[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 9/46 (2006.01)

[12] 发明专利说明书

专利号 ZL 200410080084.0

[45] 授权公告日 2007 年 4 月 4 日

[11] 授权公告号 CN 1308825C

[22] 申请日 2004.9.24
[21] 申请号 200410080084.0
[30] 优先权
[32] 2003.9.25 [33] US [31] 10/671,057
[73] 专利权人 国际商业机器公司
地址 美国纽约
[72] 发明人 J·M·阿卡帕蒂 A·邓希
D·米歇尔 M·S·斯里尼瓦斯
[56] 参考文献
US2003135711 A1 2003.7.17
审查员 曲 颖

[74] 专利代理机构 北京市中咨律师事务所
代理人 于 静 杨晓光

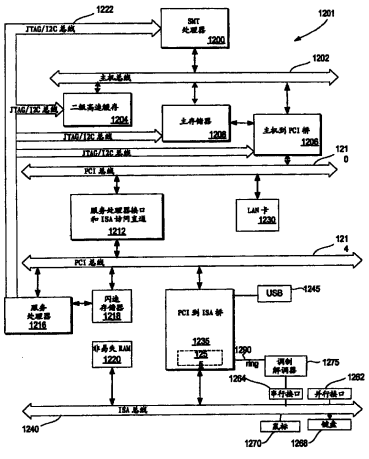
权利要求书 5 页 说明书 25 页 附图 12 页

[54] 发明名称

用于在 SMT 处理器中进行 CPI 负载平衡的系统和方法

[57] 摘要

本发明提供了一种在使用多个 SMT 处理器的同时多线程(SMT)处理器环境中调度线程的系统和方法。 识别出正运行在每个 SMT 处理器上的低性能的线程。 在被识别出之后，低性能的线程被移动到不同的 SMT 处理器上。 捕捉关于线程的性能的数据。 在一个实施例中，这个数据包括每个线程的 CPI 值。 当线程被移动时，记录关于该线程和它被移动时的性能的数据，以及一个时间戳。 关于以前的移动的数据被用于确定在移动之后线程的性能是否提高了。



1.一种为多个同时多线程处理器调度线程的计算机实施的方法，所述方法包括:

确定第一运行队列中的第一线程是一个低性能线程，其中第一运行队列相应于第一同时多线程处理器;

响应所述确定:

将与第一线程相应的第一标识符写入第二运行队列，其中第二运行队列相应于第二同时多线程处理器；以及

从第一运行队列中删除第一标识符。

2.如权利要求1的方法，还包括:

确定第二运行队列中的第二线程是另一个低性能线程;

响应关于第二线程的确定:

将与第二线程相应的第二标识符写入第一运行队列；以及

从第二运行队列中删除第二标识符。

3.如权利要求1的方法，其中所述确定还包括:

在第一同时多线程处理器上执行列于第一运行队列内的多个线程，包括第一线程，该执行还包括:

为每个线程获取一个周期数目值，它指示着当每个线程正在执行期间所发生的周期数;

为每个线程获取一个指令数目值，它指示着当每个线程正在执行期间所执行的指令的数目;

用每个周期数目值除以其相应的指令数目值，除得的结果是每条指令的周期值；以及

为第一运行队列中的每个线程记录每条指令的周期值；以及

识别所述第一线程为具有比列于第一运行队列中的多个其它线程差的每条指令的周期值。

4.如权利要求3的方法，还包括:

确定每个线程是否以前被从多个同时多线程处理器运行队列中的一个运行队列移动到了第一同时多线程处理器的运行队列中。

5. 如权利要求 4 的方法，还包括:

确定第一线程的每条指令的周期自从被移到了第一同时多线程处理器的运行队列之后，性能降低了。

6. 如权利要求 3 的方法，还包括:

在一个以前交换的数据结构中记录第一线程的标识符、第一线程的每条指令的周期和一个时间戳。

7. 如权利要求 6 的方法，其中所述时间戳相应于选自这样的一组的一个时间，该组包括计算第一线程的每条指令的周期的时间，和将第一线程的标识符从第一运行队列移动到第二运行队列的时间。

8. 如权利要求 3 的方法，还包括:

将用于多个线程中的每个线程的每条指令的周期值与以前为每个线程计算的一个或多个以前的每条指令的周期值平均，其中被记录的每条指令的周期值包括每个线程的平均每条指令的周期值。

9. 如权利要求 1 的方法，还包括:

跳过与第一线程相比的一个或多个更低性能线程，作为对确定每个所述更低性能线程以前已经被移动到第一同时多线程处理器的运行队列，并且它们每个自从被移动之后，性能都提高了的响应；以及

在跳过所述更低性能线程之后，识别第一线程。

10. 如权利要求 1 的方法，还包括:

检测到第一线程将要结束;

在第一线程结束时为其计算每条指令的周期值;

确定该每条指令的周期值比一个阈值差;

响应该确定，检查第一线程是否以前被从以前的同时多线程处理器移动到第一同时多线程处理器上，所述以前的同时多线程处理器选自包括第二同时多线程处理器的多个同时多线程处理器;

响应对第一线程以前被移动过的确定，确定第一线程在第一同时多线

程处理器上的每条指令的周期比它在所述以前的同时多线程处理器上的每条指令的周期差；以及

其中所述写入和删除步骤是响应对第一线程在第一同时多线程处理器上的每条指令的周期比它在所述以前的同时多线程处理器上的每条指令的周期差的确定而执行的。

11. 一种信息处理系统，包括：

多个同时多线程处理器；

处理器可以访问的存储器；

存储在存储器中的多个软件线程和多个运行队列，其中每个运行队列相应于同时多线程处理器中的一个；

线程调度工具，它用于在多个同时多线程处理器间调度线程，所述线程调度工具包括：

确定来自多个软件线程的第一线程是一个低性能线程、所述多个软件线程处于选自多个运行队列中的第一运行队列的部件，其中第一运行队列相应于选自多个同时多线程处理器中的第一同时多线程处理器；

响应所述确定、将相应于第一线程的第一标识符写入第二运行队列、其中第二运行队列相应于选自多个同时多线程处理器的第二同时多线程处理器、以及从第一运行队列中删除第一标识符的部件。

12. 如权利要求 11 的信息处理系统，其中所述线程调度工具还包括：

确定第二运行队列中的第二线程是另一个低性能线程的部件；

响应关于第二线程的确定、将与第二线程相应的第二标识符写入第一运行队列，以及从第二运行队列中删除第二标识符的部件。

13. 如权利要求 11 的信息处理系统，其中所述确定来自多个软件线程的第一线程是一个低性能线程、所述多个软件线程处于选自多个运行队列中的第一运行队列的部件包括：

在第一同时多线程处理器上执行列于第一运行队列内的多个线程、包括第一线程的执行部件，该执行部件包括：

为每个线程获取一个周期数目值、它指示着当每个线程正在执行期间

所发生的周期数的部件;

为每个线程获取一个指令数目值、它指示着当每个线程正在执行期间所执行的指令的数目的部件;

用每个周期数目值除以其相应的指令数目值、除得的结果是每条指令的周期值的部件; 以及

为第一运行队列中的每个线程记录每条指令的周期值的部件; 以及

识别所述第一线程为具有比列于第一运行队列中的多个其它线程差的每条指令的周期值的部件。

14. 如权利要求 13 的信息处理系统, 其中所述线程调度工具还包括:

确定每个线程是否以前被从多个同时多线程处理器运行队列中的一个运行队列移动到了第一同时多线程处理器的运行队列上的部件。

15. 如权利要求 14 的信息处理系统, 其中所述线程调度工具还包括:

确定第一线程的每条指令的周期自从被移到了第一同时多线程处理器的运行队列之后性能降低了的部件。

16. 如权利要求 13 的信息处理系统, 其中所述线程调度工具还包括:

在一个以前交换的数据结构中记录第一线程的标识符、第一线程的每条指令的周期和一个时间戳的部件。

17. 如权利要求 16 的信息处理系统, 其中所述时间戳相应于选自这样的一组的一个时间, 该组包括计算第一线程的每条指令的周期的时间, 和将第一线程的标识符从第一运行队列移动到第二运行队列的时间。

18. 如权利要求 13 的信息处理系统, 其中所述线程调度工具还包括:

将用于多个线程中的每个线程的每条指令的周期值与以前为每个线程计算的一个或多个以前的每条指令的周期值平均的部件, 其中被记录的每条指令的周期值包括每个线程的平均每条指令的周期值。

19. 如权利要求 10 的信息处理系统, 其中线程调度工具还包括:

跳过与第一线程相比的一个或多个更低性能线程、作为对确定每个所述更低性能线程以前已经被移动到第一同时多线程处理器的运行队列、并且它们每个自从被移动之后性能都提高了的响应的部件; 以及

在跳过所述更低性能线程之后识别第一线程的部件。

20. 如权利要求 11 的信息处理系统，其中所述线程调度工具还包括：

检测到第一线程将要结束的部件；

在第一线程结束时为其计算每条指令的周期值的部件；

确定该每条指令的周期值比一个阈值差的部件；

响应该确定、检查第一线程是否以前被从以前的同时多线程处理器移动到第一同时多线程处理器上、所述以前的同时多线程处理器选自包括第二同时多线程处理器的多个同时多线程处理器的部件；

响应对第一线程以前被移动过的确定、并确定第一线程在第一同时多线程处理器上的每条指令的周期比它在以前的同时多线程处理器上的每条指令的周期差的部件，其中所述用于写入和删除的软件代码是响应对第一线程在第一同时多线程处理器上的每条指令的周期比它在以前的同时多线程处理器上的每条指令的周期差的确定而执行的。

用于在SMT处理器中进行CPI负载平衡的系统和方法

技术领域

本发明一般地涉及用于在同时多线程(SMT)处理器上调度线程的系统和方法。更具体地，本发明涉及为了进行 SMT 调度，使用一种度量以便确定相互兼容的处理线程的系统和方法。

背景技术

现代计算机的基本结构包括有外围设备以便向外部世界传输信息和从外部世界接收信息；这种外围设备可以是键盘、显示器、磁带驱动器、耦合于网络的通信线路等。计算机的基础结构中还包括从外部世界接收、处理和向外部世界传递这些信息所需的硬件，包括总线、存储器单元、输入/输出（I/O）控制器、存储设备和至少一个中央处理单元（CPU）等。CPU是系统的大脑。它执行构成计算机程序的指令，并且指挥着其它系统组件的运行。

从计算机硬件的立场上看，多数系统的操作方式基本上相同。处理器实际上快速地执行非常简单的操作，诸如算术、逻辑比较、和数据从一个位置到另一个位置的移动。指挥着计算机执行大量这种简单操作的程序给人以这样的幻觉，即计算机正在执行一些很复杂的操作。然而，用户所感觉到的计算机系统新的或增强的能力实际上可能是机器正在执行同样的简单功能，但是执行得更快得多。因此，对计算机系统的继续改进要求这些系统被做得还要更快。

对计算机整体速度的一种度量，也称为吞吐量，是度量单位时间内所执行的操作的数目。从概念上讲，在所有可能的对系统速度的改进中最简

单的是提高各种部件的时钟速度，特别是处理器的时钟速度。如果所有的部件都以快两倍的速度执行，但是在其他方面以完全相同的方式工作，则系统将用原来时间的一半执行给定的任务。几年以前由分立部件构成的计算机处理器，通过缩小尺寸并且减少部件的数目，执行速度明显地更快了；最终，整个处理器被封装在单个芯片上的一个集成电路上。尺寸的缩小使得提高处理器的时钟速度成为可能，并且因此提高了系统的速度。

尽管由集成电路获得了巨大的速度提升，但是依然存在对更快的计算机系统的需求。硬件设计者通过更大的集成度、进一步减小电路的尺寸以及其它的技术可以获得速度的进一步提高。然而，设计者考虑到物理尺寸不可能无限地持续减小，而处理器时钟速度的不断提高也存在着限制。因此，已注意寻求其它方法进一步改进计算机系统的整体速度。

在不改变时钟速度的情况下，通过使用多个处理器仍然可能提高系统的速度。封装在集成电路芯片上的单个处理器的适度的成本使得这是可行的。从处理器的使用通过将工作负载从主处理器转移到从处理器上极大地提高了系统的速度。例如，从处理器例行地执行重复的和单个特定目的的程序，诸如输入/输出设备通信和控制。还可以将多个 CPU 放置在单个计算机系统内，典型地放置在基于主机的系统内，该系统同时为多个用户提供服务。每个不同的处理器可以为不同的用户单独地执行不同的任务，因此提高了系统同时执行多个任务的整体速度。

然而，提高单个任务诸如一个应用程序的执行速度更加困难。在多个 CPU 之间协调各个功能的执行并且传递各个功能的结果是一件很难处理的工作。对于从 I/O 处理器来说，这并不是如此困难，因为功能是已经预先定义的和有限的，而对于执行通用应用程序的多个 CPU 来说，协调这些功能就要困难得多，这部分地是因为，系统设计者事先不知道这些程序的细节。多数应用程序按着单一路径或由处理器所执行的步骤流执行。尽管有时可能将这个单一路径分解为多个并行的路径，进行这样处理的一种通用应用还在研究之中。一般地，将冗长的任务分解成较小的任务以便由多个处理器并行执行是由软件工程师逐个案例地编写代码来完成的。对于执

行不一定是重复的或可预知的商务事务来说，这种专用方法尤其成问题。

因此，虽然多处理器提高了整个系统的性能，但是仍然存在许多提高单个 CPU 的速度的理由。如果 CPU 时钟速度是给定的，则通过提高每个时钟周期内所执行的操作的平均数目，有可能进一步提高 CPU 的速度，即每秒执行的操作数目。一种用于高性能、单芯片微处理器的常见的体系结构是精简指令集计算机（RISC）结构，其特征在于一个经常使用的指令的小的、简化的集合以便快速执行，这些简单的操作如前面说明的被快速地执行。由于半导体技术的进步，RISC 体系结构的目标已经成为开发具有在每个机器时钟周期内执行一条或多条指令的能力的处理器。另一个增加每个时钟周期所执行的操作的平均数目的方法是修改 CPU 内的硬件。这种吞吐量度量，即每条指令的时钟周期，通常被用于表示高性能处理器体系结构的特征。指令流水线和高速缓冲存储器是使得这种成就成为可能的计算机体系结构特征。流水线指令执行允许随后的指令在先前发出的指令完成之前就开始执行。高速缓冲存储器在更靠近处理器处存储频繁使用的和其它数据，并且在多数情况下允许继续执行指令，而不必等待主存储器的整个访问时间。还已经展示了某些改进，其中多个执行单元具有前瞻硬件，用于寻找指令以并行执行。

对于在超标量系统中顺序和乱序地完成指令这两者，流水线在某些情况下将会停滞（stall）。依赖于先前发出的还没有完成的指令的结果的指令可能会使得流水线停滞。例如，在高速缓存中没有装入/存储指令中所需的数据，即高速缓存未命中时，依赖于该装入/存储指令的指令不能被完成，直到在高速缓存中可以得到该数据时为止。特别是对涉及到大数据结构的计算而言，在高速缓存中维护持续执行所必需的请求数据，并且维持高命中率，即对该数据的请求次数与在高速缓存中准备好了该数据的次数的比，并不是容易的事。一次高速缓存未命中可以使得流水线停滞几个时钟周期，并且如果数据在多数情况下都不在，则存储器延迟的总量将是严重的。虽然用于主存储器的存储器装置正在变得更快，但是这种存储器芯片的速度和高端处理器的速度之间的差距正在变得越来越大。因此，当前高端处理

器设计中的相当数量的执行时间花在了等待高速缓存未命中的解决上，并且这些存储器访问延迟占用了处理器执行时间的越来越多的部分。

提高 CPU 内硬件效率的另一种技术是将一个处理任务分成被称为线程的可独立地执行的指令序列。这种技术与将较大的任务分成较小的任务以便由不同的处理器独立地执行有关，只是此处将由同一个处理器执行这些线程。然后当 CPU 出于任何原因不能继续处理或执行这些线程中的一个线程时，CPU 切换到并且执行另一个线程。在计算机体系结构中定义的术语“多线程”与该术语在软件中的使用是不相同的，在软件中它的含义是将一个任务分割成多个相关的线程。在体系结构的定义中，这些线程可以是相互独立的。因此“硬件多线程”经常被用于区分该术语的这两种使用。

硬件多线程的传统形式涉及为每个线程复制处理器寄存器。例如，对于实现了 IBM 公司所提供的商业名称为 PowerPC™ 的体系结构以执行多线程的处理器，该处理器必须维护 N 个状态以便运行 N 个线程。因此，下面的内容要被复制 N 次：通用寄存器、浮点寄存器、条件寄存器、浮点状态和控制寄存器、计数寄存器、链接寄存器、异常寄存器、保存/恢复寄存器和专用寄存器。

此外，可以复制专用的缓冲区诸如段后备缓冲区（segment lookaside buffer），或者可以将线程号标记在每个缓冲区项上，并且否则，则必须被在每次线程转换时清仓每个缓冲区项。还有，某些分支预测机制，例如，相关性寄存器（correlation register）和返回堆栈也应该被复制。幸运的是，不需要复制处理器的某些较大的功能，诸如：一级指令高速缓存（L1 I－cache）、一级数据高速缓存（L1 D－cache）、指令缓冲区、存储队列、指令调度器、功能性或执行单元、流水线、翻译后援缓冲区（TLB）以及分支历史表。

同时多线程（SMT）是一种允许多个独立的线程在每个时钟周期内向超标量处理器的功能单元发出多条指令的技术。SMT 组合了现代超标量处理器的多指令特性和多线程体系结构的等待时间隐藏能力。与依靠快速上下文切换来共享处理器的执行资源的传统多线程体系结构不同，SMT 处理

器中所有硬件环境同时处于活动状态，它们在每个时钟周期竞争所有的可用资源。这种对功能性单元的动态共享允许同时多线程以便很大地提高吞吐量，解决了处理器利用率的两个主要障碍－－长的等待时间和有限的按线程的并行性。多个 SMT 处理器可以被包括在一个计算机系统内，从而允许计算机在多个计算机上执行同时多线程。

然而，具有多个SMT处理器的计算机面对的一个挑战是在SMT处理器上执行的诸软件线程竞争某些同样的基于处理器的资源，诸如功能性和执行单元。如在此处所使用的，术语“线程”指软件线程，除非另行说明。如果两个线程反复地竞争同样的基于处理器的资源，在一个线程使用该资源时，另一个线程将等待（或被交换出去（swap out））。等待资源降低了整个系统的吞吐量，并且降低了各个线程的效率。在多处理器环境中，由于运行在一个给定的 SMT 处理器上的其它线程的影响，一个线程可能不能在该处理器上很好地执行。然而，传统的系统不能将线程从一个SMT处理器移到另一个SMT处理器上。

因此，所需要的是一种系统和方法，所述的系统和方法识别不能在一个 SMT 处理器上很好地执行的线程，并且将该低性能线程移到另一个SMT处理器上。还需要的是一种在SMT处理器间交换低性能线程的系统和方法。最后，需要一种确定一个线程在被移动到另一个 SMT 处理器上之后性能是否提高了的系统和方法。

发明内容

已经发现，通过使用一种识别运行在一个 SMT 处理器上的低性能线程，并且将它移动到一个不同的 SMT 处理器上的系统和方法，可以解决上面提及的挑战。为运行在一个计算机系统上的软件线程记录性能数据。在一个实施例中，所述的性能数据包括每个线程的每条指令的周期(CPI)，它由用该处理器中出现的周期数除以所执行的指令数来确定。CPI值越小，线程执行得越好，并且相反地，CPI 值越高，线程执行得越糟。在一个实施例中，在多个执行上对其性能进行平均。

低性能的线程（即，具有高 CPI 值的线程）被从第一运行队列移动到第二运行队列，其中第一运行队列相应于该线程当前的 SMT 处理器，并且第二运行队列相应于一个不同的 SMT 处理器。在一个实施例中，在处理器之间交换低性能的线程。以这种方式，在一个处理器上不能很好地执行的线程可能能够在第二个处理器上令人满意地执行。性能可能被提高的一个原因在于运行在新分派的处理器上的其它线程可能不需要与被移动的线程相同的基于处理器的资源。由于对基于处理器的资源的较少的竞争，在该 SMT 处理器环境中这个线程的性能提高了。

当一个线程结束执行（即，被抢占、时间片到时等）时，该线程在最后执行过程中的 CPI 被记录并被检查。如果该线程比一个给定的阈值执行得差，则执行一个检查以查看该线程以前是否被移动过，并且如果它已经被移动过，它的性能是否因为被移动而提高了。如果该线程在移动之后性能变差了，或是该线程最近没有被移动过，则将该线程识别为潜在的将被移动的线程。在一个实施例中，来自每个 SMT 处理器的运行队列的一个线程被识别为低性能的，并且被移动到不同的运行队列中。当移动线程时，更新捕捉关于该线程的性能数据和时间戳的数据结构，以便随后确定该线程的性能是否提高了。对于具有多于两个 SMT 处理器的系统，可以各种方式将低性能线程从一个处理器移动到下一个处理器。例如可以使用轮转（round robin）实现方式来交换低性能线程，或是将处理器相互配对，并且将低性能的线程在配对的处理器间进行交换。

优选地，本发明提供一种为多个同时多线程处理器调度线程的计算机实施的方法，所述方法包括:

确定第一运行队列中的第一线程是一个低性能线程，其中第一运行队列相应于第一同时多线程处理器;

响应所述确定:

将与第一线程相应的第一标识符写入第二运行队列，其中第二运行队列相应于第二同时多线程处理器；以及

从第一运行队列中删除第一标识符。

另一方面，本发明还提供一种信息处理系统，包括:

多个同时多线程处理器;

处理器可以访问的存储器;

存储在存储器中的多个软件线程和多个运行队列，其中每个运行队列相应于同时多线程处理器中的一个;

线程调度工具，它用于在多个同时多线程处理器间调度线程，所述线程调度工具包括:

确定来自多个软件线程的第一线程是一个低性能线程、所述多个软件线程处于选自多个运行队列中的第一运行队列、其中第一运行队列相应于选自多个同时多线程处理器中的第一同时多线程处理器的部件;

响应所述确定、将相应于第一线程的第一标识符写入第二运行队列、其中第二运行队列相应于选自多个同时多线程处理器的第二同时多线程处理器、以及从第一运行队列中删除第一标识符的部件。

以上是本发明的概述，并且因此必然包括简化、概括和对细节的忽略;因此，本领域的技术人员可以理解这个概述仅是示例性的，而并不旨在以任何方式进行限制。通过下面给出的非限制性的详细说明，将明了在权利要求中完整地定义的本发明的其它方面、发明特征和优点。

附图说明

通过参考附图，本领域的技术人员可以更好地理解本发明，并且可以明了本发明的各个目的、特征和优点。不同附图中所使用的相同的标号表

示类似的或相同的项。

图 1 是将被调度以便在 SMT 处理器上并发执行的多个线程的高层示意图。

图 2 是表示了调度器使用线程度量数据和运行队列数据在 SMT 处理器上调度线程的示意图;

图 3 是表示了调度器在确定将调度到 SMT 处理器上的一个线程的过程中所采取的步骤的流程图;

图 4 是表示了更新线程兼容性列表所采取的步骤的流程图;

图 5 是表示了从线程兼容性列表内去除表项所采取的步骤的流程图;

图 6 是表示了周期性地清除在线程控制块中发现的兼容性列表所采取的步骤的流程图;

图 7 是表示了调度器将一个低性能执行的线程从一个 SMT 处理器交换到另一个 SMT 处理器上以便提高整个系统的性能的示意图;

图 8 是表示了更新线程的 CPI 所采取的步骤的流程图;

图 9 是表示了在 SMT 处理器之间交换低性能线程所采取的步骤的流程图;

图 10 是表示了识别低性能线程以便在多 SMT 处理器系统内进行交换所采取的步骤的流程图;

图 11 是表示了在 SMT 处理器间交换低性能线程所采取的步骤的流程图; 以及

图 12 是能够实现本发明的一种信息处理系统的方块图。

具体实施方式

下面的说明旨在提供对本发明的一个例子的详细说明，并且不应被理解为是对本发明本身的限制。确切地说，可以有任何数目的改型落在本发明的范围内，本发明定义于本说明之后的权利要求内。

图 1 是被调度以便在 SMT 处理器上并发执行的多个线程的高层示意图。调度器 100 读取与多个线程 110 相应的线程数据。在一个实施例中，

线程数据被存储在线程控制块（TCB）内，线程控制块被系统用于维护并管理当前存在的线程。

调度器 100 调度线程以在支持同时多线程的处理器 120 上执行。同时多线程（SMT）是一种允许多个独立的线程在每个周期内向超标量处理器的功能单元发出多条指令的技术。SMT 组合了现代超标量处理器的多指令特征和多线程体系结构的等待时间隐藏能力。与传统的依赖于快速上下文切换来共享处理器执行资源的多线程体系结构不同，SMT 处理器内的所有硬件环境同时处于活动状态，每个周期都竞争所有的可用资源。这种对功能性单元的动态共享允许同时多线程以便很大地提高吞吐量，解决了处理器利用率的两个主要障碍－－长的等待时间和有限的按线程的并行性。

处理器线程 130 和 140 代表在处理器 120 上同时执行的两个线程，它们竞争处理器资源 150、160 和 170。取决于正在执行的任务，每个可执行的线程 110 都有对处理器资源的不同需求。例如，某些线程可能是数据密集的，需要大量地访问处理器的“装入/存储”资源，而其它线程可能是计算密集的，并且需要大量地访问处理器的“加法器”资源或是浮点资源。

因为资源是在当前执行的线程之间共享的，如果每个线程需要执行同样的功能，则其中一个线程必须在另一个线程获得对资源的访问时等待。如果进程间存在对资源的高度竞争，则与低竞争的情况相比这些进程将花费更多的时间完成。

为每个可执行的线程维护指示着每个线程与其它线程的兼容性的数据。例如，如果两个线程可以很好地一起运行（即，当一起运行时，它们每个都有低的 CPI），这个信息将被调度器用于优选地使得这两个线程同时运行。

图 2 是表示了调度器使用线程度量数据和运行队列数据调度线程到 SMT 处理器上的示意图。当一个线程在 SMT 处理器 250 上执行完毕时，调度器 200 调度下一个线程以便执行。如果来自运行队列 A（270）的一个线程结束了，则另一个来自运行队列 A 的线程被调度。同样地，如果来自运行队列 B（285）的一个线程结束了，则调度器从运行队列 B 中调度另一

个准备好要执行的线程。

为了确定将调度哪个线程，调度器 200 确定当前哪个线程正在另一个处理器线程上运行。例如，如果要结束的线程来自运行队列 B，则获取与当前在处理器线程 A（255）中执行的线程相应的线程标识符。类似地，如果要结束的线程来自运行队列 A，则获取与当前正在处理器线程 B（260）中执行的线程相应的线程标识符。运行队列包括与线程的标识符有关的数据（用于运行队列 A 中的线程的标识符 275，和用于运行队列 B 中的线程的标识符 290），以及指示着该线程是否准备好执行的数据（用于运行队列 A 中的线程的指示符 280，和用于运行队列 B 中的线程的标识符 295）。

然后调度器确定是否有任何“兼容的”线程准备好了执行。例如，如果来自运行队列 A 的线程“aaa”正在与来自运行队列 B 的线程“bbb”一起执行，而线程“bbb”将要结束了，则调度器确定是否有与线程“aaa”兼容的任何线程准备好了执行。在一个实施例中，调度器从线程控制块 210 中线程“aaa”的表项内读取线程“aaa”的兼容性信息。线程控制块 210 包括用于每个线程的多个数据字段。这些数据包括线程 215 的标识符、用于多达 3 个相应于兼容线程的标识符（标识符 220、235、245）的字段、以及当这些兼容线程与这些兼容线程同时运行时所达到的每条指令的周期数（CPI）（CPI 225 相应于线程 ID 220，CPI 235 相应于线程 ID 230，CPI 245 相应于线程 ID 240）。在一个实施例中，存储在字段 225、235 和 245 中的 CPI 是该线程最后与兼容线程同时执行的过程中所得到的最后的 CPI。在另一实施例中，为各个线程保持平均 CPI，并且将在该线程与兼容线程同时运行过程中所获得的平均 CPI 存储在 CPI 字段中。

在给出的例子中，线程“aaa”是最与线程“bbb”兼容的线程，因为它在与线程“bbb”同时运行时具有最低的 CPI（0.7）。线程“aaa”还与线程“ddd”（CPI 为 1.0）以及“fff”（CPI 为 1.2）兼容。调度器确定线程“ddd”是否准备好执行，并且如果是这样的，将该线程从运行队列 B 调度到 SMT 处理器 250 中处理器的“B”线程空间（260）内。如果线程“ddd”没有准备好执行，则调度器确定线程“fff”是否准备好执行，并

且如果是的则调度器调度该线程。如果线程“ddd”和“fff”都没有准备好执行，则选择运行队列 B 中的下一个准备好执行的线程，并且由调度器调度它。

此外，当线程结束运行时，更新其 CPI。某些兼容线程 ID 是“[null]”（一个空槽（slot）），这指示着没有发现足够的兼容线程来填满所有的槽。当一个线程结束执行时，就捕获该线程的 CPI，并且将其与 CPI 阈值 265 比较。如果该线程的 CPI 好于该阈值，则可将 CPI 的度量和标识符放置到一个兼容性槽内，如果（1）存在一个空的槽的话，或者（2）不存在空的槽，但是新捕获的 CPI 比当前兼容线程中的一个线程的 CPI 更好的话（在这种情况下，新捕获的 CPI 和线程标识符替换掉具有最高即最不好的 CPI 的当前兼容线程）。

图 3 是给出了调度器在确定哪个线程将被调度到 SMT 处理器上时所采取的步骤的流程图。处理从 300 开始，在步骤 305 调度器收到一个通知，通知当前正在运行的线程将要任务调出（task out）。

确定哪个运行队列包括该将要结束的线程（判断 310）。如果将要结束的线程在运行队列“A”上，则判断 310 转向“是”分支 318，以便获得从运行队列“A”调度的下一个线程。在步骤 320，检查兼容性列表以确定运行队列“A”上的哪些线程与当前正在处理器线程“B”（它接收来自运行队列“B”的线程）上运行的线程兼容。这个检查是通过读取存储在线程控制块 315 中的兼容数据（关于存储在线程控制块中的数据的更详细的例子，参见图 2 中的块 210）而进行的。

确定是否存在有被列出的与运行在处理器线程“B”上的线程兼容的任何线程（判断 325）。如果没有兼容的线程被列出，则判断 325 转向“否”分支 326，于是在步骤 350，来自运行队列“A”的下一个可得到的准备好运行的线程被调度，并且处理在 395 处结束。

在另一方面，如果线程控制块中所列出的一个或多个线程与当前在处理器线程“B”上运行的线程兼容，则判断 325 转向“是”分支 328，于是在步骤 330，通过读取来自运行队列“A”的数据检查最兼容的线程（即具

有最小 CPI 的线程），以便确定它是否准备好了运行。确定最后检查的线程是否准备好了运行（判断 335）。如果最后检查的线程准备好了运行，则判断 335 转向“是”分支 338，于是在步骤 340 调度该线程。在另一方面，如果该兼容的线程没有准备好执行，判断 335 转向“否”分支 342，于是确定是否还有兼容线程列于线程控制块中（判断 345）。如果还有兼容线程列出，则判断 345 转向“是”分支 346，它循环回去以便检查该线程是否准备好运行。这个循环一直继续直到或是发现了一个兼容的准备好了运行的线程（判断 335 转向“是”分支 338），或是没有更多的兼容线程要检查，判断 345 转向“否”分支 348，于是在步骤 350，下一个来自运行队列“A”的可用线程（准备好运行的）被调度。然后处理在 395 处结束。

回到判断 310，如果将结束的线程是运行队列“B”上的线程，判断 310 转向分支 352，于是执行相同意义的一组判断和步骤，以便确定应该调度来自队列“B”的哪个线程。这些步骤的细节如下：

在步骤 355，检查兼容性列表以确定运行队列“B”上的哪些线程与当前在处理器线程“A”（它接收来自运行队列“A”的线程）上运行的线程兼容。这个检查是通过读取存储在线程控制块 315 中的兼容数据而进行的。

确定是否有任何列出的与运行在处理器线程“A”上的线程兼容的线程（判断 360）。如果没有列出的兼容线程，则判断 360 转向“否”分支 362，于是在步骤 390，来自运行队列“B”的准备好运行的下一个可用线程被调度，并且处理在 395 处结束。

在另一方面，如果在线程控制块中列出了一个或多个与当前运行在处理器线程“A”上的线程兼容的线程，则判断 360 转向“是”分支 364，于是在步骤 365 通过读取来自运行队列“B”的数据检查最兼容的线程（即具有最小 CPI 的线程）以便确定它是否准备好了运行。确定最后检查的线程是否准备好了运行（判断 375）。如果最后检查的线程准备好了运行，判断 375 转向“是”分支 378，于是在步骤 380 调度该线程。在另一方面，如果该兼容的线程没有准备好运行，则判断 375 转向“否”分支 382，于

是确定是否还有兼容线程列于线程控制块内（判断 385）。如果还有线程列出，判断 385 转向“是”分支 386，它循环回去以查看该线程是否准备好了运行。这个循环一直继续直到或是发现了一个兼容的准备好了运行的线程（判断 375 转向“是”分支 378），或是没有更多的兼容线程要检查，判断 385 转向“否”分支 388，于是在步骤 390，下一个来自运行队列“B”的可用线程（准备好运行的）被调度。然后处理在 395 处结束。

图 4 是表示了更新线程的兼容性列表所采取的步骤的流程图。处理从 400 开始，在步骤 405，获取刚刚在一个处理器线程上结束执行的线程的标识符，以及仍然在另一个处理器线程上执行的线程的线程标识符。接着，在步骤 410，获取在刚刚结束的该线程和仍然在执行的另一个线程的时间段内所获得的 CPI。

确定获得的 CPI 是否比已经建立的兼容阈值小或相等（即更好）（判断 415）。如果 CPI 大于该阈值，则这些线程被认为是不兼容的。所述的阈值是一个可以调整的值。该值越高，就有越多的线程将被认为是兼容的，然而由于该值较高，这些 CPI 将不会必然地极大地提高整体系统的性能。在另一方面，降低阈值更可能确保“兼容的”线程，当可以得到时，将在一起很好地执行，然而由于阈值较低，可能会识别出较少的兼容线程。因此，取决于给定的计算机系统正在执行的处理的类型，可能需要调整该兼容阈值。

如果 CPI 大于阈值，判断 415 转向“否”分支 418，于是这些线程被判定为是“不兼容的”，并且删除任何指示着这些线程为兼容线程的表项（预先定义的过程 420，处理细节参见图 5），并且处理在 425 处结束。

在另一方面，如果 CPI 小于或等于该阈值，判断 415 转向“是”分支 428，于是在步骤 430 检查刚刚结束的线程的兼容性列表。如果当前正在运行的线程已经存在于兼容性列表内了，则在步骤 430 中更新用于该线程的 CPI。在一个实施例中，所述的线程表追踪最后的 CPI，在这种情况下，在线程表中相应于当前正在执行的线程的标识符的字段中插入最新的 CPI。在另一个实施例中，线程表保留一个平均 CPI 值，在这种情况下将

最新的CPI值与当刚刚结束的线程和当前正在执行的线程一起运行时所获得的其它值进行平均。此外，在步骤430中，记录一个时间戳，以便追踪这两个线程一起运行的最后时间。

在当前正在运行的线程没有被列于最后线程的兼容性列表内的情况下，做出一个关于兼容性列表内是否有任何空槽（即字段）的决定（判断435）。如果至少有一个空（即当前没有使用的）字段，判断435转向“是”分支438，于是在步骤440记录当前正在运行的线程的线程标识符以及其CPI和一个时间戳。

在另一方面，如果在用于刚刚结束的线程的兼容性列表内没有空槽，判断435转向“否”分支442，它绕过步骤440并且进行另一个关于在这两个线程之间所获得的CPI是否好于（即小于）当前列于兼容性列表内的最不好的（即最高的）CPI的确定（判断445）。如果为这两个线程所获得的CPI好于当前列于兼容性列表内的CPI中的一个CPI，判断445转向“是”分支448，于是在步骤450，与列出的最高兼容CPI相应的线程标识符被改写为当前正在运行的线程的线程标识符，兼容性列表内的CPI值被改写为刚刚获得的CPI，并且以前的时间戳被改写一个更新过的时间戳，它反映了由这两个线程获得该CPI的时刻。

如果该CPI不比线程表中所列的最差CPI好，不触动（即，不改变）与刚刚结束的线程的线程标识符相应的兼容性列表表项，并且判断445转向“否”分支452。

与上面说明的为刚刚结束执行的线程更新兼容性列表的步骤430到450相类似，执行相同的步骤以便为当前正在运行的线程更新兼容性列表。在步骤460，检查与当前正在运行的线程的标识符相应的兼容性列表，并且如果刚刚结束的线程的线程标识符已被列于其内，则与刚刚结束的线程相应的数据被更新（即，CPI和时间戳被更新）。再次的，在一个实施例中，追踪最后的CPI，而在另一个实施例中计算并记录平均CPI。

在刚刚结束的线程没有列于当前正在运行的线程的兼容性列表内的情况下，确定兼容性列表内是否还有任何空槽（判断470）。如果至少还有

一个空（即，当前没有使用的）字段，判断 470 转向“是”分支 472，于是在步骤 474，记录当前正在运行的线程的标识符以及 CPI 值和时间戳。

在另一方面，如果在当前正在运行的线程的兼容性列表内没有空位置，判断 470 转向“否”分支 478，它绕过了步骤 475，并且进行另一个关于由这两个线程所获得的 CPI 是否比兼容性列表内当前所列出的最差（即，最大的）CPI 更好（即，更小）的确定（判断 480）。如果由这两个线程所获得的 CPI 比兼容性列表内当前列出的 CPI 中的一个 CPI 好，判断 480 转向“是”分支 485，于是在步骤 490 与所列出的最高兼容 CPI 值相应的线程标识符被改写为刚刚结束执行的线程的线程标识符，兼容性列表内的 CPI 值被改写为刚刚得到的 CPI，并且以前的时间戳被改写为更新的时间戳，它反映了由这两个线程获得该 CPI 的时刻。

如果该 CPI 不比线程表内所列出的最差 CPI 好，则不触动（即，不改变）相应于刚刚结束的线程的线程标识符的兼容性列表表项，而判断 480 转向“否”分支 492，绕过步骤 490。

此后为更新线程的兼容性列表所执行的处理在 495 处结束。

图 5 是表示了从线程的兼容性列表内删除表项所采取的步骤的流程图。当在 SMT 处理器上同时执行两个线程时所获得的 CPI 不如（即，高于）为系统设定的一个阈值时调用该过程（见图 4，已预先定义的处理 420，它调用图 5 中表示的处理）。

图 5 的处理从 500 开始，在步骤 510 读取相应于刚刚结束执行的线程的兼容性列表，以便确定当前正在执行的线程的线程标识符是否被作为兼容线程列出来了。在一个实施例中，兼容性列表存储于线程控制块 540 内。确定当前正在执行的线程的标识符是否被列于最后的线程兼容性列表内（判断 520）。如果当前线程被列于最后的线程的兼容性列表内，判断 520 转向“是”分支 525，于是在步骤 530 从兼容性列表内删除与当前正在执行的线程有关的数据。在一个实施例中，兼容性列表数据存储于线程控制块 540 中。在另一方面，如果与当前正在执行的线程有关的数据没有列于刚刚结束的线程的兼容性列表内，判断 520 转向“否”分支 535，绕过步

骤 530。

在步骤 550，读取相应于当前正在执行的线程的兼容性列表以便确定刚刚结束执行的线程的线程标识符是否作为兼容线程被列出了。确定刚刚结束执行的线程的线程标识符是否列于当前正在运行的线程的兼容性列表内（判断 560）。如果刚刚结束执行的线程的线程标识符被列于当前正在运行的线程的兼容性列表内，判断 560 转向“是”分支 570，于是在步骤 580 从该兼容性列表内删除与刚刚结束执行的线程有关的数据。在另一方面，如果与刚刚结束执行的线程有关的数据没有列于当前正在运行的线程的兼容性列表内，判断 560 转向“否”分支 590，绕过步骤 580。处理此后在 595 处结束。

图 6 是周期地清理在线程控制块中发现的兼容性列表所采取的步骤的流程图。处理从 600 开始，于是在步骤 605，处理以周期性的间隔，例如每两秒钟，被唤醒。

处理一直继续直到系统关闭为止。因此，确定系统是否正在被关闭。当系统正在被关闭时，判断 610 转向“是”分支 612，于是处理在 615 处结束。

在另一方面，如果系统不是正在被关闭，判断 610 转向“否”分支 618 以执行线程清理操作。在步骤 620，处理获得当前的系统时间（时间戳）。通过从该时间戳中减去一过时的时间，基于当前时间计算一个过时时间戳（步骤 625）。例如，一个在过去两秒钟内没有更新其时间戳值的本来兼容的线程可以被认为是“过时的”，并且因此不再被认为与该线程兼容了。其原因可能是由于另一线程已经终止了，该另一线程已被置于睡眠状态而等待其它事件的发生，或是没有调度该另一线程以便和该本来兼容的线程一起运行的某种其它原因。

在步骤 630，读取线程控制块中的第一个线程。确定该线程控制块数据是否包括兼容线程的信息（判断 635）。如果用于该线程的表项包括兼容线程信息，判断 635 转向“是”分支 638，于是在步骤 640 读取相应于第一个被列出的兼容线程的时间戳。通过比较这个时间戳和计算出的过时

时间戳值，确定兼容性列表内所列的该线程是否已过时并应当从列表中删除（判断 650）。如果兼容性列表内列出的线程过时了，判断 650 转向“是”分支 655，于是在步骤 660，过时的线程被从兼容线程列表内删除。在另一方面，如果兼容线程的时间戳位于一个可接受的参数内（即，线程没有过时），则判断 650 转向“否”分支 665，并且该线程保留在兼容线程列表内。

确定是否有更多的列于该兼容线程列表内的线程需要被处理（判断 670）。如果有更多的线程列出，判断 670 转向“是”分支 672，于是在步骤 675，读取兼容线程列表内的下一个线程的时间戳，并且处理循环回去以确定该线程是否已过时并应被从兼容线程列表内删除。这个循环一直继续直到对于从线程控制块中读取的该线程没有更多的兼容线程列出为止，此时判断 670 转向“否”分支 678。

确定是否有更多的列于线程控制块内的线程需要被处理，并且清理它们的兼容性列表（判断 680）。如果线程控制块内有更多的线程，判断 680 转向到“是”分支 685，于是在步骤 690，读取线程控制块内下一个线程的数据，并且处理循环回去以清理该线程兼容性列表内的任何过时线程。

这个循环一直继续直到线程控制块内的所有线程都已被读取为止，此时判断 680 转向“否”分支 695，它循环回步骤 605，使得处理在再次执行清理处理之前等待所述的时间间隔流逝。清理处理继续，直到系统被关闭为止，此时判断 610 转向“是”分支 612，并且处理在 615 处结束。

图 7 是一个表示了调度器将低性能线程从一个 SMT 处理器交换到另一个 SMT 处理器上以便提高整体系统性能的示意图。调度器 700 读取与存储于线程控制块 710 内的线程有关的数据，以便确定将在具有多个 SMT 处理器的系统内的各个处理器之间交换哪些线程。

存储在线程控制块内的 CPI 数据被调度器用于从该线程以前的执行过程中所收集的数据中识别低性能的线程。在图 7 给出的实施例中，给出了两个 SMT 处理器（760 和 785），它们每个都具有两个处理器线程以便同时执行两个执行线程（处理器线程 765 和 770 相应于 SMT 处理器 760，并

且处理器线程 790 和 795 相应于 SMT 处理器 785）。每个 SMT 处理器都具有一个运行队列（运行队列 755 相应于 SMT 处理器 760，并且运行队列 780 相应于 SMT 处理器 785）。运行队列识别被调度以便在处理器上运行的线程。在给出的例子中，线程“aaa”、“ccc”、“eee”和“ggg”被列于运行队列 755，并且因此在 SMT 处理器 760 上执行。类似地，线程“bbb”、“ddd”、“fff”和“hhh”被列于运行队列 780，并且因此在 SMT 处理器 785 上执行。

调度器 700 确定来自各个运行队列的哪些线程是最低性能线程。一旦识别出了低性能线程，调度器 700 内的过程 705 就将这些线程从一个运行队列交换到另一个运行队列。在给出的例子中，线程“ggg”是列于运行队列 755 内的最低性能线程，而线程“hhh”是列于运行队列 780 内的最低性能线程。当执行过程 705 时，线程“ggg”将被放置到运行队列 780 中，而线程“hhh”将被放置到运行队列 755 中。

因为线程在 SMT 环境中共享处理器资源，将线程从一个运行队列交换到另一个运行队列是将被交换的线程放置到一个具有不同处理器资源的不同线程的池中。因此，交换线程的目的是为低性能的线程寻找一个更有效的环境以便减少对处理器资源的竞争，从而提高线程的效率。此外，图 7－11 中给出的交换线程技术可以与图 1－6 中给出的 SMT 调度技术结合使用，从而一个运行队列中的线程可以与该相同运行队列内的更兼容的线程一起被调度。

图 8 是表示了更新线程的 CPI 所采取的步骤的流程图。处理从 800 开始，于是在步骤 810，由调度器将一个线程调度到 SMT 处理器内包括的处理器线程中的一个处理器线程上。在步骤 820，从处理器获得一个初始的 CPI 值。在一个实施例中，处理器记录下所执行的周期数以及执行的指令数。然后用周期数除以所执行的指令数来计算 CPI。在步骤 830，线程执行一段时间直到线程结束其处理或任务调出（即，时间到）。当线程将要结束执行时，在步骤 840 收到一个通知，通知该过程该线程将要结束处理。如果图 8 中表示的处理是由调度器执行的，则因为调度器确定何时调度线

程并且何时它们任务调出，该过程将决定该线程将要结束。在另一方面，如果图 8 中表示的处理由与调度器分离的一个过程执行，则调度器在该线程将要结束执行时向该过程发送一个信号。

在步骤 850，获得刚刚结束执行的线程的最终的 CPI。通过计算该线程执行过程中所经过的周期数和由处理器在该线程执行过程中所执行的指令数目，为线程的最后的运行周期确定 CPI 值。在步骤 860，在线程表 870（即，线程控制块）中存储线程的最后 CPI。在步骤 880，通过对存储于线程表内的这个线程的 CPI 值进行平均计算该线程的平均 CPI。然后在步骤 890，在线程表 870 内存储该线程的平均 CPI。

为由调度器调度的每个线程执行图 8 中表示的处理。出于示例的目的，图 8 给出了为单个线程执行的 CPI 追踪。由于 SMT 处理器同时运行多个线程，图 8 中表示的处理将被调用多次以便追踪各个处理器线程。例如，如果 SMT 处理器同时处理两个处理器线程，则图 8 中表示的处理将或是被执行两次（每一个线程一次），或是被修改为追踪两个线程的 CPI。

图 9 是表示了在 SMT 处理器间交换低性能线程所采取的步骤的流程图。处理从 900 开始，在步骤 905 初始化（即，清理）交换列表 970。在 910 选择相应于第一处理器的运行队列 915。在步骤 920，通过搜索整个线程表 925，寻找所选择的运行队列内的具有最差（即最高的）CPI 的线程，来识别第一运行队列 925 内的最差性能的线程。

确定该最差性能线程的 CPI 是否高于（即，差于）一个预先确定的阈值（判断 930）。做出这个确定以便确保仅有低性能的线程被交换，否则将会占用额外的资源来交换具有足够性能的线程。如果线程的 CPI 不比该阈值差，判断 930 转向“否”分支 932，它绕过了为了在 SMT 运行队列间进行交换而将线程数据写入交换列表所采取的步骤。在另一方面，如果线程的 CPI 比该阈值差，判断 930 转向“是”分支 934，于是在步骤 935，读取以前交换的线程列表 940 以便查看该最低性能线程是否最近被交换过。以前交换的线程列表包括有关于已经被交换的线程的数据。这个数据包括被交换的线程的标识符、这些线程被交换时的 CPI、以及指示着每个

线程被最后交换的时间的时间戳。

确定最低性能的线程以前是否被交换过，这是由在以前交换的列表内发现的该线程的标识符所指示的（判断 945）。如果该线程以前被交换过，则判断 945 转向“是”分支 948，于是确定该线程的 CPI 是否在交换后变差了或是在交换前更差（判断 950）。如果被交换后该线程的 CPI 改善了（即，不是更差了），则判断 950 转向“否”分支 952，于是在步骤 955，识别出来自所选的处理器运行队列的下一个最差性能线程，并且处理循环回去以确定这个线程以前是否被交换过，以及交换后该线程的性能是提高还是下降了。回到判断 950，如果所选的线程的 CPI 在交换后更差了，则判断 950 转向“是”分支 956。

确定是否将识别出的线程包括到交换列表内（判断 965）。这个判断可以基于各种因素，诸如该线程的 CPI 是否比给定的阈值差，以及如果该线程以前被交换过，交换发生在多久之前。可以决定不交换近期内刚刚被交换过的线程，以避免在处理器运行队列间将同一个线程换来换去。如果该线程将要被包括在交换列表中，判断 960 转向“是”分支 962，于是在步骤 965，通过将该线程的标识符写入交换列表 970 来更新该列表。在另一方面，如果线程不是将要包括在交换列表内，判断 960 转向“否”分支 968，绕过步骤 965。

确定是否还有其它的 SMT 处理器，可以从它们之上识别低性能线程（判断 975）。为了在处理器间交换线程，计算机系统内至少要有两个 SMT 处理器，因此判断 975 将至少有一次转向“是”分支 978。

如果有更多的 SMT 处理器要处理，判断 975 转向“是”分支 978，于是在步骤 980，选择该多处理器系统内的下一个处理器，并且处理循环回去以从所选的 SMT 处理器识别一低性能线程。这个循环一直继续直到所有的 SMT 处理器都被处理过了，此时判断 975 转向“否”分支 982，于是在各个处理器之间交换交换列表内的线程（预先定义的处理 985，参见图 11 和相应文字来了解处理细节）。在步骤 990，更新以前交换的线程的列表 940，以便记录该些被交换的线程的线程标识符、最后的 CPI 和时间戳。

此外，如果系统内包括多于两个 SMT 处理器，则以前交换的线程列表也追踪该线程所来自的处理器。此后处理在 995 处结束。

图 10 是表示了识别低性能线程以便在多 SMT 处理器系统内进行交换所采取的步骤的流程图。处理从 1000 开始，于是在步骤 1005，运行在 SMT 处理器中的一个处理器上的线程中的一个线程结束了（即，被抢占了，时间片到时等）。在步骤 1010 从线程表 1015 内读取该最近结束的线程的 CPI，线程表 1015 包括有当前正在执行的线程的与 CPI 有关的信息。

确定最近结束的线程的 CPI 是否比一个预先确定的阈值差（判断 1020）。该预先确定的阈值是一个可以调整的值。将该阈值设高将减少识别出的低性能线程的数目，而相反地，将该值设低将增加识别出的低性能线程的数目。如果该最近结束的线程的 CPI 不比该阈值差，判断 1020 转向“否”分支 1022，于是处理在 1025 结束。在另一方面，如果该最近结束的线程的 CPI 比该阈值差，判断 1020 转向“是”分支 1028 以执行进一步的处理，以便最终确定该最近结束的线程是否应被交换到其它处理器。

在步骤 1030，读取以前交换过的线程的数据结构 1040。这个数据结构包括关于以前从一个 SMT 处理器被交换到另一个 SMT 处理器上的线程的信息，并且包括诸如线程的标识符、线程在其最后被交换时的 CPI、以及指示着该线程最后一次被交换的时间的时间戳。

确定该最近结束的线程是否是最近被交换的（判断 1050）。如果该最近结束的线程是以前交换过的，判断 1050 转向“是”分支 1055，于是确定该最近结束的线程的 CPI 在交换后是否变差了（判断 1060）。如果在交换之后该最近结束的线程的 CPI 没有变差(即相同或有所改善)，判断 1060 转向“否”分支 1065，于是在步骤 1070，在所选择的处理器上识别出下一个低性能线程，并且处理循环回去以确定新识别的线程是否比所述的阈值差，该线程是否以前被交换过，以及该新识别出的线程的 CPI 在被交换后是否变差了。这个循环一直继续或是直到识别出的线程的 CPI（基于线程的 CPI）比给定的阈值好（此时处理在 1025 处结束），或是直到识别出其 CPI 比该阈值差的线程，并且该线程或是以前尚未被交换过了（判断 1050

转向“否”分支 1075），或是在交换后其 CPI 变差了（判断 1060 转向“是”分支 1078）。

当已识别出了低性能的线程时，确定是否将识别出的线程包括到交换列表内（判断 1080）。这个判断可以基于各种其他因素，诸如多久以前该线程被交换，该线程在一个不同的处理器上执行的情况好了多少（即，可将在不同处理器上的边际改进（marginal improvement）与交换线程的操作相权衡）等等。如果该确定是仍然将该线程包括在交换列表内，判断 1080 转向“是”分支 1082，于是在步骤 1085，通过将线程的标识符写入交换列表 1090 内更新交换列表。在另一方面，如果该确定是不将该线程包括在交换列表内，判断 1080 转向“否”分支 1092，绕过步骤 1085。此后处理在 1095 处结束。

图 11 是表示了在 SMT 处理器间交换低性能线程所采取的步骤的流程图。处理从 1100 开始，于是在步骤 1110 从交换列表获取两个线程的标识符。在一个实施例中，交换列表用于交换来自多于两个 SMT 处理器的线程。因此，交换列表既识别线程也识别所述线程最后执行于其上的 SMT 处理器。此外，基于所述线程在不同处理器上的过去的性能，交换列表可以识别该线程应被调度在其上的处理器。例如，在具有 4 个 SMT 处理器的系统内，其中一低性能线程已经在第一、第二和第三个 SMT 处理器上试运行过了，则交换列表可以指出该线程应被调度到第四个 SMT 处理器上运行。

在步骤 1125，从交换列表中读取的第一个线程被放置在一个不同的运行队列上（即，相应于一个不同的 SMT 处理器的运行队列）。在步骤 1130，从交换列表读取的第二个线程也被放置在一个不同的运行队列上。在一个实施例中，第一个线程被放置在与第二个线程相应的运行队列上，并且第二个线程被放置在与第一个线程相应的运行队列上。在步骤 1140，更新线程表 1150，反映出对线程的运行队列所做的修改。在步骤 1160，与这些线程有关的数据被写入以前交换过的线程的数据结构 1170。这个数据包括线程的标识符、线程在被交换时的 CPI、以及指示线程被交换的时刻的当前

时间戳。

在步骤 1175，从交换列表 1120 内删除与刚刚被交换到不同运行队列的线程相应的信息。确定交换列表内是否还有更多表项需要被交换（判断 1180）。如果交换列表内还有其它表项，判断 1180 转向“是”分支 1190，它循环回去以便交换交换列表内的下两个表项（并且从该列表内删除这些表项）。这个循环一直继续直到交换列表为空时为止，此时判断 1180 转向“否”分支 1192，并且处理在 1195 处结束。

图 12 表示了信息处理系统 1201，该系统是能够执行此处说明的计算操作的计算机系统的一个简化的例子。计算机系统 1201 包括处理器 1200，它连接到主机总线 1202。二级（L2）高速缓冲存储器 1204 也连接到主机总线 1202。主机到 PCI 的桥 1206 连接于主存储器 1208，该桥包括高速缓冲存储器和主存储器控制功能，并且提供总线控制以便处理 PCI 总线 1210、处理器 1200、L2 高速缓存 1204、主存储器 1208 与主机总线 1202 之间的传输。主存储器 1208 连接于主机到 PIC 桥 1206 以及主机总线 1202。完全由（诸）主机处理器 1200 使用的设备，诸如 LAN 卡 1230，连接于 PCI 总线 1210。服务处理器接口和 ISA 访问直通（ISA Access Pass-through）1212 在 PCI 总线 1210 和 PCI 总线 1214 之间提供了一个接口。以这种方式，PCI 总线 1214 被从 PCI 总线 1210 分隔开。诸如闪速存储器 1218 的设备连接于 PCI 总线 1214。在一种实现中，闪速存储器 1218 包括 BIOS 代码，它包括有用于各种低级系统功能和系统启动功能的必要的处理器可执行的代码。

PCI 总线 1214 为由（诸）处理器 1200 和服务处理器 1216 共享的各种设备提供了一个接口，所述的设备包括，例如，闪速存储器 1218。PCI 到 ISA 桥 1235 提供了总线控制以便处理 PCI 总线 1214 和 ISA 总线 1240 间传输、通用串行总线（USB）功能 1245、电源管理功能 1255，并且能够包括其它未给出的功能元件，诸如实时时钟（RTC）、DMA 控制、中断支持和系统管理总线支持。非易失 RAM 1220 连接于 ISA 总线 1240。PCI 到 SCSI 桥 1280 提供了总线控制以便处理 PCI 总线 1214 和 SCSI 总线 1285

间的传输。SCSI 设备 1290（即，SCSI 硬盘驱动器）使用 SCSI 总线 1285 与计算机系统 1201 的其它部分通信。

服务处理器 1216 包括用于在初始化步骤中与（诸）处理器 1200 通信的 JTAG 和 I2C 总线 1222。JTAG/I2C 总线 1222 也连接于 L2 高速缓存 1204、主机到 PCI 桥 1206 以及主存储器 1208，提供了处理器、服务处理器、L2 高速缓存、主机到 PCI 桥以及主存储器之间的通信路径。服务处理器 1216 还可访问系统电源资源以便关闭信息处理设备 1201 的电源。

外围设备和输入/输出（I/O）设备可以被连接到各种接口上（例如，并行接口 1262、串行接口 1264、键盘接口 1268 和连接于 ISA 总线 1240 的鼠标接口 1270）。或者，可由一个连接到 ISA 总线 1240 上的超级 I/O 控制器（未示出）来接纳许多 I/O 设备。

为了将计算机系统 1201 连接于其它的计算机系统以便通过网络复制文件，LAN 卡 1230 被连接于 PCI 总线 1210。类似地，为了使用电话线连接将计算机系统 1201 连接到 ISP 以便连接到因特网，调制解调器 1275 被连接到串口 1264 和 PCI 到 ISA 桥 1235。

虽然图 12 中说明的计算机系统能够执行此处说明的过程，但是这个计算机系统只是计算机系统的一个例子。本领域的技术人员可以理解许多其它的计算机系统设计能够执行此处说明的过程。

本发明的一个优选实施方式是一个应用程序，也就是一个代码模块内的一组指令（程序代码），该代码模块可以，例如，驻留在计算机的随机访问存储器内。直到该计算机需要时为止，这组指令可以被存储在其它的计算机存储器内，例如，在硬盘驱动器上，或是在可移动存储装置中，诸如光盘（最后用于 CD ROM 中）或软盘（最后用于软盘驱动器）中，或通过因特网或其它计算机网络下载。因此，本发明可以被实现为用于计算机内的计算机程序产品。此外，虽然所说明的各种方法被方便地实现于由软件可选择地激活的或重新配置的通用计算机内，本领域的普通技术人员将会认识到，这些方法可以在硬件、固件或在被构造用来执行所需要的方法步骤的更专用的装置内执行。

虽然已经给出并说明了本发明的特定的实施例，对于本领域的技术人员来说显而易见的是，基于此处讲述，可以做出改变和改型而不脱离本发明和其更宽的方面，因此，所附的权利要求将在它们的范围内包括所有处在本发明的实际精神和范围之内的改变和改型。此外，应当理解，本发明完全由所附的权利要求定义。本领域的技术人员可以理解如果打算的是所引入的权利要求要素的特定数目，这种打算将被明确地表述在权利要求内，并且在没有这种表述的地方，就没有这种限制。为了帮助理解，作为一个非限制性的例子，下面所附的权利要求包含引导性的短语“至少一个”和“一个或多个”以便引出权利要求要素。然而，这种短语的使用不应被理解成暗示，用不定冠词“一”或“一个”引出的权利要求要素将包含这种被引出的权利要求要素的任何特定的权利要求限制为仅包括一个这种要素的发明，即使当该同一权利要求包括引导性的短语“一个或多个”或“至少一个”以及不定冠词诸如“一”或“一个”时；同样的规则对于定冠词权利要求中的使用也成立。

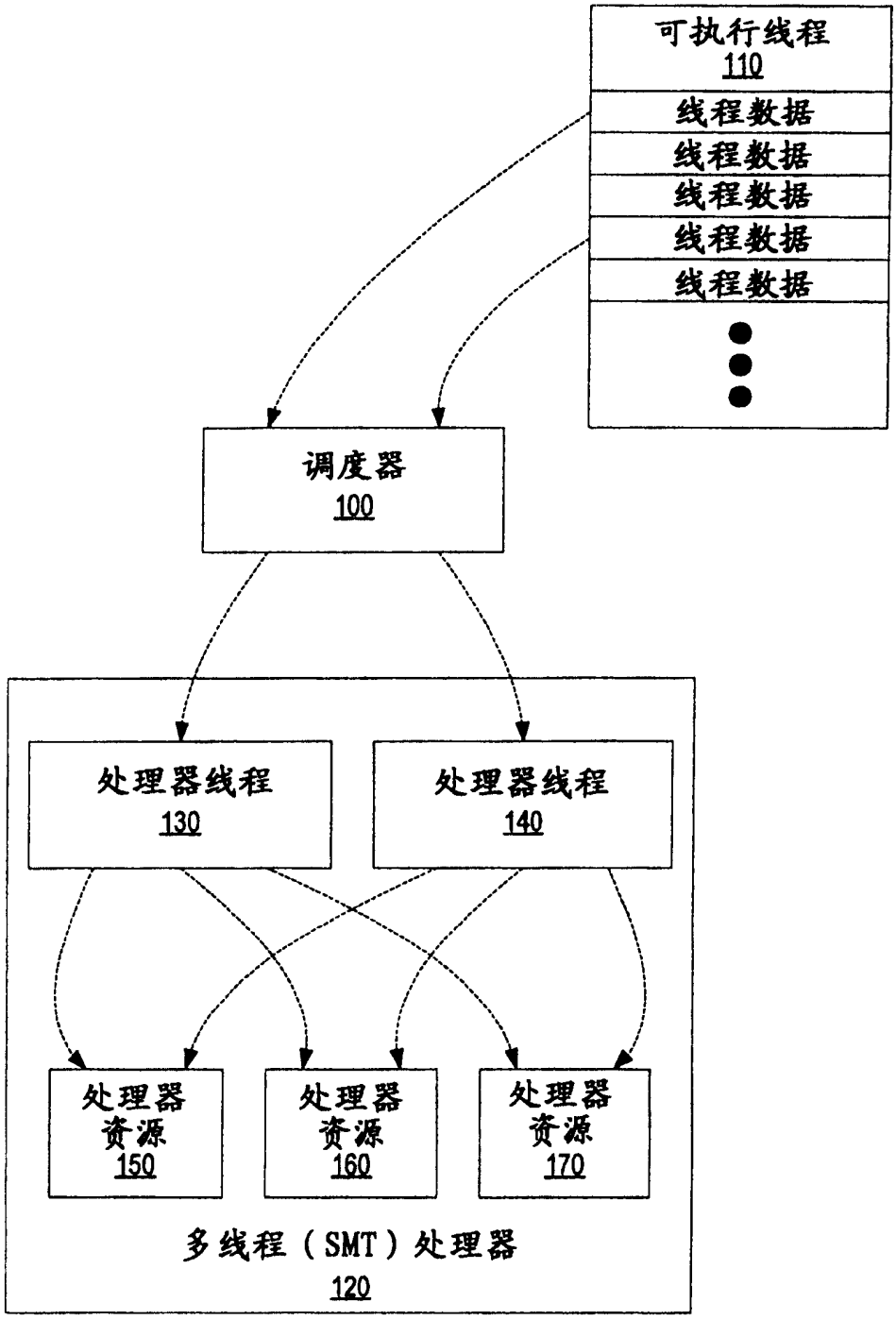

可执行线程
110
线程数据
线程数据
线程数据
线程数据
线程数据
调度器
100
处理器线程
130
处理器线程
140
处理器
资源
150
处理器
资源
160
处理器
资源
170
多线程（SMT）处理器
120

图 1

线程 ID 215	第 1 兼容线程 ID 220	第 1 兼容线程 CPI 225	第 2 兼容线程 ID 230	第 2 兼容线程 CPI 235	第 3 兼容线程 ID 240	第 3 兼容线程 CPI 245
aaa	bbb	0.7	ddd	1.0	fff	1.2
bbb	aaa	0.7	ccc	1.3	[null]	[null]
ccc	bbb	1.3	ddd	0.8	fff	1.5
ddd	aaa	1.0	ccc	0.8	ggg	0.9
eee	ddd	1.4	[null]	[null]	[null]	[null]
fff	aaa	1.2	ccc	1.5	[null]	[null]
ggg	ddd	0.9	[null]	[null]	[null]	[null]
hhh	[null]	[null]	[null]	[null]	[null]	[null]

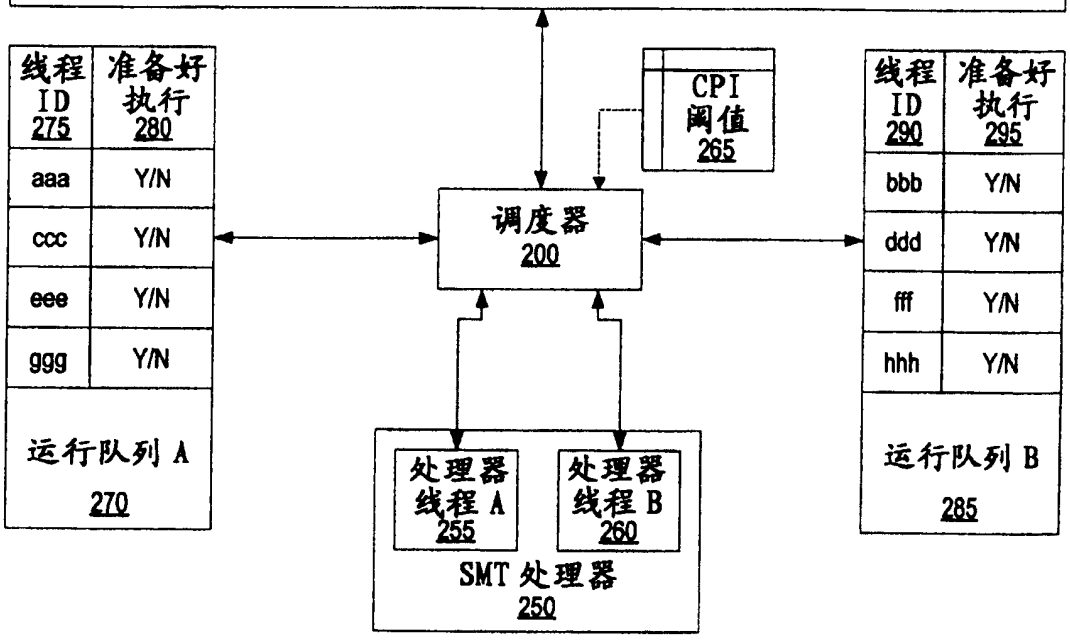


图 2

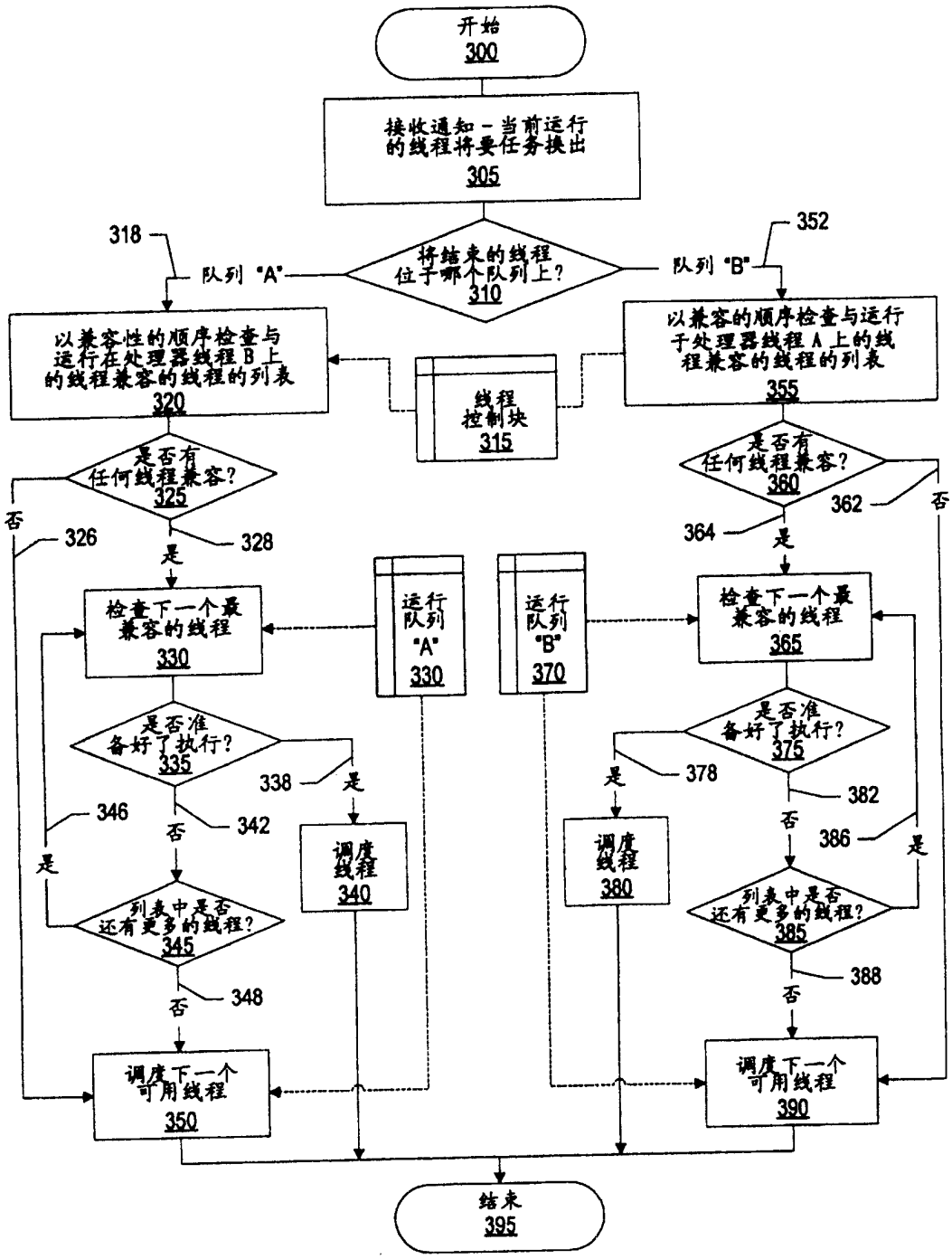

开始
300
接收通知－当前运行的线程将要任务换出
305
将结束的线程位于哪个队列上?
310
318
队列 "A"
352
队列 "B"
以兼容性的顺序检查与运行在处理器线程 B 上的线程兼容的线程的列表
320
线程控制块
315
以兼容的顺序检查与运行于处理器线程 A 上的线程兼容的线程的列表
355
是否有任何线程兼容?
325
否
326
是
328
是否有任何线程兼容?
360
362
否
364
是
检查下一个最兼容的线程
330
运行队列 "A"
330
运行队列 "B"
370
检查下一个最兼容的线程
365
是否准备好了执行?
335
338
是
是否准备好了执行?
375
是
378
382
否
346
是
否
342
调度线程
340
调度线程
380
386
是
列表中是否还有更多的线程?
345
列表中是否还有更多的线程?
385
否
348
388
否
调度下一个可用线程
350
调度下一个可用线程
390
结束
395

图 3

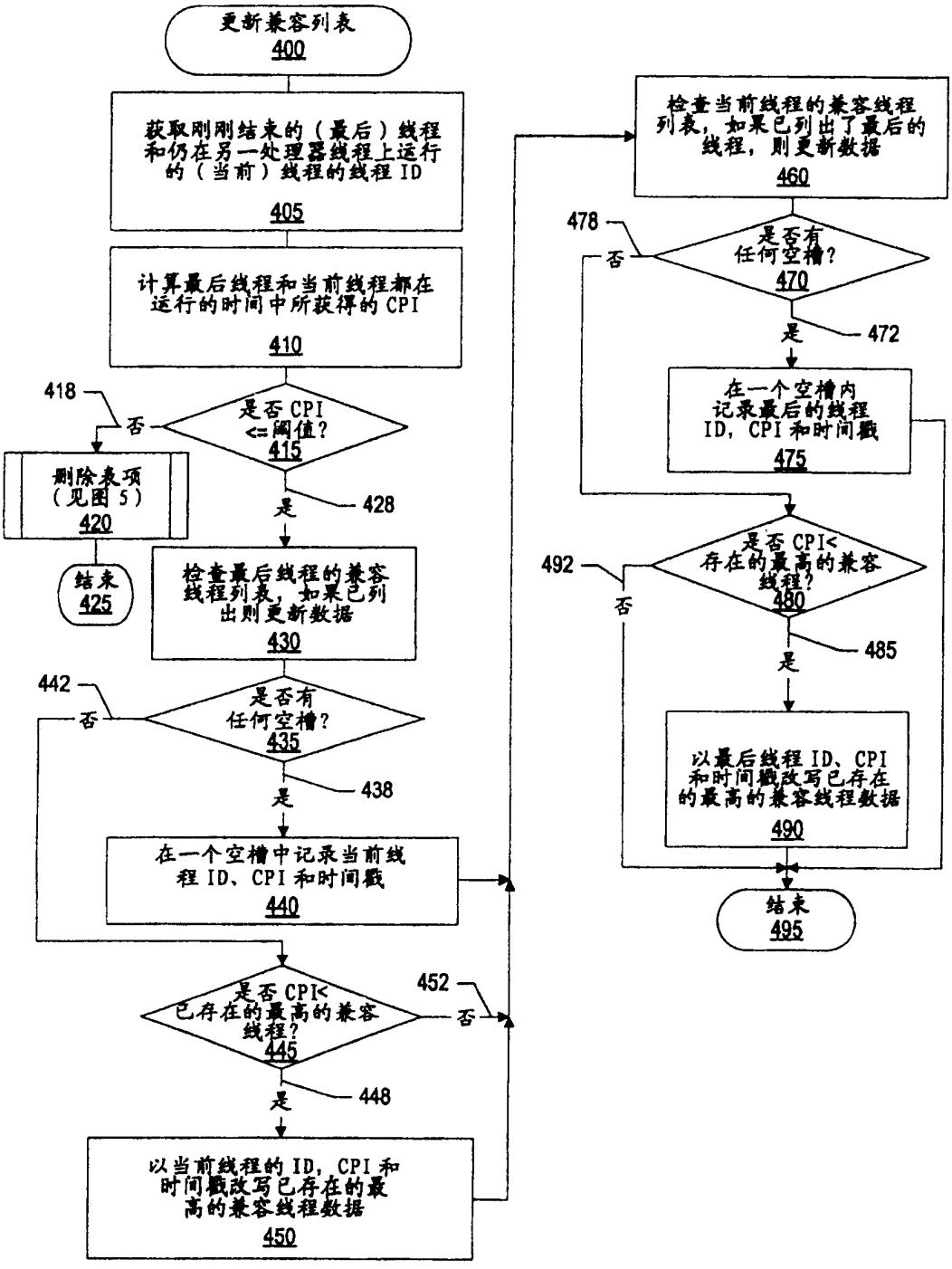

更新兼容列表
400
获取刚刚结束的（最后）线程和仍在另一处理器线程上运行的（当前）线程的线程 ID
405
计算最后线程和当前线程都在运行的时间中所获得的 CPI
410
是否 CPI <= 阈值?
415
418
否
是
428
删除表项（见图 5）
420
结束
425
检查最后线程的兼容线程列表，如果已列出则更新数据
430
是否有任何空槽?
435
442
否
是
438
在一个空槽中记录当前线程 ID、CPI 和时间戳
440
是否 CPI< 已存在的最高的兼容线程?
445
452
否
是
448
以当前线程的 ID，CPI 和时间戳改写已存在的最高的兼容线程数据
450
检查当前线程的兼容线程列表，如果已列出了最后的线程，则更新数据
460
是否有任何空槽?
470
478
否
是
472
在一个空槽内记录最后的线程 ID，CPI 和时间戳
475
是否 CPI< 存在的最高的兼容线程?
480
492
否
是
485
以最后线程 ID、CPI 和时间戳改写已存在的最高的兼容线程数据
490
结束
495

图 4

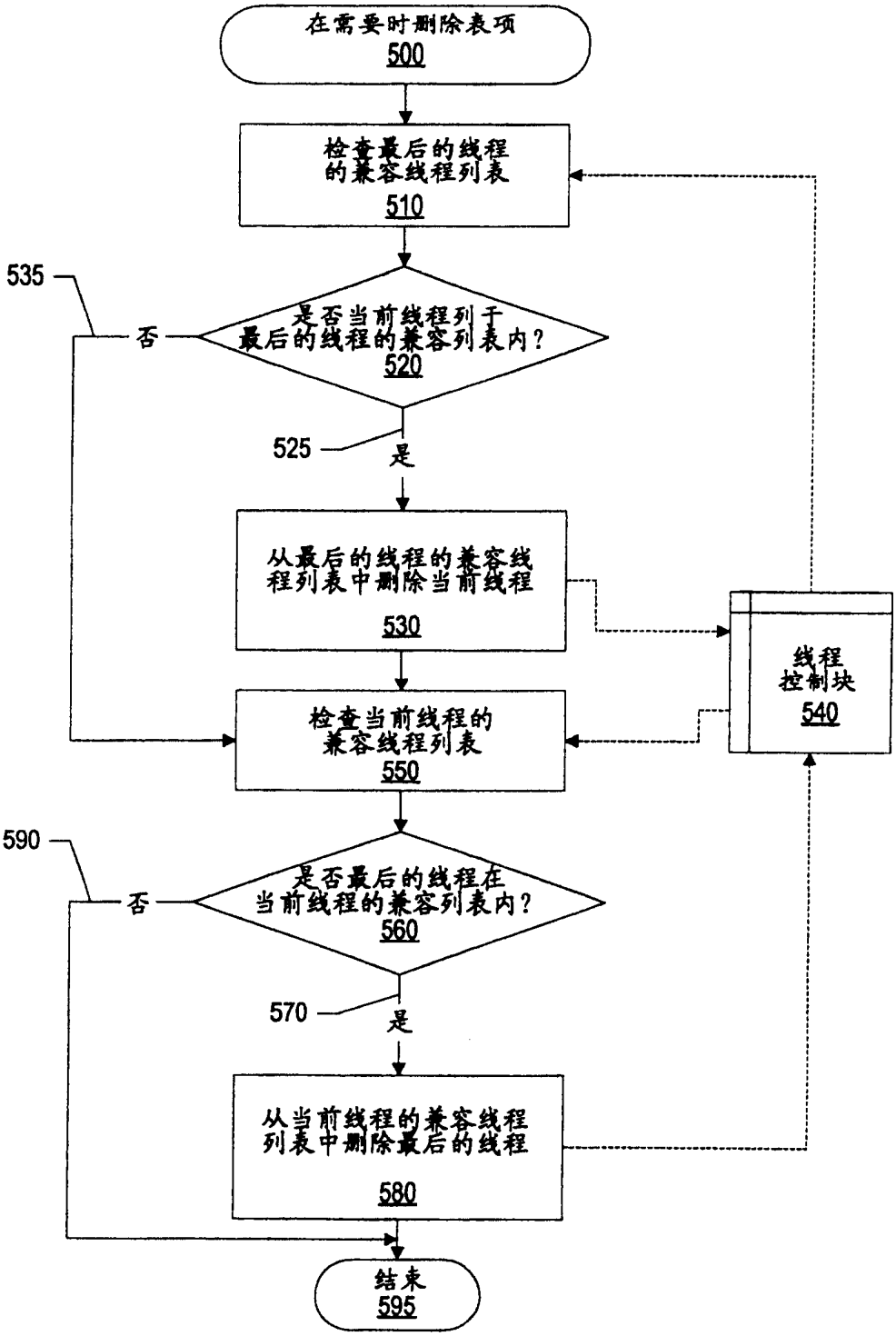

在需要时删除表项
500
检查最后的线程
的兼容线程列表
510
535
否
是否当前线程列于
最后的线程的兼容列表内？
520
525
是
从最后的线程的兼容线
程列表中删除当前线程
530
线程
控制块
540
检查当前线程的
兼容线程列表
550
590
否
是否最后的线程在
当前线程的兼容列表内？
560
570
是
从当前线程的兼容线程
列表中删除最后的线程
580
结束
595

图 5

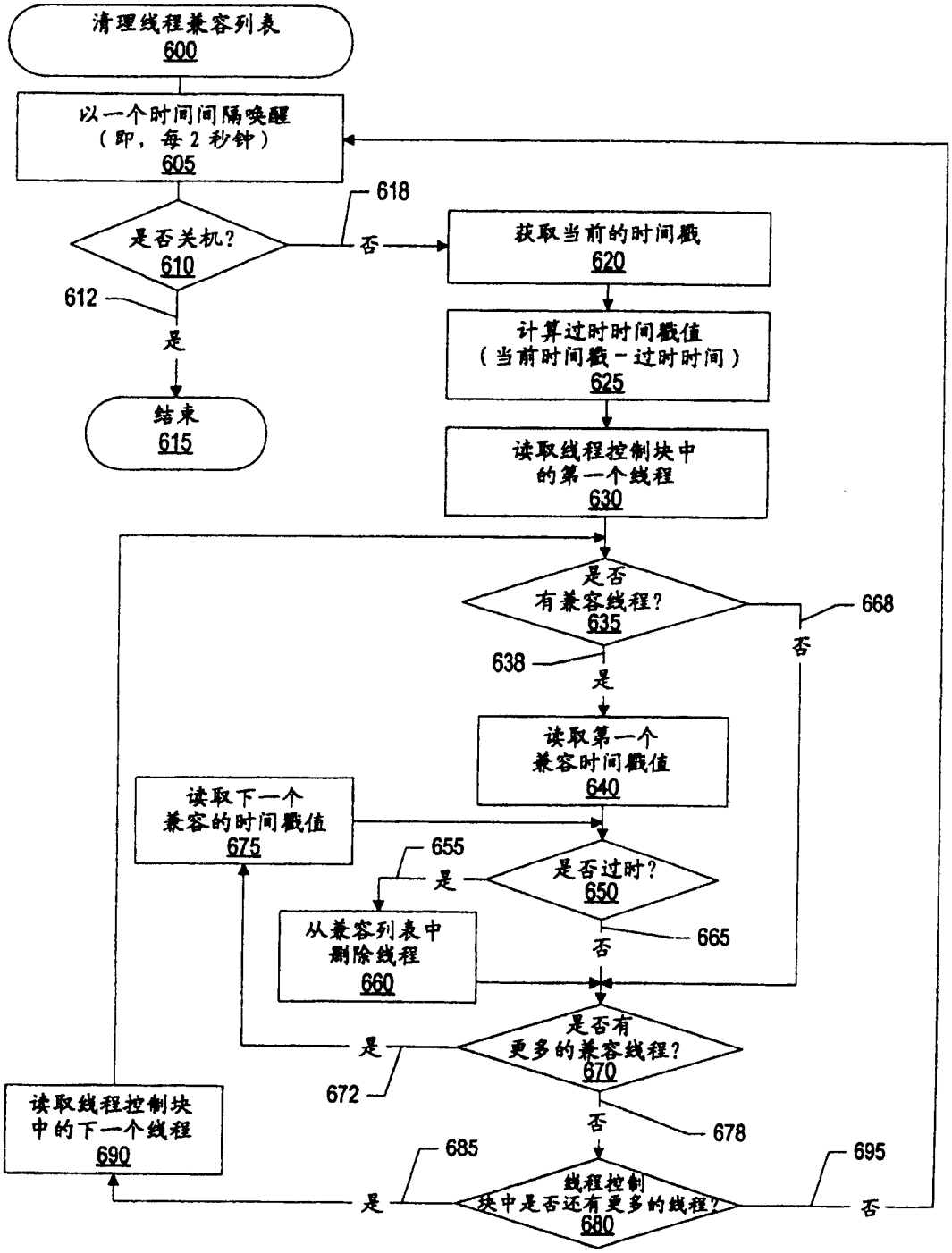

清理线程兼容列表
600
以一个时间间隔唤醒
（即，每2秒钟）
605
是否关机?
610
618
否
获取当前的时间戳
620
612
是
结束
615
计算过时时间戳值
（当前时间戳－过时时间）
625
读取线程控制块中
的第一个线程
630
是否
有兼容线程?
635
668
否
638
是
读取第一个
兼容时间戳值
640
读取下一个
兼容的时间戳值
675
655
是
是否过时?
650
从兼容列表中
删除线程
660
665
否
是否有
更多的兼容线程?
670
是
672
否
678
读取线程控制块
中的下一个线程
690
685
是
线程控制
块中是否还有更多的线程?
680
695
否

图 6

线程 ID 715	线程最后的 CPI 值 720	线程最后的 CPI 值 -1 725	线程最后的 CPI 值 -2 730	线程最后的 CPI 值 -n 735	线程的平均 CPI 740
aaa	0.7	0.3	0.4	0.6	0.5
bbb	0.5	0.7	0.8	0.4	0.6
ccc	1.0	0.6	0.9	0.7	0.8
ddd	1.1	0.8	1.2	0.9	1.0
eee	1.4	1.3	1.6	1.7	1.5
fff	2.4	2.5	2.7	2.8	2.6
ggg	1.8	2.4	1.9	2.3	2.1
hhh	2.7	2.8	2.8	2.9	2.8
线程控制块 710					

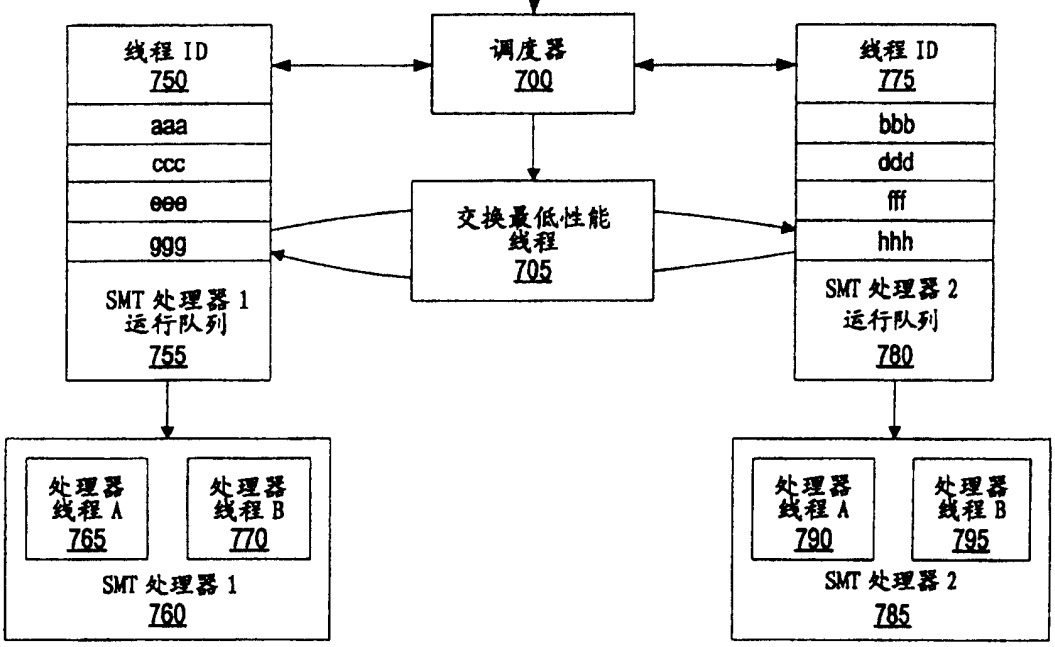


图 7

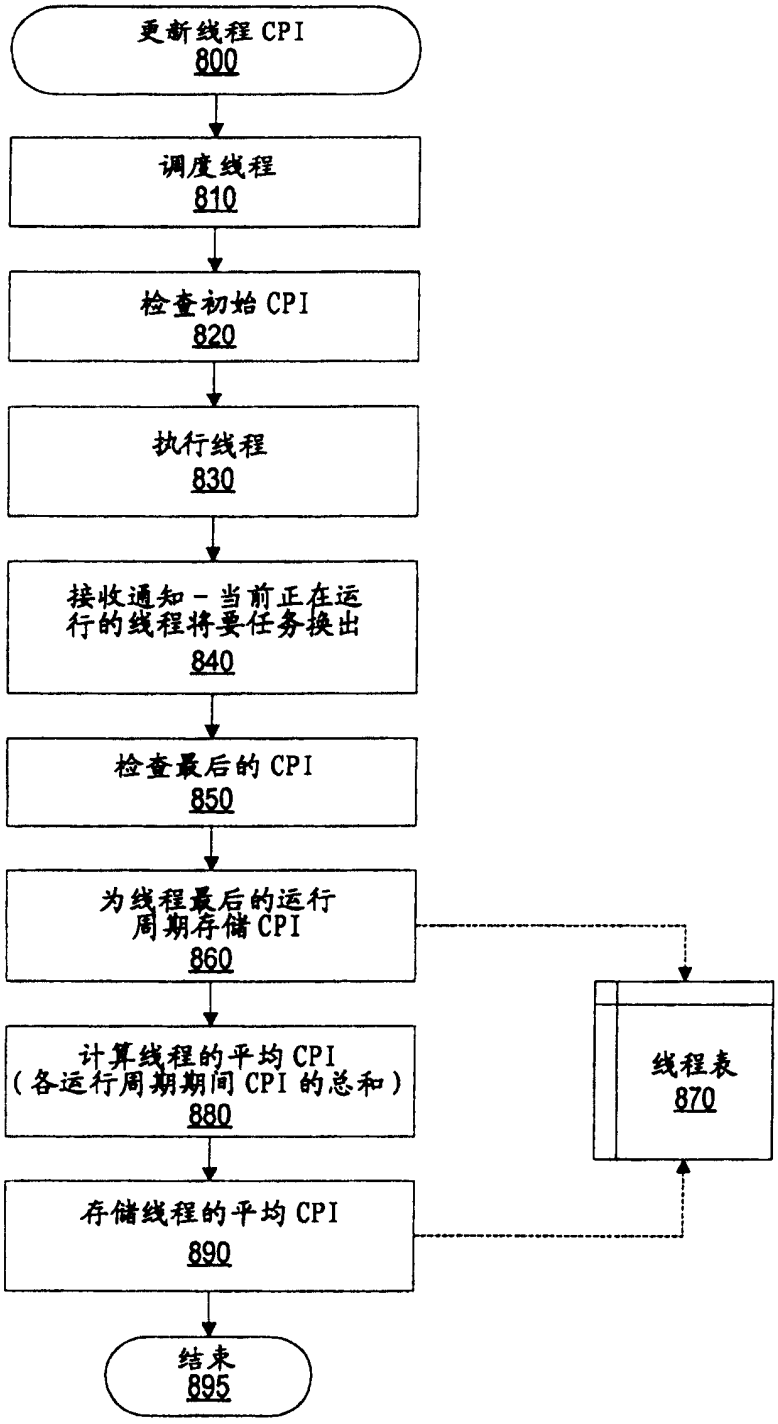

更新线程 CPI
800
调度线程
810
检查初始 CPI
820
执行线程
830
接收通知 - 当前正在运行的线程将要任务换出
840
检查最后的 CPI
850
为线程最后的运行周期存储 CPI
860
计算线程的平均 CPI
(各运行周期期间 CPI 的总和)
880
存储线程的平均 CPI
890
结束
895
线程表
870

图 8

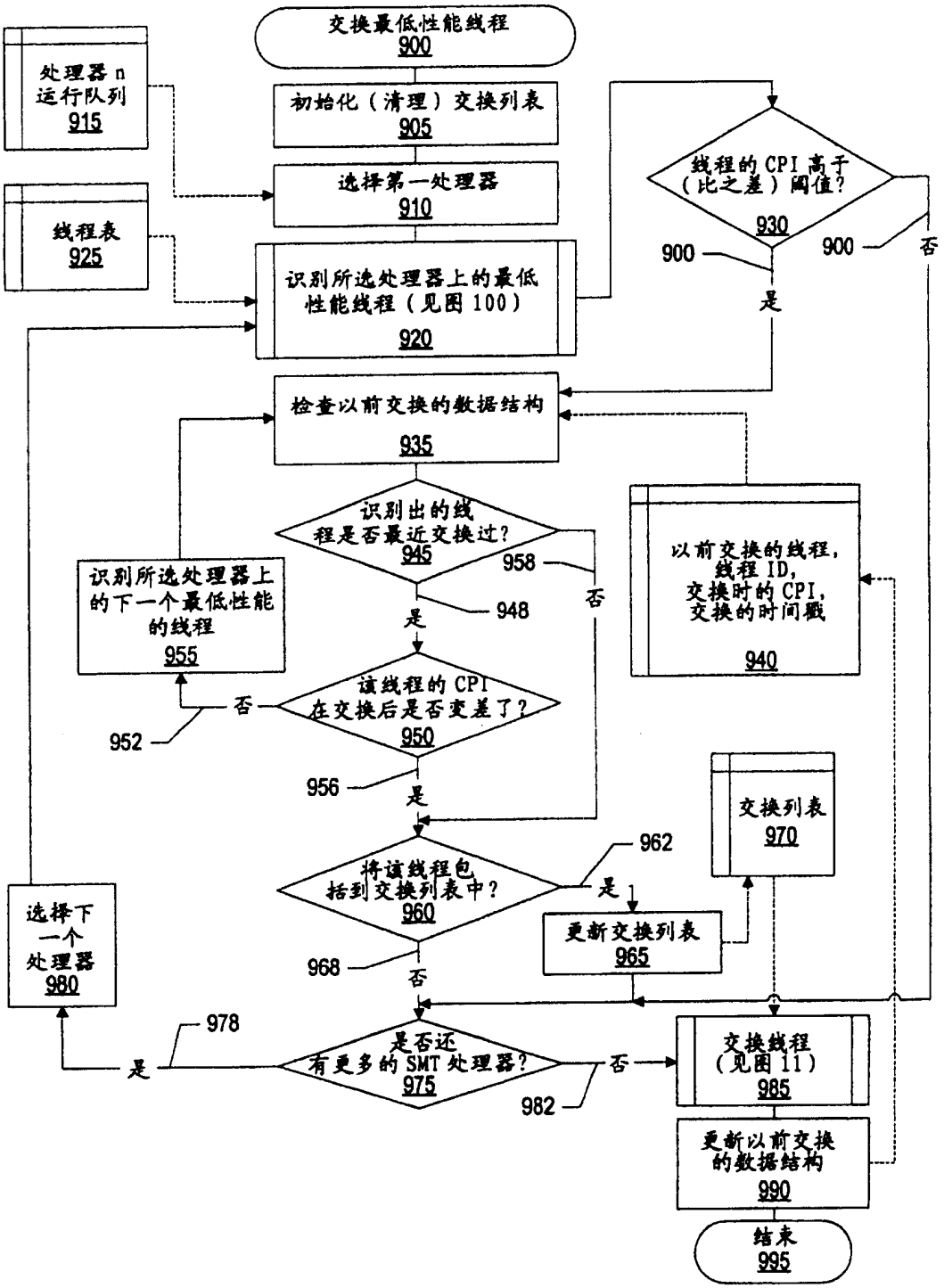

交换最低性能线程
900
处理器 n 运行队列
915
初始化（清理）交换列表
905
选择第一处理器
910
线程表
925
识别所选处理器上的最低性能线程（见图 100）
920
线程的 CPI 高于（比之差）阈值？
930
900
900
否
是
检查以前交换的数据结构
935
识别出的线程是否最近交换过？
945
958
否
948
是
以前交换的线程，线程 ID，交换时的 CPI，交换的时间戳
940
识别所选处理器上的下一个最低性能的线程
955
该线程的 CPI 在交换后是否变差了？
950
否
952
956
是
交换列表
970
962
将该线程包括到交换列表中？
960
是
更新交换列表
965
选择下一个处理器
980
968
否
978
是否还有更多的 SMT 处理器？
975
是
否
982
交换线程（见图 11）
985
更新以前交换的数据结构
990
结束
995

图 9

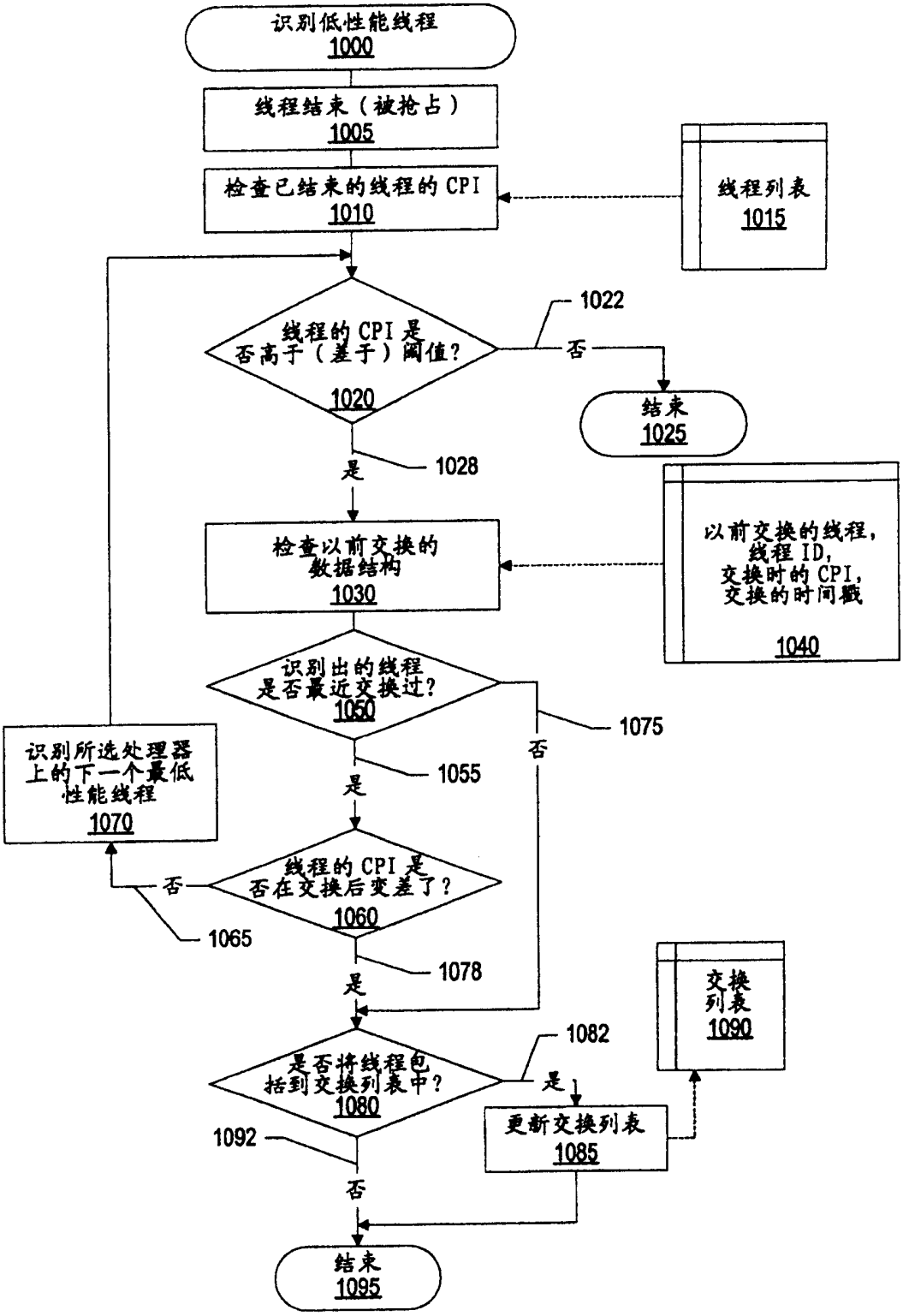

识别低性能线程
1000
线程结束（被抢占）
1005
检查已结束的线程的 CPI
1010
线程列表
1015
线程的 CPI 是否高于（差于）阈值？
1020
1022
否
结束
1025
是
1028
检查以前交换的数据结构
1030
以前交换的线程，线程 ID，交换时的 CPI，交换的时间戳
1040
识别出的线程是否最近交换过？
1050
1075
否
识别所选处理器上的下一个最低性能线程
1070
1055
是
线程的 CPI 是否在交换后变差了？
1060
否
1065
1078
是
交换列表
1090
是否将线程包括到交换列表中？
1080
1082
是
更新交换列表
1085
1092
否
结束
1095

图 10

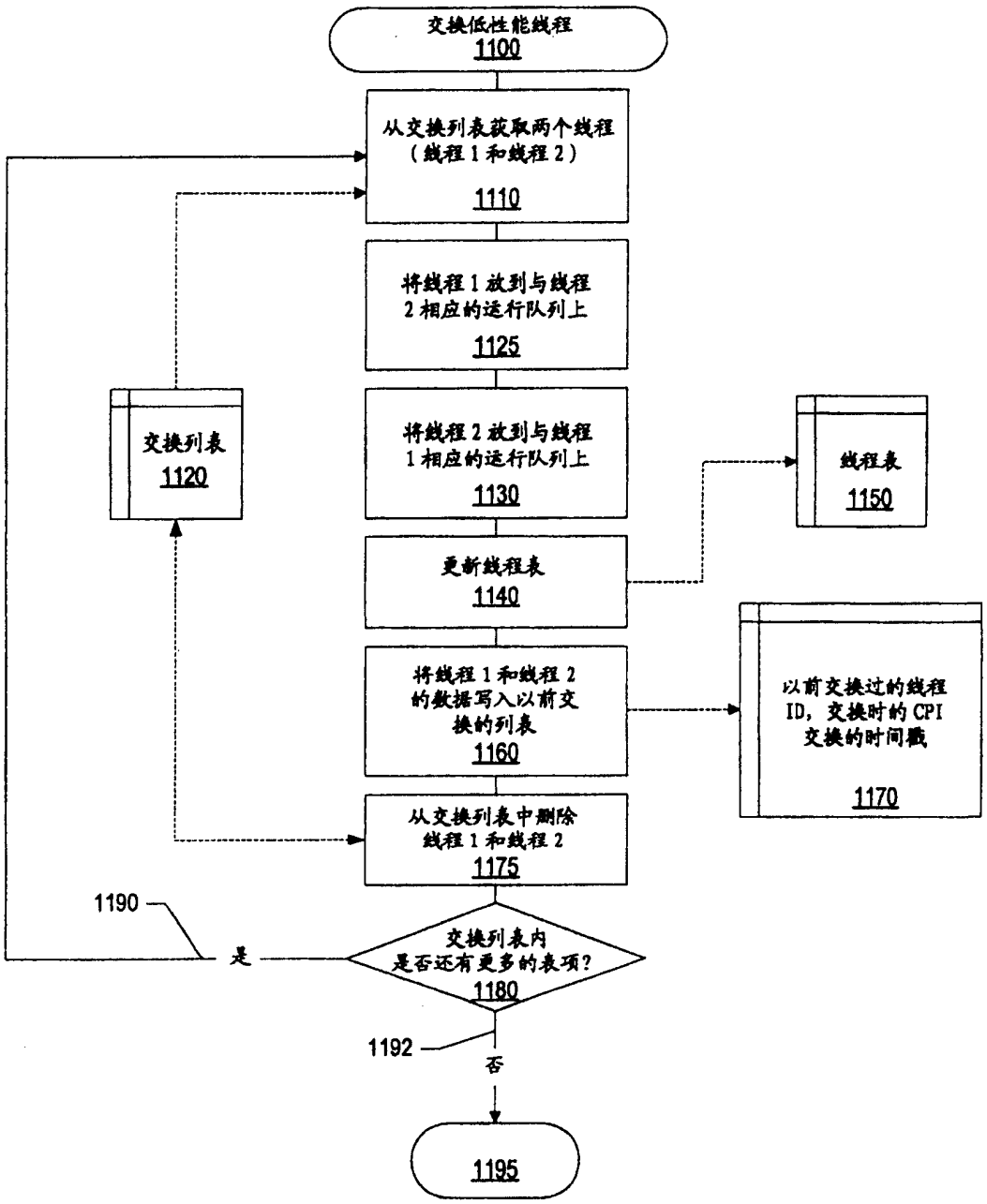

交换低性能线程
1100
从交换列表获取两个线程（线程 1 和线程 2）
1110
将线程 1 放到与线程 2 相应的运行队列上
1125
将线程 2 放到与线程 1 相应的运行队列上
1130
更新线程表
1140
将线程 1 和线程 2 的数据写入以前交换的列表
1160
从交换列表中删除线程 1 和线程 2
1175
交换列表内是否还有更多的表项？
1180
交换列表
1120
线程表
1150
以前交换过的线程 ID，交换时的 CPI 交换的时间戳
1170
1190
是
1192
否
1195

图 11

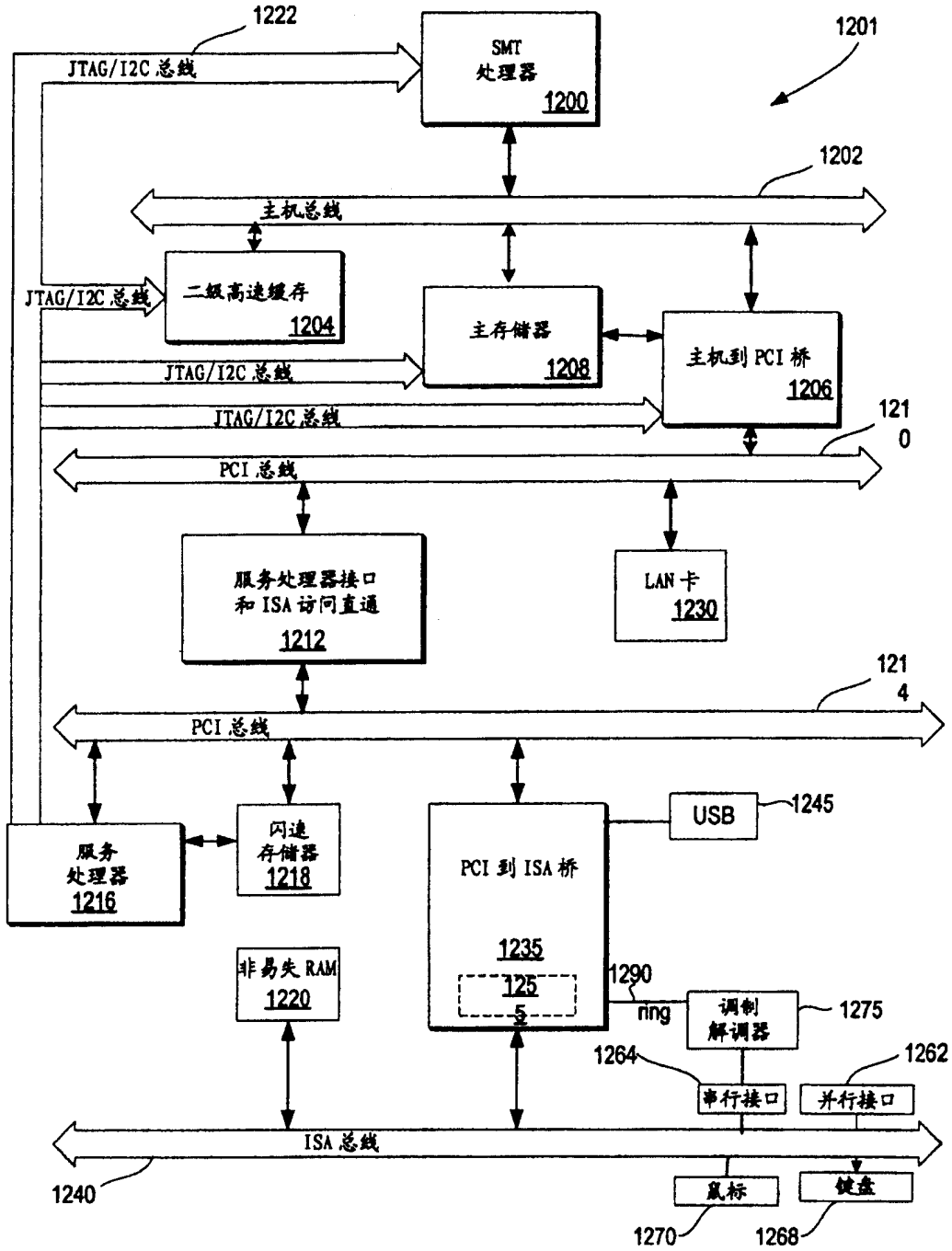

1222
JTAG/I2C 总线
SMT
处理器
1200
1201
1202
主机总线
JTAG/I2C 总线
二级高速缓存
1204
主存储器
1208
主机到 PCI 桥
1206
JTAG/I2C 总线
JTAG/I2C 总线
121
0
PCI 总线
服务处理器接口
和 ISA 访问直通
1212
LAN 卡
1230
121
4
PCI 总线
服务
处理器
1216
闪速
存储器
1218
PCI 到 ISA 桥
1235
125
5
USB
1245
非易失 RAM
1220
1290
ring
调制
解调器
1275
1264
1262
串行接口
并行接口
ISA 总线
1240
鼠标
键盘
1270
1268

图 12