# 多线程计算模型、体系结构与编译技术*)

林海波　汤志忠
(清华大学计算机科学与技术系　北京 100084)

## Multithreaded Computing Model, Architecture and Compiling Technique

LIN Hai-Bo　TANG Zhi-Zhong
(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)
E-mail: linhaibo99@mails.tsinghua.edu.cn　http://www.tsinghua.edu.cn

**Abstract**　Multithreading has been proposed as an efficient computing model for improving parallelism. It combines advantages of both dataflow architecture and von Neumann architecture, leading to high performance and efficiency. The-state-of-the-art multithreaded computing model includes Blocking thread and Non-blocking thread, the corresponded multithreaded architecting can be classified as Multiple Context Processor and Hybrid Architecture. Thread partitioning is one of the most important compiling issues in multithreaded computing. The idea of multithreading will be developed further on the move of architecture, compiling technique, and operating system.

**Keywords**　Multithreading, Architecture, Compiling technique

## 1　引言

在过去的30年里，计算机体系结构经历了长足的发展，超标量、超流水线、VLIW等先进思想显著地提高了计算机的性能，但这些单线程的体系结构在提高指令级并行度方面也面临着巨大的困难。多线程体系结构被认为是一种提高并行度的有效模型，它结合了数据流结构和传统的冯·诺依曼控制流结构，既保持了指令执行的高性能，又实现了处理器的高效率，是一种通用而高效的延迟隐藏技术。通过多个控制流，处理器可以很方便地开发线程级的并行性，这种方法与紧耦合的片内多处理器有些相似。与超标量或VLIW结构相比，多线程结构可以更灵活地开发线程级的并行性；与多处理器结构相比，多线程结构支持更先进的数据通信和同步机制以开发更细粒度的并行性。另外，多个线程可以共享功能部件，提高利用效率。

微电子技术的发展将使得在一个芯片内集成10亿个晶体管成为可能，如何有效地利用这些硬件资源成为当前的一个研究热点，这也使多线程体系结构引起了广泛的兴趣。本文介绍多线程计算模型、体系结构以及相关的编译技术，并讨论了多线程体系结构新近的发展现状及对未来的展望。

## 2　多线程计算模型

多线程计算是通过使用多个程序计数器并发执行程序不同位置中的指令来加速程序的执行过程的。有些体系结构提供在运行时创建和运行线程的功能，也有些体系结构在编译阶段对线程进行划分和优化。大部分多线程执行模型都需要指令集体系结构(ISA)的支持，以减少动态创建线程的软件开销。

根据线程的特点，多线程计算可广义地分两大类[1]：

可阻塞线程(Blocking Thread)：在这种模型中，线程在执行过程中可以被阻塞或挂起，并可被恢复，这需要一种保存线程状态并在多个状态中进行切换的机制。例如，当线程发出远程访问请求或需要同步操作时，它将被挂起。此模型所需的硬件支持可以简单至将挂起的线程放入队列中，也可复杂至将整套寄存器保存起来。线程的大小也可以从10条到上万条指令不等。这种多线程计算模型的例子包括HEP、Tera MTA、Dash、IHA和J-Machine等。

非阻塞线程(Non-blocking Thread)：在这种模型中，线程一旦开始执行，将不间断地一直执行到结束为止。这对线程增加了两个限制条件：一是在线程开始执行之前所有的输入必须全部就绪；二是所有的指令必须在确定的时间内完成。第二个条件隐式地要求对于长时延的操作(如远程访问指令)，其生产者和消费者必须在不同的线程中。这种计算模型需要的硬件支持较少，但线程的大小也受到限制。这种多线程计算模型的例子包括Monsoon、* T、EM-4、TAM、EARTH、S-TAM和Pebbles等。

根据线程的产生和执行方式，多线程计算又可分为以下4类：

可变指令流(Variable Instruction Stream)：这种模型是Wolfe和Shenused在XIMD体系结构中提出的。XIMD是对VLIW的一种扩充，它可以将功能部件划分到一条或多条指令流中。而且指令流的个数可以在运行时根据编译器产生的模式动态地变化。与VLIW相似，XIMD指令也被分为几个域，每个域对应一个功能部件。每个域包含一个数据操作(在此功能部件上执行的操作)和一个控制操作(指明在此功能部件上执行的下一条指令)。每个功能部件有单独的取指逻辑，以支持多个指令流。

主/从线程(Master/Slave Threads)：在这种模型中，程序由一个主线程开始，由主线程创建一个或多个子线程，子线程继承了主线程的状态，执行与主线程独立的代码。主线程和子线程在不同的处理单元或功能部件上并发执行。当子线程结束时，它将被合并到主线程中。这种多线程计算模型的例子包

*)基金项目：国家自然科学基金资助项目(60173010)。林海波　博士生，主要研究领域为体系结构，编译技术。汤志忠　教授，博士生导师，主要研究领域为计算机并行算法，并行编译技术。

括 M-Machine、SPSM 和 SSMT[2]等。

前驱/后继线程(Predecessor/Successor Threads):在这种模型中,编译器将程序的数据流图划分为多个线程的序列。在运行时,程序的头线程被分配在某一处理单元上执行,它再在其他的处理单元上创建出它的后继线程,后继线程又可以在另一个处理单元上创建自己的后继线程,以此类推。与主/从线程模型不同的是,前驱/后继线程模型在头线程结束后,它的直接后继线程成为新的头线程,其状态代表当前程序的执行状态。

这种模型有两个主要的优点:首先它很自然地支持线程级的猜测执行,因为程序的执行状态总是由头线程所表示,当猜测执行的后继线程不正确时,处理器只需要重新开始执行它即可,不需要进行现场恢复;其次,线程的执行顺序使得处理单元可以用一条单向的环连接起来,因为数据总是由头线程向其后继线程流动的。这种多线程计算模型的例子包括 Multiscalar 和 SMA[3]等。

动态线程(Dynamic Threading):这是最近提出的一种在运行时动态划分线程的计算模型。它不是通过编译器将源程序转换成多线程的程序,而是将顺序的可执行代码作为输入,通过在运行时动态分析控制流和数据流来产生线程。这种方法使得不需重新编译就可以开发现有程序的并行性,但进行动态分析的硬件开销是非常大的,而且受到资源的限制,动态分析的范围通常比较小,这将影响开发线程级并行性的能力。

## 3 多线程体系结构

我们把从硬件上支持多线程计算模型的处理器体系结构称为多线程处理器体系结构,简称多线程体系结构。构造一个多线程体系结构有两种方法:一种方法是在冯·诺依曼控制流结构上扩充对多线程并发的支持,称为多现场处理器;另一种方法是在数据流结构上扩充顺序执行的线程功能,称为数据流/冯·诺依曼混合结构。

### 3.1 多现场处理器

多现场处理器直接在传统的冯·诺依曼单处理器结构中引入对多个线程并发执行的支持(如多个硬件现场,高速的硬件现场切换机制等等)[4]。

多现场处理器的最新发展是将多线程和超标量技术结合,称为多线程超标量结构,它的主要目的是提高超标量处理器中功能部件的利用率,开发更大的指令级并行性;以及利用多线程的延迟隐藏特性,隐藏存储延迟、长浮点操作延迟等各种引起流水线阻塞的延迟,提高多个线程的整体性能。

Tullsen 的 SMT(Simultaneous MultiThreading,并发多线程)[5]是多线程超标量中最有代表性的研究。它允许多个线程竞争功能部件,当执行程序时,它可以交替开发程序的指令级并行性和线程级并行性。如果程序只有一个线程,SMT 所有的功能部件就被用来开发这一个线程的并行性;如果存在更多的线程时,SMT 就可以通过提高线程级并行度来弥补单个线程指令级并行度低对功能部件造成的浪费。所以,SMT 更能充分有效地利用处理机的功能部件,从而获得更高的加速比。如图 1 所示。

SMT 的结构是从一个高性能、乱序发射的超标量体系发展而来的,它的动态调度核心类似于 MIPS R10000,如图 2 所示。为了支持多线程,需要在原超标量结构的基础上增加一些功能部件:线程状态字(寄存器和程序计数器),流水线清空、指令回收、陷阱、中断和函数返回的线程控制,以及转移目标缓冲区和旁视缓冲区的线程标识。取指部件和指令流水线被重新设计,以支持多线程的指令发射。

SMT 对超标量的扩充并不大,但却可以得到较大的性能提高。由于它可以利用线程间并行性弥补线程内并行性的不足,所以它受分支预测、预测执行、存储带宽等的影响不大,但取指带宽和寄存器访问延迟会成为 SMT 的瓶颈。

关于并发多线程方面的研究工作还有很多。Gulati 和 Bagherzadeh 以超标量为基础实现了并发多线程结构。与 SMT 相对比,他们的处理器的指令发射带宽为 4,功能部件也少一些,这就降低了它的性能。

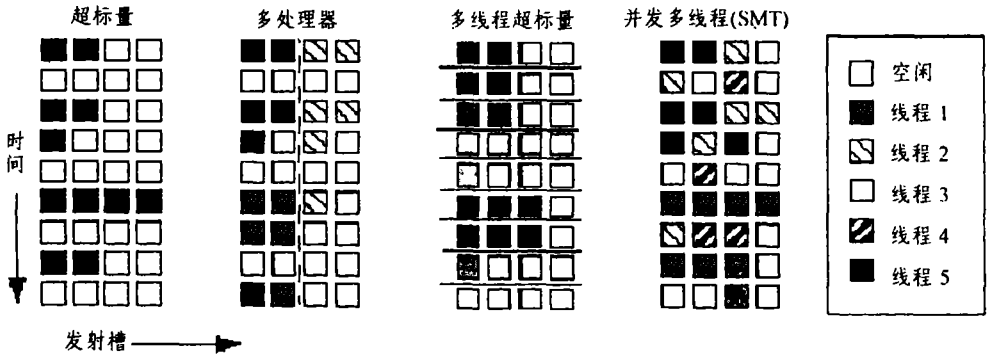


图 1 超标量、多处理器、多线程超标量与 SMT 结构的对比

Prasadh、Wu,以及 Keckler、Dally 基于 VLIW 设计了一种新的体系结构,其中多个线程中的 VLIW 指令可以动态地交错执行。

Govindarajan 等、Gunther、Beckmann 和 Polychronopoulos 提出在线程间分配指令发射带宽,每个周期只能从一个线程中发射一条指令,这种体系结构缺少灵活性,在只有一个线程可用时资源的浪费非常严重。

### 3.2 数据流/冯·诺依曼混合结构

混合结构是纯数据流结构的发展。每个线程是数据流图的一个子图或一个顺序执行序列。一旦线程的第一条指令开始执行,其后的指令就不中断地执行下去,即属于 Non-blocking Threading 模型。线程代替了指令作为数据流模型中的调度单位。通过在数据流中引入指令顺序执行机制,克服了数据流模型通信开销太大和无法实现关键段等灵活控制的缺点,同时又保持了数据流模型可以开发极大的并行性和隐藏延迟的作用。

混合结构最成功的例子包括 MIT 的 P-RISC、IBM 的 Empire 和 MIT/Motorola 的 *T。P-RISC 是 RISC 式的数据流系统结构。它允许数据流图较紧凑地编码并且产生较长的线程,以便获得较好的性能。这是通过把"复杂"的数据流指令

分解成使编译器可以控制的、独立而又"简单"的指令分量来实现的。它使用传统的指令序列，通过存储器完成所有的处理机内部的通信，并显式地使用存储单元实现连接。

*T[6]是一个多线程的 MPP 系统原型机，它是一个单地址空间系统，16 个结点装配在一个立方体中。本地网络用 8×8 交叉开关芯片构成，存储器分布在各个结点上。*T 的每个结点用 4 个部件实现，如图 3 所示。经修改的 Motorola 超标量 RISC 处理器作为数据处理器(data processor，DP)，它适宜于处理长线程，每个 DP 可以并发地执行整数和浮点操作。同步协处理器(synchronization coprocessor，SP)作为专用功能部件(SFU)，它适宜于处理简单的短线程。DP 和 SP 都能够处理快速加载。DP 处理新来的程序，而 SP 处理新来的消息、远程加载/远程存储的回答以及消息或同步的连接。存储控制器除了管理结点存储器外，还负责处理远程存储器的加载或存储的请求。网络接口部件分别接收来自或发送去网络的消息。SP 是 DP 的片内 SFU。

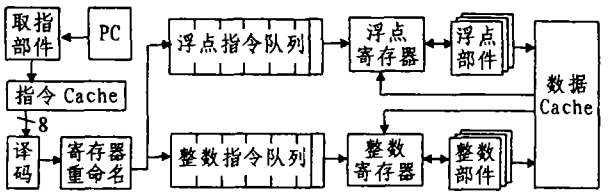


图 2　SMT 的处理器体系结构

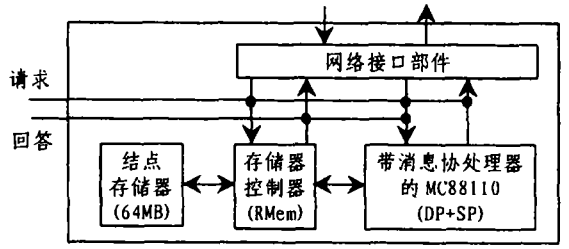


图 3　*T 的结点内部体系结构

线程现场由(IP、FP)对构成，存储在线程队列(continuation queue)中。DP 每次从队列中取出一个线程执行，每个线程执行不可中断。DP 扩充了 4 条指令：start、next、rload 和 rstore 用于启动、停止线程和远程存取。

新增的 SFU 能够提供 16 个用于输入消息的缓冲区和 4 个用于输出消息的缓冲区，它还可以存放 64 个项的程序堆栈和一个微线程调度器，它提供来自消息和程序堆栈的程序，并有专门的指令用于程序的打包和解包。

I-结构语义也在*T 上实现了。生产者-消费者变量使用了满/空位。*T 把消息作为虚拟程序来处理，因此忙-等待。*T的其他优化技术还包括通过多线程和一致性高速缓存来推测避免额外的加载和存储操作。

ETL/EM-4 是 Sigma-1 的扩充。其处理机的执行部件负责取指令，直到一个线程结束。令牌区匹配的指令被执行。指令可以发出令牌或写入寄存器。指令采用传统的顺序(PC+1 或转移)连续地取来，直到表示线程结束的"停止"标志出现，之后再接收另一对令牌。一个线程中的每条指令指出该线程下一条指令的两个源。

其他的混合结构还有很多。如 MIT 的 Monsoon 在纯数据流基础上(Tagged-Token Dataflow Architecture，TTDA)消去了令牌匹配，对等待令牌的存储器直接寻址。它采用多寄存器组支持多线程，并引入了基于线程的程序设计概念。

J-Machine 是一个由众多价格便宜、功能简单的处理单元组成的多机系统，但其消息处理器(MDP)以及集成的网络接口部件可支持快速的消息传递机制，以此进行线程间的通信。HEP/Tera 利用多现场来隐藏时延，每个时钟周期就完成一次现场切换，其每台处理机有 128 个现场，线程并必须很好地共享大量的寄存器，但它的单现场性能很低，实现成本和复杂性也很高。EARTH-MANNA 系统是一个由通用微处理器构成的多线程 MPA 系统，它具有与*T 类似的结构，但其每个结点由两个 Intel i860 构成，一个作执行单元(EU)，另一个作同步单元(SU)。EU 和 SU 之间的通信是整体性能的瓶颈。

## 4　多线程与编译技术

多线程思想推动了计算机体系结构的发展，同时也对编译技术提出了新的挑战。无论对于多现场处理器结构，还是数据流/冯·诺依曼混合结构，线程的划分，即多线程代码的生成都是编译器的一项重要工作。此外，多线程机制对于传统的编译优化技术如循环展开、寄存器分配等也有一定影响。

### 4.1　线程的划分

在多线程体系结构中，线程的大小直接影响到程序运行的性能：线程太小，则线程内可调度的指令太少，难以开发指令级并行度；线程太大，则线程间通信的数据交换太大，线程切换开销大，难以开发线程级并行度。Sarkar 在研究中指出，在给定通信延迟和函数调用开销的情况下，在多处理器上将程序划分为执行时间最优的多个非猜测线程是 NP 完全问题，所以，现有的线程划分方法都是某些条件下的启发式算法。

总体来说，线程划分的方法可分为两大类：自上而下(top-down)的方法和自下而上(bottom-up)的方法。自上而下的线程划分方法是指从程序的控制流图入手划分多个线程。因为它的分析对象是较高级的程序语言，所以这种方法较为容易。自下而上的线程划分方法是从细粒度的数据流图入手，将指令组合成线程。这种方法不受程序语言的结构限制，调度指令的范围更大，有利于优化。但它要求先生成数据流图，这实际上制约了它的应用范围。一般来说，自上而下的方法产生的线程要比自下而上的方法产生的线程大。在 Pebbles 系统中综合使用了这两种方法，编译器首先利用自上而下的方法产生线程，然后用自下而上的方法来增加线程的长度。

1. 自上而下的线程划分方法　大部分自上而下的线程划分方法基于宏数据流(macro-dataflow)和 MIMD 模型。宏数据流将程序表示为非循环的数据流图，并组成层次的结构。在 Sarkar 提出的方法中，他将处理器个数、操作延迟、通信开销等参数记入一个表中，通过测试给每个宏数据流图的结点标记执行开销，在编译时通过更进一步的分析来开发线程内的并行性，如对 FORALL 型循环的处理，在运行时根据一系列的启发算法对图进行划分，以达到并行度和通信开销的平衡。

Wolski 和 Feo 为 NUMA 多处理器提出了线程划分方法，他们在权衡执行开销的同时，考虑了变量的通信开销。Girkar 和 Polychronopoulos 在中间表示这一级开发循环间和函数间的并行性。

在 SMA 中，编译器将现场个数和流水线级数之和作为线程大小的参考(Thread_size = Context_num + Pipeline_

depth),并通过循环展开、函数内嵌和代码复制来增加线程内的指令条数。

在 EARTH-MANNER 中,研究者根据经典的列表调度算法提出了一种启发式线程划分方法。此方法通过在并行性能、时延隐藏能力、线程切换开销和顺序执行效率这 4 个方面寻找平衡来将指令划分到线程中去。

2. 自下而上的线程划分方法　自下而上的线程划分方法以数据流图的划分为基础,它们大都基于 Id 语言程序。

Iannucci[7]提出了相关集(dependence sets)的线程划分方法,数据流图根据以下标准划分为调度子(Scheduling Quanta,SQ)的集合:

- 最大化并行度:不损失过程间和循环间的并行度
- 最大化 SQ 的长度(关键路径的长度)
- 最小化动态同步(SQ 之间的弧)
- 避免死锁

当对同一个远程地址进行读写操作时,就可能产生死锁。相关集方法为每一条"输入"指令赋与一个唯一的名字,"输入"指令是任一个远程读操作的目标。一条指令的相关集就是不通过其他"输入"即可到达该指令的所有"输入"的集合。每一个相关集就对应着一个线程,从而避免了死锁。

Iannucci 的方法得到了更多的扩展。TAM 结构中采用了 merge-up 和 merge-down 方法来对相邻的线程进行合并,以得到更大的线程。Traub 等提出了与相关集类似的需求集(demand set)方法,它是以"输出"而不是"输入"作为集合。所谓"输出"是指将结果送至远程目标(远程访问或消息发送)的指令。这种方法可将某些不确定相关转化成确定性相关,为线程划分提供更充分的信息。

### 4.2　猜测执行

相当一部分体系结构利用多线程机制进行猜测执行,即用不同的线程来执行分支前后的指令,或不同支路的指令。

S3(Simultaneous Speculation Scheduling)[8]是结合了编译器和体系结构的特性来支持多路执行。它可被用于两路的分支预测和多个循环体的猜测执行。当遇到分支操作时,编译器对每路分支分别产生相应的线程,并行执行,当分支条件产生后,不被执行的那一个线程被取消。编译器不能解决的循环体间的数据相关性通过数据猜测执行来处理。体系结构需要支持两个或两个以上的线程同时执行,指令系统需要有管理线程的指令(fork,sync,wait)。

在 Slipstream 结构中,编译器跳过与程序执行路径无关的指令产生新的线程。被缩短的程序(Advanced Stream)执行速度快,但它带有猜测成分;因此需要原有的程序(Redundant Stream)与它并发地执行。A-Stream 将它的控制流和数据流传递给 R-Stream,通过程序的完整执行来检查其正确性;由于 A-Stream 提高了猜测的准确性,所以 R-Stream 的执行也更有效。

SMA 将 do-while 循环转换为 while-do 循环,并采用 Disjoint Eager Execution 来处理多路分支。

**结论与展望**　多线程计算的优点是显而易见的。它有效地结合了控制流和数据流的优点,一方面保持了线程顺序执行的高效率,另一方面又可隐藏时延、开发并行性。

多线程计算也暴露出一些自身的缺点。一个是硬件实现的复杂度高,资源开销和设计难度都会影响多线程结构的执行效率。另一方面是可编程性差,在混合结构中,一般需要特定的多线程编程语言,由程序员显式地划分线程;在多现场处理器结构中,则更多地需要编译器的支持。这些限制也指明了多线程计算的未来发展方向:

1. 多线程与体系结构。在现有多线程体系结构的基础上,有效地利用硬件资源,合理地分配功能部件。或者与其他的体系结构相结合,探索新的发展之路。

2. 多线程与编译技术。为了更好地利用多线程的硬件支持,应用程序需要具有多线程的特点,这一方面需要多线程的程序设计语言,另一方面更需先进的编译技术支持。我们的研究工作将主要集中在这一方面。

3. 多线程与操作系统。线程间的通信开销也是多线程计算的瓶颈之一。线程同步、负载平衡将对操作系统提出新的要求,当然这也需要体系结构或编译技术的支持。总之,多线程计算是硬件结构与软件环境相互配合的模型,只有合理地进行分工,才能充分发挥多线程计算的优势。

## 参 考 文 献


1　Lucas J R. Code Generations, Evaluations, and Optimizations in Multithreaded Executions: [Ph D dissertation]. Colorado State University, Fort Collins, CO, 1995

2　Chappell R, Stark J, Kim S, Reinhardt S, Patt Y. Simultaneous Subordinate Microthreading (SSMT). In: Proc. of the 26th Annual Intl. Symposium on Computer Architecture. 1999. 186～195

3　Deng K, Dou Y, Zhou X M. Compiler Support for Speculative Multithreaded Execution. In: Workshop on Compiler for High Performance Processor System. 2001. 37～44

4　Hwang K. Advanced Computer Architecture: Parallelism, Scalability, Programmability. New York: McGraw Hill, 1993

5　Lo J, Eggers S, Emer J, Levy H, Stamm R, Tullsen D. Converting Thread-Level Parallelism Into Instruction-Level Parallelism via Simultaneous Multithreading. ACM Transactions on Computer Systems, 1997, 15(3): 322～354

6　Nikhil R S, Papadopoulos G M. *T: A Multithreaded Massively Parallel Architecture. In: Proc. of the 19th Annual Intl. Symposium on Computer Architecture. 1992. 156～167

7　Iannucci R A. Toward A Dataow/Von Neumann Hybrid Architecture. In: Proc. of the 15th Annual Intl. Symposium on Computer Architecture. 1988. 131～140

8　Ungerer U T, Zehendner E. A Combined Compiler and Architecture Technique to Control Multithreaded Execution of Branches and Loop Iterations. In: The 4th Annual Workshop on Interaction between Compilers and Computer Architecture (INTERACT-4). 2000