面向线程级前瞻的线程划分方法浅析

鲁建壮　王志英　张春元
（国防科大计算机学院　长沙 410073）

摘　要　正确合理的线程划分方法是提取线程级并行性的必要前提，线程级前瞻技术是简化线程划分复杂度提高系统性能的重要手段。本文讨论了几种支持线程级前瞻的典型线程划分方法，在此基础上提出了线程级划分需要解决的关键问题，并结合一典型自动线程划分算法进行了具体分析，提出了线程划分需要进一步研究的问题。

关键词　线程级并行性，线程划分，线程级前瞻

Analysis of Thread Partition Method Oriented to Thread Level Speculation

LU Jian-Zhuang　WANG Zhi-Ying　ZHANG Chun-Yuan
(School of Computer, National University of Defense Technology, Changsha 410073)

Abstract　A proper thread-partition method is the precondition when extracting thread level parallelism. Thread level speculation can reduce the complexity of thread partition and improve the performance of the system. In this paper, several thread partition methods supporting thread level speculation are discussed, and key techniques are issued base on the discussion. Then thread partition methods are analyzed with an automatic thread partition algorithm in detail. At last, questions need to be studied fatherly are proposed.

Keywords　Thread level parallelism, Thread partition, Thread level speculation

1　引言

对微处理器性能的不懈追求促使各种软硬件技术不断地优化更新，目前通过挖掘指令级并行性(ILP)来提高性能的处理器仍然占据着主导地位，但伴随着挖掘深层次的 ILP 而不断增加的硬件设计复杂度、研发周期等诸多方面的负面影响也越来越明显。为了弥补深层次 ILP 难以开发的缺陷，缓和性能提升和硬件设计之间的矛盾，开发更高层次的线程级并行性(TLP)是一个很好的选择[1]。

当前运行于微处理器上的应用程序绝大部分为串行程序，必须对其进行线程划分才能在支持线程级并行的系统上高效运行。线程划分可以通过软件实现，也可以通过硬件实现，但通过硬件进行线程提取显然会增加硬件设计的复杂性和面积开销，因此大多数的多线程体系结构均采用软件方法进行线程划分，本文也将集中讨论基于软件线程划分方法。

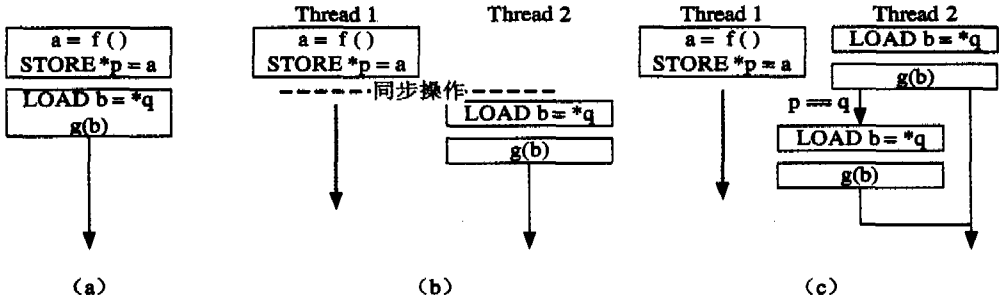

图 1　线程级前瞻执行

通过纯软件的方法进行线程划分在理论上是可行的，但在现实中却很困难，因为串行程序通常具有数据相关性强、控制关联紧密等特点，要挖掘其中的 TLP，将其划分为线程并行执行，必须采用有效手段解决其中的控制和数据相关性。解决这一难题的根本在于保证每个线程在执行过程中能够按照正确的顺序读取和存储数据，精确同步可能存在相关性的每次数据访问可以解决这一问题，并且是软件可以办到的，但是这样势必造成硬件利用率低下并最终导致性能上的损失，因为这其中有相当多的数据同步是没有必要的；完全通过软件手段准确地标定每次数据访问是否需要同步不仅工作量大而且是不现实的，因为有些数据的相关性在编译阶段根本没办法确定。采用线程级前瞻(TLS)技术，在不确定的情况下，提前创建并猜测执行一些将来可能需要执行的线程，通过前瞻执行和相应的前瞻错误检测机制可以去除不必要的数据同步、发现并维护真正的数据相关，从而降低线程划分的难度[1,3,4]。图 1 是一个线程级前瞻的例子，(a)为原串行执行的情况；(b)为采用同步时并行执行的情况，由于不能确定指针 q 是否等于 p，所以只能同步；(c)为采用线程级前瞻执行

鲁建壮　博士研究生，主要从事微处理器系统结构，芯片设计等方面的研究。

的情况。

从上面的分析和例子可以看出，线程级前瞻执行是降低线程划分难度，并提高线程级并行执行效率的有效途径，高效的线程划分方法离不开线程级前瞻执行的支持，本文的后续部分将依次论述面向线程级前瞻的线程划分方法的研究现状、关键问题和支持技术，并结合一典型线程划分算法进行具体分析，最后进行小结。

2 线程划分方法的特点和现状

线程划分方法是伴随着单芯片多线程并行处理体系结构的出现，为在当前存在的大量成熟的串行应用程序中提取线程级并行性而产生的一个新研究方向，当前没有成熟的方法和技术可以借鉴。

从表面上看它与传统的并行编译技术[6]都以提取和优化比 ILP 更高的并行性为目标，但它们之间存在本质的区别，首先两者处理的对象不同，现有的并行编译技术大都是针对并行程序设计语言所编写的并行应用程序进行并行优化或将串行应用程序向量化、并行化，本文所讨论的线程划分方法却是针对现有串行应用程序进行线程级并行性提取；其次它们针对的目标系统不同，并行编译技术以传统的多处理系统为目标进行编译优化，线程划分方法以单芯片多线程的处理器为目标结构线程划分，这两类系统在组织结构、连接通信方式等方面都存在很大的差异；最后研究内容存在差异，并行编译技术涉及的内容比较广泛，包括并行性提取，数据私有化、分布和通信优化等方面，而线程划分方法集中在将串行程序合理的划分为可以并行执行、调度和控制的程序段(即线程)。当然并行编译技术对程序的一些基本分析和处理方法方面的研究成果如对循环的处理、控制和数据的相关性分析等对线程划分研究也有一定的帮助。

当前针对线程划分方法的研究大都是伴随着多线程系统的研究进行的，不同的多线程结构甚至目标结构相似的不同研究小组在线程划分方法上都不尽相同。首先其目标系统不同，有同时多线程(Simultaneous Multithreading, SMT)和单芯片多处理器(Single Chip Multiprocessor, SCMP)、Superthreaded 等结构，它们的组织结构、执行方式都存在很大差异，相应的线程划分及优化的方法也就不同，但他们也有共同之处，即为了简化线程划分的复杂度，都支持线程级前瞻执行。

其次具体实现途径也不同，Stanford 的 Hydra 不仅仅面向通用串行应用，因此在处理针对串行应用的线程划分时，仅仅是借助并行编译器 SUIF 的指导通过手工实现[1]，没有深入研究以设计更好的方法；明尼苏达大学的 Superthreaded 结构在编译优化和 profile 信息的帮助下，通过手工插入特殊函数实现[2]；Illinois 的 SCMP 通过对二进制代码的循环等结构进行分析，而后对其进行标注实现[3]；文[4]通过构建控制流图，而后根据特定规则进行规约确定线程候选；文[5]通过分析基本块之间的可达概率、控制和数据相关性关系确定线程候选。

虽然这些线程划分方法存在诸多不同，但它们解决的是同一个问题，因此它们面临的关键问题是相同的，同时本文认为手工划分的方法只是权宜之计，在下一节讨论线程划分方法的关键问题时将以线程自动划分作为划分方法的最终目标。

3 线程划分方法的关键问题

3.1 线程划分的关键问题

线程级前瞻技术简化了线程划分的复杂度，在此基础上根据对现有线程划分方法的分析，本文认为线程划分必须解决好以下几个问题：

首先针对串行程序和线程要确定一个适合描述方法。描述方法的优劣直接影响到线程划分的质量和复杂度，作为一个好的描述方法要含有足够的信息量，同时又要简洁以利于后续的分析和优化。

其次进行线程划分时，需要同时以线程的体积、线程间的控制和数据相关性、线程间的访存负载平衡等作为选择线程候选的准则，只有体积相近、控制和数据相关性少、负载均衡的划分结果才能充分发挥多线程系统的性能。

再次根据上述准则确定启发式策略。针对一个具体的应用程序，可能存在最优化分结果，但无论采用控制流图描述、数据依赖图描述还是基本块的可达性描述，该过程的复杂度都是难以接受的，甚至是 NP 难的，因此本文认为线程划分最好的解决策略是通过合适的启发式策略求得近似最优划分。

最后，线程划分必须保证程序的完整性，涵盖整个应用程序；同时与线程级前瞻技术紧密配合保证最终执行的正确性。

3.2 线程划分的支持技术

良好的线程划分方法可以充分挖掘程序中的线程级并行性，大幅度提高处理器的性能，但这需要编译技术和体系结构的共同支持。

编译支持 函数处理是线程划分的一个难点，允许线程候选中包括函数调用，应用程序中诸多大小各异的函数调用会增加评估和平衡线程体积的难度；反之在每个函数调用点创建一个新线程，又会导致线程体积的波动太大，利用编译的 function-inlining 技术[7]可以很方便地将体积太小的函数直接插入到调用位置，从而简化线程划分。

指针的引入增加了分析和确定数据相关性的难度，将其判别工作完全交给硬件，势必增加硬件开销，影响系统性能，通过编译优化中的别名分析技术[7]可以解决这一矛盾；同时编译中的跨循环数据相关性分析、变量私有化等优化手段都会对线程划分产生积极的作用，缓和甚至消除部分跨线程的数据相关，最终提高系统性能。

体系结构支持 这里提及的线程与操作系统层面上的线程含义不同，它的主要意义在于是一个连续执行的指令序列，在调度机制上通常借助硬件实现，线程划分的最终结果就是在适当的位置插入线程控制指令，及维护线程现场的必要指令，为此，体系结构必须扩展指令集、修改流水线实现这些指令。而这些指令处理方式的设计，应以给处理器核带来的影响尽量小为原则。

线程级前瞻执行是简化线程划分复杂度的重要手段，为实现线程级前瞻执行，在体系结构上需要增加线程执行状态、前瞻数据等信息的标识和存储部件，设计并实现前瞻执行正确性监测机制和前瞻错误恢复机制。

在一些系统中，为了使线程划分更加合理，在硬件上增加专门的控制逻辑对线程划分的结果进行进一步的优化，如消除执行小粒度线程时频繁的线程控制操作造成的性能损失，可以通过动态扩展线程的体积实现。

总之，线程划分不是独立存在的过程，需要编译和体系结构的紧密配合：借助编译过程的一些优化策略，可以降低线程

划分的复杂度，并提高系统的整体性能；体系结构不仅是承接线程划分结构实现线程级并行的载体，而且其中的一些具体参数，线程控制和前瞻执行的延迟开销等也是线程划分时确定启发式规则的重要依据；如在设计线程体积的阈值时，应使其足够大以忽略线程的控制开销。

4 典型的线程划分算法分析

如前所述当前线程划分方法有很多，下面以一个典型的线程自动划分算法为例进行具体分析。该算法是由日本东京大学 Niko Demus Barli 等人提出来的，主要分两个步骤：第一，找出可作为线程的候选者(candidates)来；第二，从这些候选者中选出最终要划分的线程。

算法主要用基于结构分析(structural analysis)的方法从程序控制流图中找出线程的候选者。线程的候选定义为只有一个入口节点的控制流图子图：G〈Nodes，Edges，entry〉，其中 Nodes 表示该线程中所包含的节点，Edges 表示线程中所包含的控制流图的边，entry 表示线程的入口节点。

在线程候选的选择过程中，结构分析过程从控制流图中找出特定的控制流结构，这些结构有 if-then、if-then-else 和循环等，然后将它规约为一个新的节点来代替原有的组成该控制流结构的若干节点；而后在重建控制流图之后，重复这一过程，不断地寻找符合条件的特定控制流结构并将这些结构规约为一个节点，直至整个控制流图规约为一个节点，从而整个控制流图转变为一个树结构。具体过程如图 2 所示。

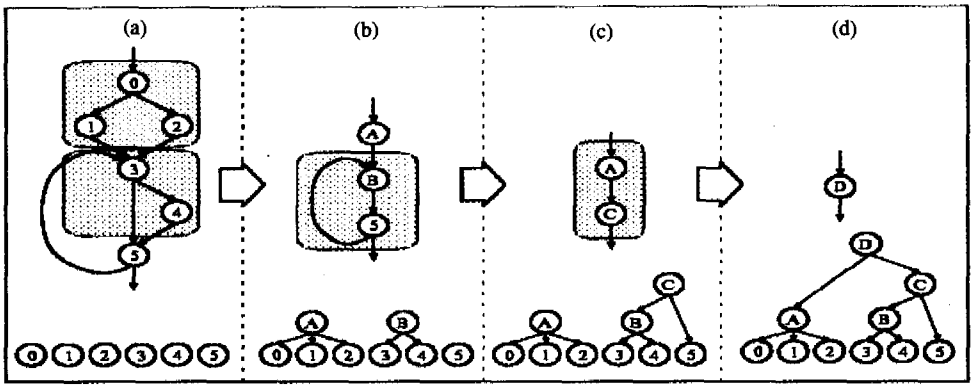

图 2 结构分析实例

程序的控制流千差万别，在规约过程中未必只有一条规约路径，这时就要借助启发式的策略得到一个近似最优解，本算法的做法是采用前序遍历的方法分析控制流图，同时针对线程体积平衡、函数调用优化和不规则循环等采取特殊的处理和规约规则指导规约的方向以达到较理想的规约效果。

结构分析完成后，树中的节点即可作为线程的候选。线程划分时很难知道程序执行时会发生什么事情，因此这里同样只能运用一些启发策略来选择线程使得尽可能地得到一个较好的结果。该算法认为线程的体积对于系统的性能影响很大，因此将线程的体积作为首要的启发式规则，同时将循环结构作为优先考虑的对象。

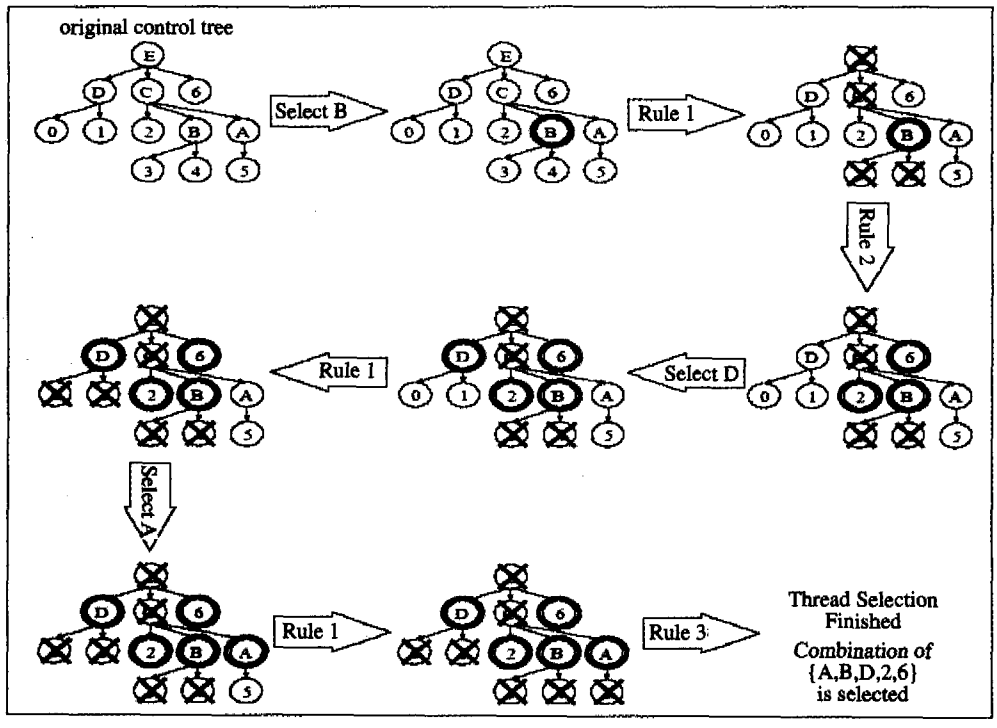

图 3 树着色规则

在线程选择过程中，为保证线程划分的完整性还确定了

(下转第 281 页)

设置为 f_1，否则设置为 f_2；第 7 至 12 行代表在无错情况下，以选定的速度执行任务，当发现错误后回卷任务并更新 R_N 和 R_d 的值，然后转到第 13 行；第 13 至 15 行代表出错后判断若能以速度 f_1 完成任务，则处理器速度设置为 f_1，否则设置为 f_2。该算法在完成任务的条件下尽可能让处理器低速工作，从而节约能量。

总结 以 DMR(double modular redundancy)系统为例，从插入额外 CCP 和 SCP 检查点的角度，本文提出了一种集成 CCR、任务重用和电源 DVS 技术的一种提高系统性能的方法。在此基础上，进一步提出了速度可适配算法，在提高任务完成率的条件下，节约系统的能量。本文提出的方法同样也适用于其它任务重复系统，如 TMR-F、DMR-F-1 和 RFCS 等。仿真结果表明与原有的方法相比，所提出的方法明显减少了任务的执行时间。下一步将改进本算法，使之适用于多任务环境。

参 考 文 献

1 Duda A. The effects of checkpointing on program execution time, Information processing Letter, 1983, 16(6): 221～229
2 李凯原，杨孝宗. 提高用任务重复的检查点方案的性能. 电子学报，2000，28(5)：33～35
3 周双娥，袁由光，熊兵周，等. 基于任务复制的处理器预分配算法. 计算机学报，2004，27(2)：216～223
4 杨金民，张大方. 一种基于虚拟对象的进程检查点实现方法. 系统仿真学报，2004，16(6)：1354～1357
5 Li Guo-Hui, Wang Hong-Ya. A novel min-process checkpoinging scheme for mobile computing systems. Journal of systems architecture, 2005, 51: 45～61
6 刘建，汪东升，沈美明，郑纬民. 一种基于检查点的并行程序调试器的设计与实现. 计算机研究与发展，2002，39(12)：1580～1586
7 Ziv A, Bruck J. Performance Optimization of Checkpointing Schemes with Task Duplication. IEEE Trans. Computers, 1997, 46(12): 1381～1386
8 Sayori N, Satoshi F, Naohiro I. Optimal checkpointing intervals of three error detection schemes by a double modular redundancy. Mathematical and computer modeling, 2003, 38: 1357～1363
9 Kimura M, Yasui K, Nakagawa T. Optimal checkpointing interval of a communication system with rollback recovery. Mathematical and computer modeling, 2003, 38: 1303～1311
10 Intel Corp. SpeedStep. http://developer. intel. com/mobile/, PentiumIII, 2003
11 Demers Y F, Shenker S. A scheduling model for reduced CPU Energy. Proc. IEEE Ann. Foundations of Computer Science, 1995. 375～382
12 Zhang Y, Chakrabarty. Energy-Aware Adaptive Checkpointing in Embedded Real-Time Systems, In: Proc. of the design, automation and test in Europe Conference and Exhibition, 2003
13 Melhem R, Mosse D, Elnozahy E. The interplay of power management and fault recovery in real-time systems. IEEE Tran. On computers, 2004, 53(2): 217～231
14 Ziv A, Bruck J. Analysis of checkpointing schemes with task duplication. IEEE Trans. Computer, 1998, 37(2): 222～227
15 George M A, Burns A. An optimal fixed-priority assignment algorithm for supporting fault-tolerant hard real-time systems. IEEE Transactions on computers, 2003, 52(10): 1332～1346

(上接第 272 页)

一些准则，在该算法中以树着规则(Tree Coloring Rules)的形式给出，图 3 是应用树着色规则的一个实例，具体的规则如下：

• 如果树的一个节点被选为线程，那么它的祖先节点和后继节点便不能作为线程。这是为了保证所选出来的线程之间不存在重叠的部分。

• 如果树的一个节点被决定不能作为线程，那么它的后继节点中不存在后继节点的，就必须作为线程。这是为了保证所选出来的线程能覆盖整个控制流图。

• 当所有的节点都被决定了是作为线程还是不作为线程后，该过程结束。

采用上述算法对 spec95int 程序进行线程划分，实验结果表明同规模的单芯片多处理器比超标量结构性能提高 50%～80%。

算法设计者研究中发现上述算法的分析和划分方法，决定了体积过小的线程是难以避免的，为了解决这一问题，在目标结构的硬件实现中，增加了线程动态扩展机制，即将多个小线程合并成为一个大线程的监测和控制机制。最终实验表明采用这一优化措施比不采用这一措施又有 8.5%～16%的性能提高。

从整体上看，上述算法对于 3.1 节中所提到的线程划分的关键问题都给出了相应的解决措施，并针对自身难以解决的问题，借助硬件技术进行了再优化。但该算法仍存在一些不足之处，在建立线程的描述方法时，没有考虑程序的 profile 信息，这使线程划分难以定位在程序执行频率最高的部分进行划分和优化；在结构分析和线程的选择过程中没有将线程间的数据相关性作为原则之一，而线程之间的相关性直接影响到线程前瞻的执行效率；最后在划分过程中没有很好的借助编译技术，影响了算法的复杂度和最终效果。

小结 微电子技术的进步使单芯片多线程处理成为一种有潜力的微处理器结构，线程划分是串行程序在其上面高效执行的必要途径，线程级前瞻执行可以大大降低线程划分的复杂度。综合现有的线程划分方法，本文提出了线程自动划分必须解决的关键问题，并结合一个典型算法进行了具体实现分析。通过分析本文认为线程划分方法还存在不成熟和需要进一步研究的方面，主要包括：

• 影响线程划分质量的因素很多，怎样才能综合考虑这些因素？

• 编译技术长于分析，如何将更多快捷的编译技术，如指令的调度策略与线程划分过程相结合？

• 多线程执行微处理器结构各异，是否可以设计一个通用的线程划分方法，通过简单的参数调整就可以适用于各种系统？

参 考 文 献

1 Hammond L, Nayfeh B A, Olukotun K. A Single-Chip Multiprocessor. IEEE Computer Special Issue on "Billion-Transistor Processors", Sept. 1997. 79～85
2 Tsai J Y, Huang J, Amlo C, Lilja D J, Yew P C. The Superthreaded Processor Architecture
3 Tullsen D M, Eggers S J, Levy H M. Simultaneous Multithreading: Maximizing On-Chip Parallelism. In: Proc. of the 22nd Int'l Symposium on Computer Architecture(ISCA), 1995. 392～403
4 Barli N D, Mine H, Sakai S, Tanaka H. Thread Partitioning Method for Speculative Multithreading Architectures. 修士论文东京大学大学院工学系研究科，Feb. 2002
5 Marcuello P, González A. Thread-Spawning Schemes for Speculative Multithreading. HPCA, Feb. 2002. 55～64
6 陈火旺，刘春林，谭庆平，赵克佳，刘越. 程序设计语言编译原理. 国防工业出版社，2000
7 Stallman R M. Using the GNU Compiler Collection for GCC3. 3 Dec. 2002