第 34 卷 第 15 期 计 算 机 工 程 2008 年 8 月
Vol.34 No.15 Computer Engineering August 2008
·开发研究与设计技术· 文章编号：1000—3428(2008)15—0283—03 文献标识码：A 中图分类号：TN911.72

多线程处理器资源分配策略

何 军，王 飙
(国家高性能集成电路设计中心，上海 201204)

摘 要：处理器资源如何在多个线程之间进行分配和共享是直接影响多线程处理器性能的关键问题。该文总结 4 种分配模型，提出其实现机制，讨论资源分配平衡问题，指出可根据目标应用和流水线不同阶段的特点，在各流水线阶段综合采用不同分配模型和实现机制，实现处理器资源的合理分配。

关键词：多线程处理器；资源分配；分配策略

Resource Allocation Policies for Multithreading Processor

HE Jun, WANG Biao
(National High Performance IC Design Center, Shanghai 201204)

【Abstract】 The problem of how to allocate and share processor resources among multiple threads of multithreading processors has a direct effect on the performance of the processor. Four allocation models and the mechanism for implementation are brought out in this paper, and the balance of resource allocation is discussed. It points out that different allocation models and implementation mechanisms can be applied to different pipeline stage according to such factors as the character of the object application and pipeline stage, and so on, to allocate the processor resources reasonably.

【Key words】 multithreading processor; resource allocation; allocation policies

1 概述

现代超标量处理器一般通过增加发射宽度和提高时钟频率来提高处理器性能，利用超标量发射、乱序发射执行、超级流水、动态转移预测、大容量片内 Cache 等技术来开发程序的指令级并行性(Instruction Level Parallelism, ILP)。随发射宽度的增加，超标量处理器的设计复杂性也极大增加，由于指令间的数据与控制相关，因此限制了通过开发 ILP 提高性能的可行性。随着互联网的普及，商业网络应用日益广泛，例如在线事务处理、决策支持系统和 Web 服务等。这类应用的特点是线程级并行性(Thread Level Parallelism, TLP)较高而 ILP 较低，而传统超标量处理器结构难以开发 TLP。因此，采用多线程处理器开发越来越复杂的 TLP，以提高处理器的吞吐量和资源利用率是较好的方法。目前主要有 4 种多线程结构(见表 1)：细粒度多线程(Fine-Grained Multi-Threading, FGMT)、粗粒度多线程(Coarse-Grained Multi-Threading, CGMT)、同时多线程(Simultaneous Multi-Threading, SMT)和单片多核(Chip Multi-Processor, CMP)。

表 1 多线程结构比较

多线程	线程切换机制	资源利用率	并行性开发	单线程性能	实现复杂性
FGMT	每时钟周期	较低	较高 TLP 低 ILP	差	低
CGMT	流水线停顿时	较高	TLP、ILP 较高	较好	较低
SMT	同时活跃，无切换	高	高 TLP 较高 ILP	好	高
CMP	同时活跃，无切换	较低	高 TLP 低 ILP	差	低

由表 1 可见，多线程结构在性能和实现复杂度上各有优缺点。经过深入分析可以发现，这主要由处理器资源在多个线程之间的分配和共享方式决定。FGMT 在每个时钟周期内从不同线程取指执行，多个线程共享除寄存器文件之外的所有处理器资源。CGMT 只在遇到流水线长时间停顿的情况下(如 Cache 失效)才进行线程切换，多线程共享除取指缓冲、寄存器文件和相应控制逻辑之外的所有处理器资源。上述 2 种多线程结构是分时共享处理器资源，但分时粒度不同。SMT 能在每个时钟周期内从不同线程发射指令到多个执行部件，允许多个线程在共享的执行资源中动态交叉执行，因此，要求为每个线程提供一组相应的上下文，包括取指缓冲、返回地址堆栈、寄存器文件、重排序缓冲、load/store 队列等，其他处理器资源被所有线程共享。SMT 中所有线程同时活跃，资源共享最灵活，相应的控制逻辑最复杂。CMP 通过简化单处理器核设计，将多个处理器核集成在同一芯片上，利用多个核分别开发 TLP。CMP 与上述几种模型最大的不同是它对处理器资源进行空间静态划分，即将处理器划分为多个相独立的处理器核，这些核的资源之间不存在共享，从而简化了设计，并能更好地适应工艺比例缩放的发展。由于多线程之间分配和共享处理器资源的方式不同，因此这些多线程结构在资源利用率、并行性开发、单线程性能和实现复杂性等方面各有优缺点。

多个线程之间如何分配和共享处理器资源是多线程处理器设计中的关键问题，将直接影响其设计实现的性能和复杂性。针对多线程处理器资源分配这一问题，本文从分配模型、实现机制和资源分配平衡 3 个方面进行研究，总结了 4 种分配模型，提出一般实现机制和关于资源分配平衡的基本原则，为多线程处理器设计提供了参考。

2 多线程处理器资源分配

2.1 分配模型

资源分配是一个经典问题，主要考虑如何解决效率与公

作者简介：何 军(1980－)，男，硕士研究生，主研方向：计算机体系结构，微处理器设计；王 飙，高级工程师

收稿日期：2007-10-12 **E-mail：**joyhejun@yahoo.com.cn

平的矛盾，避免饿死和死锁。在多线程处理器中，就是要保证每个线程都能获得处理器资源，并提高线程吞吐量和资源利用率。本文根据上述 4 种多线程结构，总结抽象出 4 种资源分配模型：静态分配，动态共享，有限共享和分时共享。图 1 以 2 个线程为例，展示了上述 4 种分配模型。

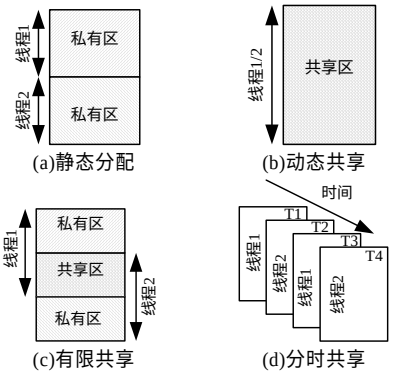

图 1　4 种资源分配模型

最直接的方法是静态分配，即为每个线程分配独立的私有资源，每个线程只能使用其私有资源，这些资源在线程间不存在共享。该模型的优点是将资源及其管理方式分布化，以简化资源管理控制，从而有效避免线延的不利影响，能更好地适应工艺可扩展要求。其不足是不能适应线程资源动态变化的需求，资源利用率不高，这种情况在执行单线程时最为明显。静态分配能保证公平性，不会出现饿死和死锁。

最理想的方法是动态共享，即所有线程共享整个资源，并根据不同线程在不同执行阶段对资源的不同需求，按需分配资源。此方式最大的优点是能充分适应线程资源动态变化的需求，资源利用率高，执行单线程时整个资源能被充分利用，最大限度地保证了单线程性能。其不足是资源管理复杂，尤其是线程较多时，资源大小随之增加，使资源管理的复杂性极大增加。这将导致设计复杂化，可能影响周期时间，降低整体性能。这种集中式复杂设计容易受到线延的不利影响，难以适应工艺可扩展要求。动态共享模型还需要解决效率与公平的矛盾机制，避免饿死和死锁。

有限共享模型是静态分配与动态共享的折衷，即为每个线程都分配一定私有资源，剩余部分由所有线程共享。这种模型在一定程度上缓解了动态共享设计复杂的不足，也在一定程度上改进了静态分配利用率不高的缺点，但需要仔细权衡每个线程私有资源的大小，即需要在开发 ILP 与 TLP 之间进行较好的平衡，尤其是线程较多时，否则可能无法有效保证单线程 ILP 并充分发掘 TLP。该模型能兼顾效率与公平，避免了饿死和死锁。

上述 3 种模型都从空间角度进行资源分配，分时共享模型则从时间角度来分配资源。分时共享模型与早期分时系统类似，各线程分时共享资源。多个线程可以按时钟周期轮转，也可以按不同线程优先级进行。其与上述 3 种模型最大的不同是，在某一时刻只有一个线程占用该资源。这种方式实现简单，需要考虑分时粒度和分时机制。分时粒度是进行线程切换的间隔时间；分时机制是切换线程的条件，最简单的方式是按周期轮转，也可以按某种条件动态触发。周期轮转能保证公平性，但效率不高；条件动态触发能提高效率，但需要保证公平性。

CMP 是静态分配模型的典型代表，它对大部分处理器资源进行静态分配。细粒度和粗粒度多线程是分时共享模型的代表，细粒度多线程采用时钟周期轮转的方式切换线程，粗粒度多线程根据线程是否发生 Cache 失效来决定线程的切换。理想的 SMT 结构应该是动态共享模型的代表，考虑到设计实现的复杂性，现有 SMT 结构一般在超标量结构基础上略加改进来实现。

以上 4 种模型都能在改进传统超标量结构基础上实现多线程。可以根据目标应用的特点、流水线不同阶段的特点、设计复杂性和性能等方面的因素，在流水线的不同阶段综合采用不同分配模型，实现处理器资源的合理分配。

2.2　实现机制

对于静态分配模型，由于资源静态分配给每个线程，各线程之间互相独立，因此只需要从逻辑或物理上分割资源即可。资源可以平均分配或非平均分配。对于周期轮转方式的分时共享模型，只要保证每个周期由不同的线程轮流占用即可。对其他分配模型需要考虑如何使用共享资源，同时避免饿死和死锁。

不同应用程序对处理器资源的需求通常不同，同一程序在执行的不同阶段对处理器资源的需求也不同。因此，共享资源的分配使用需要根据应用程序执行过程中的需求和特点来确定，以提高效率。这就需要跟踪记录每个线程执行过程中的一些动态信息，如取指缓冲、发射队列、重排序缓冲、load/store 队列利用率、转移预测失败率和 Cache 命中率等。根据这些信息或其中一部分来控制共享资源分配，确定处理器资源(如取指缓冲、发射缓冲、重排序缓冲、load/store 队列条目和取指、发射、提交带宽等)在多线程之间的分配与共享，如图 2 所示。上述执行信息的跟踪记录可以通过设置相应的计数器来实现。可以根据执行信息为线程划分相应优先级，然后根据优先级的权重，划分共享资源。文献[1]在 SMT 结构中采用 Icount 取指策略，根据发射队列的利用率来划分取指带宽，优先从占用发射队列条目最少的线程取指，极大提高了 SMT 性能(IPC)。

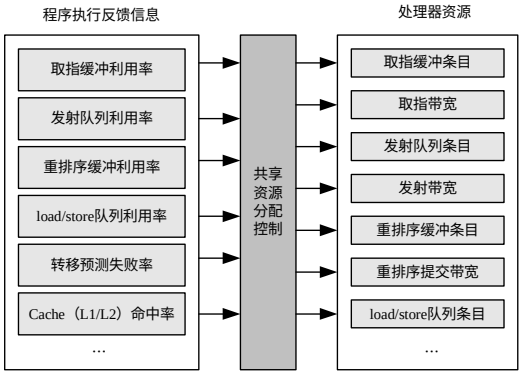

图 2　资源共享实现机制

在分配共享资源时，可以根据优先级在所有线程间划分资源。但在某些情况下，为了提高效率，无须为某些线程分配资源，这时需要屏蔽这些线程，在其他线程之间分配共享资源。为了保证公平性，避免饿死和死锁，或者为了提高性能，需要抢占某些线程的资源。例如，在处理器不命中 2 级 Cache 的 load 指令[2]时，可以将发生 2 级 Cache 失效的线程屏蔽，暂停从该线程取指，也可以直接抢占该线程的资源，将发生失效的指令后面的所有指令从流水线上清除，并暂停从该线程取指。文献[3]的研究表明，在 SMT 结构中综合运用这 2 种方式处理 2 级 Cache 失效，能更好地提高效率。

如上所述，共享资源的分配可以通过跟踪程序执行的动

态信息来反映程序对资源的需求，并根据需求动态分配，以提高效率。可以通过设置优先级、屏蔽或抢占等方式来实现。应根据追求的性能目标、设计复杂性和处理器具体参数配置等因素确定需要跟踪的执行信息。

2.3 资源分配的平衡

处理器资源分配是一个系统性问题，需要把握如下 3 个方面的平衡：

(1)效率与公平。效率是指处理器的吞吐量和资源利用率。上述 4 种分配模型在效率上有不同取舍。一般来说，能适应不同应用和应用不同阶段特点的灵活分配方式更有利于提高效率但可能不利于公平，而平均分配的方式能保证公平性却不利于效率提高。因此，应按照效率优先，兼顾公平的原则进行，避免饿死和死锁。

(2)数据通路资源之间的平衡。在处理器流水线数据通路上，不同资源对处理器性能的影响和作用不同，资源之间的相关度也不同，因此，需要针对不同资源特点采用相应分配模型，而且要联系起来考虑，避免出现瓶颈。对多线程结构来说，取指是关键，需要仔细考虑取指缓冲和取指带宽的分配，已有不少相关研究(参见第 3 节)。Joseph Sharkey 等研究表明，程序执行过程中具有不同特点，可能受限于发射资源或重排序资源，受限于前者时，即使增加重排序缓冲条目也不能提高处理器性能，反而会增加对共享的发射队列和寄存器文件的压力而使整体性能下降。

(3)设计复杂性与系统性能。复杂的分配策略可能影响周期时间，限制时钟频率提升，降低系统性能。虽然动态共享模型具有最大的灵活性和提高效率的潜力，但如果对所有处理器资源都采用这种方式，可能使资源分配控制管理复杂而无法达到预期目标。应该根据程序对资源的需求特点、该资源对性能的影响和实现代价来确定其分配模型。

需要根据应用目标和处理器结构参数进行处理器资源分配。可以根据目标应用的特点、流水线不同阶段的特点、设计复杂性和性能等方面的因素，在流水线的不同阶段综合采用不同分配模型和实现机制，实现设计目标。例如，可以对关键的取指部件采用动态分配模型以提高效率，多个线程之间动态分配取指带宽或分时共享取指带宽；对发射队列采用静态分配以简化发射逻辑，对重排序缓冲采用有限共享以取得某种平衡；发射带宽和提交带宽由多个线程动态共享以提高效率，或动态共享发射带宽、分时共享提交带宽以简化指令提交逻辑。

3 多线程处理器资源分配的相关工作

目前关于多线程处理器资源分配的研究主要是基于 SMT 结构进行的，其中不少集中于取指策略的研究。文献[1]提出的 Icount 策略优先从占用发射站台之前资源最少的线程(即具有高 ILP 的线程)取指，以减少发射队列阻塞。文献[4]在 Icount 策略基础上进一步研究对如何减少发射队列阻塞，并提出 4 种改进策略。Icount 策略对高 ILP 应用较有效，但在遇到长延迟 load 时效率不高，因此，D.M. Tullsen 等提出在遇到长延迟 load 时将该线程清除(Flush)以提高效率。文献[3]指出 Flush 策略仅适合与经常发生 2 级 Cache 失效的程序，对偶尔发生 2 级 Cache 失效的程序采用 Stall(即暂停从该线程取指)策略更好，并提出根据程序特点综合运用两者的方法——Flush++[3]。文献[5]综合考虑效率和公平性，对 3 种策略，即轮转法、Icout 和 LC_Bpcount 进行分析评估，提出综合运用这些策略以获得效率与公平性的平衡。文献[6]提出的动态资源分配策略(DCRA)较特别，它根据线程所处的不同阶段动态分配关键资源，并将更多资源分配给慢线程。

文献[7]研究了发射队列和重排序缓冲，指出发射队列和重排序缓冲条目采用静态划分或动态共享对性能影响不大，但发射带宽与提交带宽的静态划分对性能不利。文献[8]对静态划分的发射队列进行研究，提出线程内采用老指令优先，线程间根据每个线程就绪指令的数量来划分发射带宽的策略。文献[9]提出根据程序执行特点动态调整重排序缓冲分配的方法，并获得了性能提升。

本文主要从理论上对多线程处理器资源分配策略进行研究，不涉及具体实现，不受制于某种具体的多线程结构。多线程结构设计关键在于处理器资源分配，笔者从一般性理论角度出发，提出了 4 种分配模型，阐述一般的实现机制，并讨论资源分配平衡问题，希望可以实现处理器资源的合理分配，构建更好的多线程处理器结构，充分开发 ILP 和 TLP。

4 结束语

多线程处理器结构设计的关键是处理器资源如何在多个线程之间进行分配和共享，这与具体应用目标和处理器结构参数直接相关。下一步研究将结合具体处理器的结构参数，对不同资源分配模型和实现机制的组合进行性能评估及分析，为未来处理器设计提供借鉴和参考。

参考文献

[1] Tullsen D, Eggers S, Emer J, et al. Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor[C]//Proceedings of the 23rd Annual International Symposium on Computer Architecture. Philadelphia, USA: [s. n.], 1996.

[2] Tullsen D, Brown J. Handling Long-latency Loads in a Simultaneous Multithreaded Processor[C]//Proceedings of the 34th Annual ACM/IEEE International Symposium on Microarchitecture. [S. l.]: ACM Press, 2001.

[3] Cazorla F J, Fernandez E, Ramirez A, et al. Improving Memory Latency Aware Fetch Policies for SMT Processors[C]//Proceedings of the 5th IEEE International Symposium on High Performance Computing. Tokyo, Japan: IEEE Press, 2003.

[4] Moursy A E, Albonesi D. Front-end Policies for Improved Issue Efficiency in SMT Processors[C]//Proceedings of the 9th IEEE International Conference on High Performance Computer Architecture. [S. l.]: IEEE Press, 2003.

[5] Luo K, Gummaraju J, Franklin M. Balancing Throughput and Fairness in SMT Processors[C]//Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software. Arizona, USA: IEEE Press, 2001.

[6] Cazorla F. Dynamically Controlled Resource Allocation in SMT Processors[C]//Proceedings of the 37th ACM/IEEE International Symposium on Microarchitecture. [S. l.]: ACM Press, 2004.

[7] Raasch S E, Reinhardt S K. The Impact of Resource Partitioning on SMT Processors[C]//Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques. New Orleans, USA: [s. n.], 2003.

[8] Robatmili B. Thread-sensitive Instruction Issue for SMT Processors[C]//Proceedings of the IEEE Computer Architecture Letters. [s. n.]: IEEE Press, 2004.

[9] Sharkey J, Balkan D, Ponomarev D. Adaptive Reorder Buffers for SMT Processors[C]//Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques. Seattle, Washington, D. C. USA: [s. n.], 2006.