来说，任务之间的同步可以是显式指出的，也可以是蕴含在消息传递的通讯之中的。这里只讨论前者，后者将在通讯技术中叙述。作为显式指定的同步技术，这里主要讨论Valiant提出的BSP(bulk synchronous parallel)模型[16]。它与以局部同步事件概念为核心的actor模型以及数据流模型的基本差别在于它是以总体阻挡层同步(global barrier synchronization)的概念作为基本机构的；而且在BSP(以及PRAM与数据并行方法)中，通常有由程序设计者作出的严密的调度与次序管理(在数据流模型中程序设计者是不考虑这些问题的)。

BSP计算机由三部分组成：一是一个处理器存储器对的集合；二是以点对点方式传送消息的一个通讯网络；三是用于所有处理器或部分处理器有效阻挡层同步的一个机构。这里没有特殊的组合、复制与播送的机构。直接为这一模型编写的程序由阻档层操作

(barrier operation)分开的一些超步(supersteps)组成。每个超步包含处理器完成的一些局部计算与发送或接收一些消息。所有消息的投递是需要在下一个超步开始之前完成的。特别是一个处理器不能企图去读另一个正在调整数据的处理器,否则,这两个操作就要用一个阻挡层分开。通过采用随机路由(randomized routing),一个消息经过网络的发送与接收之间的一个可预知延迟是能实现的。然而,如果有许多处理器要同时访问在同一存储器模块上的数据,则由于请求的串行化处理,性能就要下降。解决的办法就是采用一个随机散列函数(randomized hash function)把每个数据放到一个随机选择的存储器模块上,于是,每个节点(从概率上讲)接收一个均匀数目的请求。通过从消息经过网络所形成的路径中有效地分离处理器的数据访问模式,随机路由可以去掉网络中潜在出现的任何热点。尽管理论上讲,对于 p 个节点,散列函数是一个 $\log_2 p$ 次的多项式,但实践结果表明,可以化简成一个低次的多项式,只占用一个近似不变的计算时间。已经证明,与数据到存储器的手工映射相比,采用这种散列函数仅会造成一个小的不变的效率损失。在算法中采用并行宽松度(parallel slackness),即建立比处理器更多的进程,可以有效地隐藏远程访问延迟,且提供用户算法的有效实现。BSP 模型的原本目的在于提供足够的信息,允许用户编写可剪裁与可移植的算法。

如果定义一个时间步是单个局部操作所要求的时间,也就是说,对局部保存的数据值的一个基本操作,则任何 BSP 计算机的性能可以用四个参数来描述:一是参数 P(处理器的数目);二是参数 S(处理器速度,也就是每秒的时间步数目);三是参数 l(同步周期,也就是相继同步操作之间的极小时间步数目);四是参数 g(在一秒内由所有处理器完成的局部操作的总数,用在一秒内由通讯网络传送的字的总数相除的结果)。参数 l 涉及网络的延迟,也就是在一个连续的消息传送情况下,访问非局部存储器所要求的时间,提供完成一条阻挡层操作的开销。参数 g 对应于非局部存储器访问的可行频率,提供机器带宽的开销。在 g 值较高的机器中,则必须使非局部存储器的访问频率更低些。

BSP 计算机按下列方式操作。计算由一个序列的并行超步(supersteps)组成,而每个超步是一个步的序列,后面接着的是一个阻挡层同步,在这里任何存储器访问有效。在每个超步时,每个处理器有一组要执行的程序或线程(thread)以完成下列事情:一是对超步开始时存放在局部存储器中的值,完成一些计算步;二是发送与接收对应于非局部存储器的读与写请求的消息。BSP 计算机是一个两级存储器模型。也就是说,每个处理器有它自己的实际的局部存储器模块;所有其它存储器是非局部的,而且是可以一种一致有效方式访问的。一致有效是指一个处理器对在另一个处理器存储器对中的一个非局部存储器元素读或写所占用的时间,应当是与值保存在那个实际存储器模块中无关的。算法设计者或程序设计者不需明白(当前所用通讯网络的具体互连结构中的网络局部性的)任何层次的存储器组织。相反地,通讯网络的性能只应当用其总体性质(例如,完成一个非局部存储器操作所要求的最大时间以及在任何时间能同时在网络中的这种操作的最大数目)来描述。在一个 BSP 算法中一个超步 S 的复杂性是这样确定的:设 L 是在 S 中由任何处理器执行的最大局部计算步数目,$h1$ 是在 S 中由任何处理器发送的最大消息数目以及 $h2$ 是在 S 中由任何处理器接收的最大消息数目。于是 S 的开销是 $\max\{l, L, gh1, gh2\}$ 时间步(另一种办法是取超步的 $\max\{l, L+gh1, L+gh2\}$ 时间步,这两种开销的差别一般不是很大

的)。当是小的 g 时,比如,$g=1$,BSP 计算机严格地对应于后面将要讲到的一个 PRAM,l 决定实现最优效率所要求的并行宽松度(slackness)的程度。$l=g=1$ 的情形对应于理想的 PRAM,不要求并行宽松度,在设计大 g 值 BSP 计算机的算法时,需要通过利用两级存储器中线程局部性测量通讯宽松度,也就是说,必须保证对每个非局部存储器的访问,能对局部数据近似地完成 g 个操作。为了实现 BSP 模型中的体系结构无关性,设计并行算法时不仅要采用问题大小的参数 N,处理器数目的参数 P,还要采用参数 l 与 g。在 BSP 计算机中采用参数 l 与 g 刻画通讯性能与当前市场上大多数分布存储器体系结构描述通讯性能的方法是不大相同的,通常要讲到许多网络的局部性质,比如,每个节点的通道数目、通道的速度、网络的图结构等等。这些描述方法强调网络的局部性质,而不是它的总体性质,反映这些机器的大多数设计是适用于采用网络局部性工作方式的。BSP 模型的主要特征是它把网络性能的考虑从局部提高到整体上。这样,就不必再特别关注网络是一个二维阵列、一个蝶式网或一个超立方网,也不必关注它是用 VLSI 或光学技术实现的。而只须关注网络的总体参数,比如,l 与 g,以描述按一致有效的方式支持非局部存储器访问的能力。

尽管 BSP 计算机是作为一个体系结构模型描述的,也可以把块同步(bulk synchrony)当作程序设计模型,或者,实际上是一种程序设计方法学。BSP 方法的实质是超步的概念以及与超步相关的输入输出(或者说读写)是作为一个包含一组单个发送与接收的整体操作来完成的概念。按这种观点,一个"BSP 程序"就是一个分阶段(phase)处理的程序,在阶段之间进行必要的总体通讯。所以,BSP 途径可以看作一个程序设计方法学,可用于所有种类的并行体系结构。例如,共享存储器的多处理机,分布存储器的体系结构或者工作站的网络(在这些不同形式体系结构中的 g 值将自然是变化很大的)。看来,BSP 途径能为在将来可能出现的许多并行体系结构上开发可移值的并行软件提供一个一致的而且非常通用的基础构架。

支持分布存储器体系结构上的单一地址空间是在共享地址空间中开发基于细粒度并发的通用并行计算的一个主要问题。现在的大多数分布存储器体系结构是基于常用的微处理器的。能同时支持非常多的轻量线程(lightweight threads),而且提供快速上下文转换、消息处理、地址变换、散列等等的另外的处理器设计是需要的,如果没有这种设计,则会发现系统瓶颈将是处理器,而不是通讯网络。

大型通用并行计算机系统在运行时也会有各种硬件故障,应开发有效的技术为处理器、存储器以及通讯链路提供一定程度的故障容错能力。解决此问题的一个有益的途径就是采用信息散开(dispersal)的概念,利用数据的一个空间有效冗余编码以提供安全与可靠的信息存储以及有效的消息故障容错路由。尽管通过散列(hashing)的自动存储器管理的潜力是 BSP 模型的一个主要优点,但 BSP 算法的设计者可能希望保留控制两级存储器中的存储器管理,以达到更高的效率。直接的块同步算法的系统研究仍在进行。

程序设计语言与方法学也许是并行体系结构中最难解决的问题之一。除了 BSP 途径之外,最保守的另外办法就是 SIMD 或者说数据并行性,尽管对这种体系结构已开发了许多有用的并行算法,但它的模型是不够通用的,即使扩展成 SPMD 形式也会如此。另一个保守的办法就是继续用基于固定通道数目上消息传递的体系结构。尽管这种模型适合于

开发许多专用的并行系统以及低层系统的程序设计，它也不能提供充分的体系结构无关性。

BSP 模型是作为一个桥梁模型提出来的。因此，它一方面必须方便程序设计，使软件人员能长时间的接受这个模型；同时，另一方面对硬件人员，它必须是充分强有力的，能不断地提供模型的更好实现。基于 BSP 模型的计算机可按多种风格进行程序设计，但一个 PRAM 语言可能是理想的。程序将具有所谓并行宽松度(parallel slackness)，意思是为 V 个虚拟处理器设计的程序在 P 个具体处理器上运行，$V>P$。为了使编译程序能够将编辑与组装总体有效地作为执行所谓超步的指令块，这是必要的。采用 PRAM 模型作为程序设计模型的程序通常具有一个大的并行宽松度。还有，下面将会讲到，带 CTP(compiler-time padding)的编译程序将产生程序各部分带有执行时间信息的程序。这正是装配指令块所需要的信息。

传统上讲，同步是用来使处理器之间实施一个具体的排序(ordering)，从而解决竞争情况。通常，这样的竞争情况不仅仅涉及两个，而是多个或者全部处理器。阻挡层同步是用来解决这种竞争情况的办法之一。阻挡层同步 BS(barrier synchronization)这个术语是 1978 年在文献[1]中提出的，它是管理并行处理器的一种重要机构，对其硬件或软件的有效实现已有许多研究工作。假定在每个阻挡层同步中，可以是对机器中处理器的任意子集合进行的。因此，在一个阻挡层同步中处理器通常要完成下列三步工作：一是标志该处理器已达到此阻挡层；二是等待所有参加的处理器到达此阻挡层；三是在所有参加的处理器均已到达此阻挡层后，而且为检测这个条件的小的固定延迟时间后，所有参加的处理器超越此阻挡层同时恢复运行。这种作法就是将 VLIW 与 SIMD 机的静态指令调度性质扩展到 MIMD 领域。因此，许多同步就可以在编译时解决，而不必采用运行时同步机构。同时，也为 SIMD 机中常用的同步通讯网扩展到 MIMD 机中提供了可能性。

在文献[2]中是用一个阻挡层处理器(barrier processor)产生阻挡层屏蔽码，以决定参加一个具体阻挡层同步的处理器子集合。阻挡层处理器把阻挡层屏蔽码送到阻挡层同步缓冲器中，以保存每一个屏蔽码直到它被执行。每个处理器用一条到阻挡层同步缓冲器的 wait 线，指明一个具体的处理器是参与到一个阻挡层同步的。

为叙述方便起见，设有经过 4 个处理器的 5 个阻挡层要执行，如图 1.1 中所示。头两个阻挡层，分别经过处理器 0、1 与处理器 2、3，可以任何次序执行，或者说它们是不排序的。阻挡层的屏蔽码如图 1.1(a)右边所示，屏蔽码中为 1 的位对应于一个参加的处理器。在这里假定经过处理器 0、1 的第一个阻挡层是首先执行的，其余四个阻挡层是顺序执行的。但是，如果在运行时，经过处理器 2、3 的第二个阻挡层实际上是准备好首先执行的，则称它在执行之前是被阻塞的(blocked)。

屏蔽码是一个位向量，用 mask 表示，每个处理器对应于一位。$\mathrm{mask}(i)=1$ 时，表示处理器 i 是参加该具体的阻挡层同步的。而当 $\mathrm{mask}(i)=0$ 时，则表示处理器 i 是不参加该具体的阻挡层同步的。由于只检查所有参加的处理器是否都已到达该阻挡层，故表示这种情况的逻辑方程为：

$$\mathrm{GO} = \prod_i \overline{(\mathrm{mask}(i) + \mathrm{wait}(i))}$$

其中 $\mathrm{wait}(i)$ 表示参加该阻挡层的处理器已发出 wait 指令，表示到达该阻挡层，$\overline{\mathrm{mask}(i)}$

表示不参加该阻挡层的，wait 指令信号是忽略不计的。按照这种约定，图 1.1(a)硬件的实现如图 1.1(b)所示，阻挡层的屏蔽码是放在一个缓冲队列寄存器中。

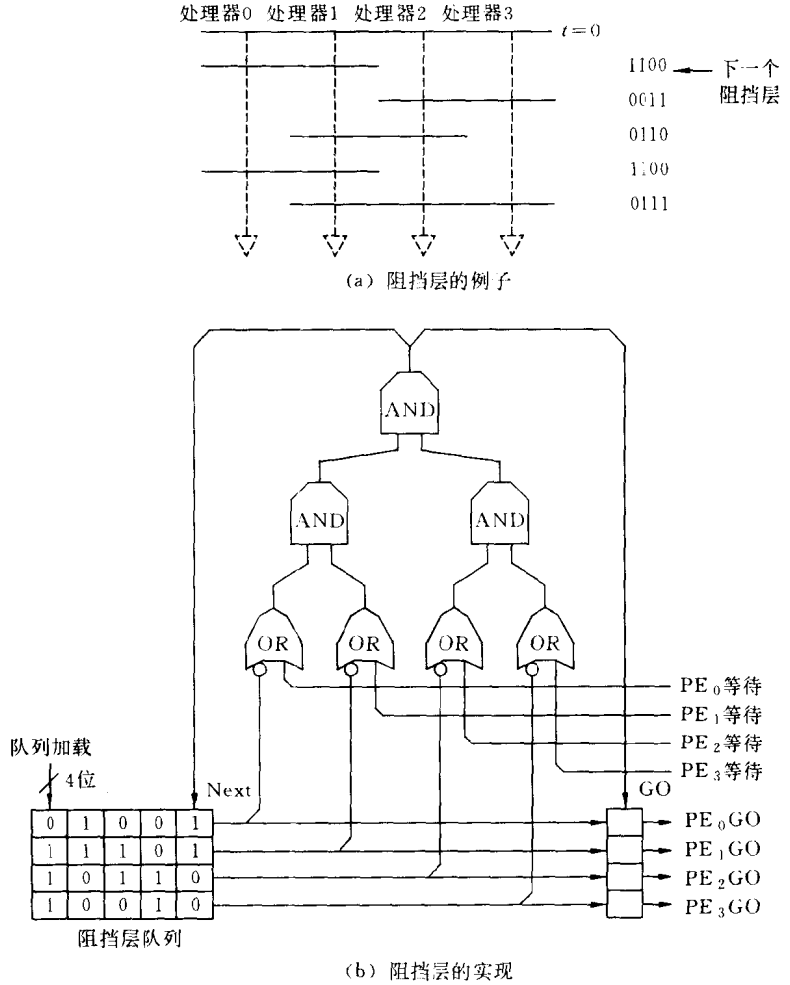


图 1.1 静态阻挡层的硬件实现

这种方法的最明显缺点就是先假定一个阻挡层的排序，它对应于所希望的阻挡层的排序，但这往往不是运行时阻挡层到达的次序，于是引入阻塞延迟。例如，图 1.2(a)中的三个阻挡层，就可能有图 1.2(b)中的三种实际完成次序，如果事先排定的次序与实际的完成次序不同，就要产生被阻塞延迟的情况。

文献[2]中已经证明，对于有 n 个阻挡层的情形，静态阻挡层实现方法的阻塞函数由下列递归方程给出：

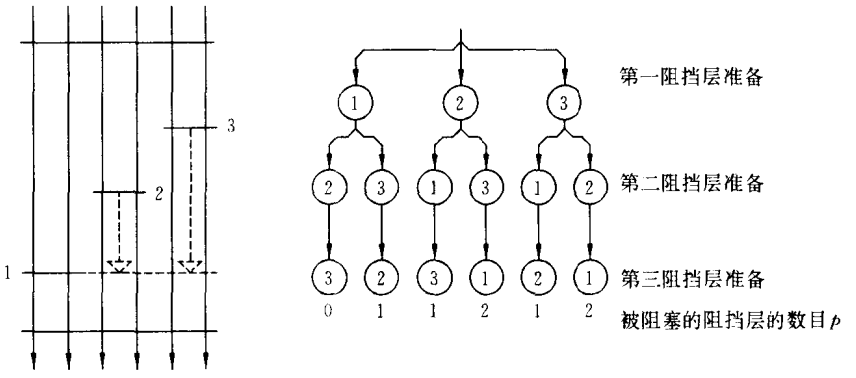
$$k_n(p)=\begin{cases}0, & if \quad p<0 \ or \ p\geqslant n\\ 1, & if \quad p=0\\ k_{n-1}(p)+(n-1)k_{n-1}(p), & if \quad 1<p<n\end{cases}$$

如果令阻塞商(blocking quotiet)$\beta(n)$是在有 n 个阻挡层中被阻塞的百分比的期望值，则

有下列方程式成立：

$$\beta(n) = \sum_{p=0}^{n-1} p \frac{k_n(p)}{n!}$$

按照这个结果，当 n 大于 11 时，则有 80%以上的阻挡层将被阻塞延迟；对于较小的阻挡层数目，百分比要小些。当 n 在 2 与 5 之间时，少于 70%的阻挡层将被阻塞延迟。因此，需要通过超排序的阻挡层执行才能减小阻塞商。



(a) 阻挡层的被阻塞延迟　　(b) 阻挡层的可能排序表示

图 1.2　阻挡层的可能次序

从图 1.2(a)可以看出，如果将其三个阻挡层合并成一个阻挡层，则阻挡层 2 与 3 最快也只能与阻挡层 1 同时完成，因此，不是最有效的。最有效的办法就是使这三个阻挡层可以超次序的执行。在图 1.1 的实现中，阻挡层是放在队列中的，每次只有一个阻挡层能参与执行。为了实现超次序的执行，则就要使多个阻挡层能同时参与执行。为此，在文献[2]中提出了一种用联想存储器实现的办法，如图 1.3(a)所示。由于在联想存储器中的所有阻挡层可以同时参与执行，从而可以实现超次序的执行。也就是说，放到联想存储器中的阻挡层必须是不排序的。未放到联想存储器中的阻挡层是不参与执行的，一直要等到它前面的阻挡层已经执行，并且从联想存储器中移走之后。

联想存储器的大小是决定有效程度的参数。按照文献[2]中的论证，设联想存储器的大小为 b，在图 1.3 实现中作为有 n 个阻挡层中被阻塞的百分比的期望值的阻塞商 $\beta^b(n)$，由下列方程给出：

$$\beta^b(n) = \sum_{p=0}^{n-1} p \frac{k_n^b(p)}{n!}$$

它说明当 b 值愈大时效果愈好，如图 1.3(b)所示。

分布的硬件实现就是将图 1.1(b)的阻挡层队列以及实现超次序执行的逻辑等分布在每个处理器中实现。从图 1.1(a)的例子中可以看出，阻挡层的超次序执行只对其前两个阻挡层是可行的，而对其后三个阻挡层来说是不可行的，因为它们同时要用到处理器 1。换句话说，只有不用到相同处理器的阻挡层才能是可以超次序执行的。这样一来，这些可以超次序执行的阻挡层不管有多少，如果能给它们加以编号(而不是排序)区分的话，则总是可以在一个屏蔽码中表示的。最多是这个屏蔽码中全是 1，剩下的是要能区分那些 1