基于模糊聚类的推测多线程划分算法

李远成 阴培培 赵银亮
(西安交通大学计算机科学与技术系 西安 710049)

摘 要 推测多线程(Speculative Multithreading,SpMT)技术是一种实现非规则程序自动并行化的有效途径.然而,如何有效评估由诸如控制、数据依赖等因素导致的多种并行开销并实现最优线程划分一直是制约加速比性能提升的关键问题.基于启发式规则的传统划分方法虽然可以取得一定的加速效果,但由于启发式规则只能对多种并行开销进行定性评估,因而导致只能得到经验上较优的线程划分.针对传统划分方法的局限性,文中首次提出并实现了一种基于模糊聚类的线程划分方法.在该方法中,作者首先提出一种评估模型来定量评估各种并行开销,然后通过深入分析各种并行开销来确定最佳的线程解搜索空间,最终利用聚类方法实现有效线程解空间搜索以求取更优的线程划分.基于 Olden 程序集的测试结果表明,文中提出的线程划分方法可以有效地对非规则程序进行划分,其平均加速比可达到 1.85.

关键词 推测多线程;线程划分;模糊聚类;自动并行化;代价评估
中图法分类号 TP314 **DOI 号** 10.3724/SP.J.1016.2014.00580

A FCM-Based Thread Partitioning Algorithm for Speculative Multithreading

LI Yuan-Cheng YIN Pei-Pei ZHAO Yin-Liang
(*Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an* 710049)

Abstract Speculative multithreading (SpMT) technology is an effective mechanism for automatic parallelization of irregular programs. Selecting optimal thread partitioning solution by effectively assessing the speculative parallelization overheads is a key issue for the speedup performance improvement. However, most existing thread partitioning methods are still based on simple heuristics rules to select speculative threads. Because these heuristics rules can only be used to indirectly estimate the speculative multithreaded execution overheads and simply tackle individual overheads, these methods can only get the empirical optimal speculative thread solution. To overcome the limitation of these existing methods, in this paper, we first propose a fuzzy c-means (FCM) algorithm based thread partitioning method which can be used to effectively search the solution space of speculative threads and get the optimal solution. In this method, we effectively determine the solution space of speculative threads based on the analysis of several overheads. Meanwhile, we design and implement the cluster validity function by building a cost estimation model. The experimental results show that the proposed method can effectively search the solution space of speculative threads and we can indeed get better performance. On average, we achieve an average speedup of 1.85 on Olden benchmark suits.

Keywords speculative multithreading; thread partitioning; fuzzy c-means clustering; automatic parallelization; cost estimation

收稿日期:2011-12-31;最终修改稿收到日期:2013-10-16.本课题得到国家"八六三"高技术研究发展计划项目基金(2008AA01Z136)和国家自然科学基金(61173040)资助.**李远成**,男,1981 年生,博士研究生,主要研究方向为计算机体系结构、并行计算、机器学习.E-mail:yuanch_li@126.com.**阴培培**,女,1985 年生,硕士研究生,主要研究方向为并行计算.**赵银亮**,男,1960 年生,博士,教授,中国计算机学会(CCF)高级会员,主要研究领域为程序语言设计理论与实现、并行计算、人工智能.

1 引　言

挖掘并行性是提高串行程序执行性能的有效方法之一. 随着超标量(Superscalar)和指令级并行(ILP)等技术遇到越来越多的瓶颈以及片上多处理器(CMP)的迅速发展,线程级并行(TLP)逐渐成为一个更佳的选择. 近年来,推测多线程(SpMT)技术[1-3]作为一个能有效开发非规则程序并行性的线程级并行技术,已经得到了迅速发展. SpMT 技术一般采用软硬件协同设计的方法实现串行程序的并行执行. 在 SpMT 系统中,编译器通过在程序中插入边界指令和推测指令将程序划分成多个推测线程,并在存在大量控制和数据等依赖的情况下,以激进的方式挖掘线程级并行性. 在推测执行过程中,由执行模型检测控制和数据等依赖的发生,并采取撤销和重新运行等硬件手段来保证程序执行的正确性.

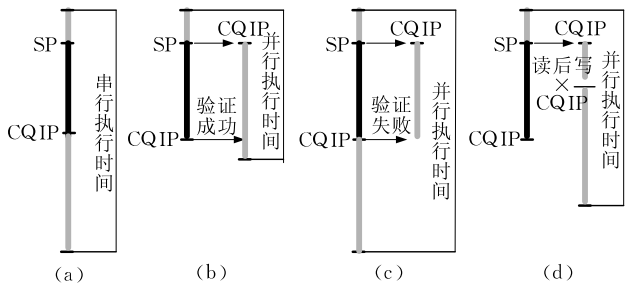
SpMT 技术在有效提升程序执行性能的同时,诸如控制、数据依赖等导致的多种并行开销成为制约程序加速比进一步提高的重要因素. 在 SpMT 系统中,线程划分是 SpMT 编译器的核心,线程划分的结果直接决定了推测多线程编译器的最终性能. 因此,有效评估诸如控制、数据依赖等因素导致的多种并行开销并实现最优推测多线程划分将是提升加速比性能的主要途径之一. 目前,研究者已经提出了多种推测多线程划分方法[4-5]. 同时,诸多优化技术如值预测[6]和分支预测[7]等技术也被大量提出用以进一步消减线程间的数据和控制等依赖造成的开销. 从整体上看,这些划分方法都取得了一定的加速效果,但却存在以下局限性:(1) 现有线程划分方法的核心都是基于启发式规则,而这些基于经验性的启发式规则只能对多种并行开销进行定性评估,这将导致现有的划分方法只能得到经验上较优的线程划分解;(2) 虽然有些方法通过建立动态评估模型在一定程度上克服了定性评估的不足,但是其低效的评估方法严重限制了此类方法的进一步应用.

针对现有线程划分方法存在的局限,为了更加深入探索线程划分影响程序加速比的内在规律,本文首次提出一种可以有效搜索线程划分解空间以寻求更优解的线程划分方法,即基于模糊聚类(Fuzzy Clustering Method,FCM)的线程划分方法. 该方法首先通过深入分析各种影响加速比性能的并行开销,并结合启发式规则确定有效的线程划分解的搜索空间. 然后,基于对多种并行开销的分析,提出一种定量评估多种并行开销的评估模型来判定聚类结果的有效性. 在此基础上,利用 FCM 算法对线程解空间进行搜索,进而求取更优的线程划分. 最后,本文将提出的划分方法在项目组开发的 Prophet 编译系统平台[8]上进行了实现. 基于 Olden 测试程序集[9]的实验结果表明,本文提出的线程划分方法可以有效地对非规则程序进行推测多线程划分,相对于传统的基于启发式规则的线程划分方法,本文方法可以取得平均 9.9%加速比性能提升,其平均加速比值达到了 1.85.

2 推测多线程技术

2.1 SpMT 执行模型

在 Prophet 并行编译器中,串行程序被划分成为多个推测线程进行执行,每个推测线程执行程序的不同部分, 程序的串行语义用以保证推测线程的提交顺序. 在并行推测执行中,有且只有一个是非推测线程,该线程可以提交其执行结果,代表程序当前确定执行的状态. 其它线程为推测线程,由串行程序代码片段及其预计算片段(pre-computation slice, p-slice)构成,线程间以前驱和后继的形式保持串行程序的语义. p-slice 是由编译器根据程序切片技术生成的一小段代码,用来对推测线程使用的 live-ins 变量(指线程体使用但并非由该线程定义的值)进行值预测. 如图 1 所示.



(a) 忽略SP和CQIP;
(b) 验证成功确定线程确定执行权利转交给推测线程,此后推测线程以确定方式执行;
(c) 验证失败推测线程撤销,确定线程继续执行它的代码段;
(d) 推测线程遇到RAW内存依赖违规,推测线程重启.

图 1　推测多线程执行模型

激发点(Spawn Point, SP)和准控制无关点(Control Quasi-Independent Point,CQIP)指令对唯一确定一个激发线程对. 串行程序中插入 SP-CQIP 点就被映射为推测多线程程序,SpMT 程序中忽略 SP-CQIP 就得到串行程序(图 1(a)). CQIP 点把程序分成一些代码段,当程序执行到 SP 点的时候,会发起一个新的线程并推测执行 CQIP 点之后的那个代码段. 执行过程中使用执行模型检测线程的运行

情况，如果线程推测执行成功，那么推测线程一直执行直到自身的 CQIP 点，如图 1(b)所示. 验证失败或读后写(Read After Write，RAW)内存依赖违规等导致推测执行失败. 若验证失败则由父线程串行执行验证失败的代码段，如图 1(c)所示，否则，在当前的状态下重新启动该线程，再次谋求推测执行成功，如图 1(d) 所示.

2.2 推测并行开销

在 SpMT 并行执行模式下，主要有 5 种开销影响推测多线程的并行性能. 这 5 种开销分别为：线程分发和提交开销、线程间通信开销、缓冲区溢出开销、线程撤销和重启开销以及线程负载不平衡开销等. 在这 5 种开销中，缓冲区溢出开销、线程撤销和重启开销以及线程负载不平衡开销是影响并行性能最大的 3 种开销[10].

线程分发和提交开销主要是由调度一个新线程到处理单元和将推测缓冲区数据向安全存储器传输并更新等时间组成. 此开销主要取决于硬件总线带宽以及线程需要处理的数据量的大小等因素. 线程间通信开销主要由处理单元等待前驱线程进行值传递的时间构成. 此开销主要依赖于体系结构对同步寄存器或者内存通信的支持，同时也受到线程间通信点位置以及通信频率的影响. 推测缓冲区溢出开销则主要是由于推测缓冲区溢出处理单元保持等待状态直至成为非推测或者被撤销所引起的时间组成. 此开销主要和缓冲区物理大小、缓冲区的组织结构以及线程体大小有关. 对于以上 3 种开销来说，尽管开销的大小在一定程度上会受到不同划分策略的影响，但其共同特点就是更多地受到硬件体系结构因素的影响，例如总线互联机制、缓冲区大小、高速缓存以及内存的访问机制等等.

线程撤销和重启开销主要由于发生控制和数据依赖推测失败所导致. 如图 2 所示，当线程 0 验证线程 1 时，发现线程 0 实际执行时的直接后继不是线程 1 或者线程 1 所使用的预测数据是错误的，此时将发生控制或数据依赖违规. 因此，线程 1 将会被撤销并重启执行，同时，线程 1 的所有子线程将全部被撤销，同时处理单元 2 和处理单元 3 被释放，并重新分配给新激发的推测线程(2′和 3′可能是线程 2 和 3，也可能是其它的推测线程). 这些回滚线程的执行时间就构成了线程撤销和重启的时间开销. 负载不平衡开销则主要是由于线程粒度大小不一造成的. 如图 2 所示，线程 1 的粒度小于线程 0，因此在线程 0 执行完并验证线程 1 之前，处理单元 1 将处于等待状态. 这些等待时间就构成了负载不平衡开销. 此两种开销的一个共同特点就是严重依赖于线程划分策略，合理的线程划分策略能够有效地减少这两种开销所造成的影响.

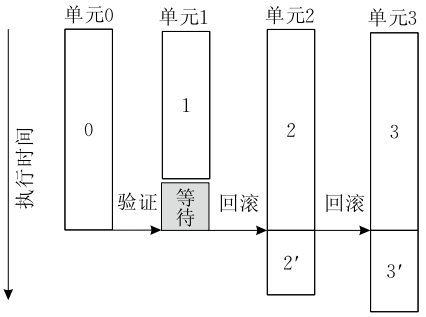


图 2 控制、数据依赖和负载不平衡开销示意图

在本文中，假设硬件体系结构是基于良好设计的，可以有效地减少前 3 种开销. 本文主要关注于如何有效定量评估这 2 种并行开销，并提出一种高效的线程划分方法来实现更加合理的线程划分.

2.3 基于预计算的值预测

在推测并行执行中，值预测技术作为一种有效减少数据依赖开销的技术得到了研究者的广泛研究. 目前，预计算片段技术作为一种有效的值预测技术已经得到了广泛的应用. 预计算片段技术是一种基于程序切片技术[11-12]的值预测方法，它通过对推测线程的 live-ins 变量构建预计算片段，并在推测线程执行前预先执行此代码片段实现推测值的预测. 构建预计算片段主要分为 3 个步骤：(1) 构建基于程序控制流图的数据依赖图；(2) 确定推测线程的 live-ins 变量；(3) 构建基于 live-ins 变量的程序片段.

在 Prophet 中，为了有效减少预计算开销，编译器只是从 CQIP 到 SP 沿推测路径前向遍历，构建推测线程的预计算片段，因此预计算片段不正确的可能原因或者是发生需要的 live-ins 位于非推测路径上，或者产生该 live-ins 的指令来自于非推测路径. 另外，当构建预计算片段过程中遇到函数调用指令时，如果将函数调用指令调用的子程序全部包含进预计算片段，这就可能导致预计算片段非常庞大，而一般情况下，子程序中很多代码可能根本是不需要的. 因此，本文采取了比较保守的方案，即在产生的预计算片段中裁剪函数调用指令.

3 基于 FCM 的线程划分框架

3.1 基本思想

SpMT 技术在允许存在大量控制和数据依赖的

情况下，试图以激进的方式挖掘线程级并行性. 研究结果表明，如果线程选择合理，并采用合适的值预测技术，推测并行能获得远远高于超标量获得的加速比，极大地提高系统性能.

在 SpMT 系统中，编译器一般是基于程序 CFG，按照某一选定的推测路径进行线程划分. 线程划分的本质就是将推测路径上的代码以基本块为单位划分成多个线程组合(即线程解). 如果将推测路径所包含的基本块看作若干个有序对象 x_i，构建一个由这些对象组成的样本集，则针对每个过程的线程划分可以看作是将此样本集按照一定的划分策略进行分类的过程，所有的分类结果构成线程划分的解集. 直观地，样本集分类的类别数目 c 的范围为[1，对象个数]. 基于 2.2 节对并行开销的分析可知，线程负载不平衡是影响加速比性能最重要的因素之一，在线程划分时如何尽可能使线程负载趋于平衡至关重要. 因此，对每个分类数 c 所对应的划分解子集，可以近似认为线程负载最平衡的划分解将具有最佳的加速比性能.

对于一个串行程序，由于将其代码划分为多个线程单元本质上可以看作是一个分类问题，因此这使引入现有的基于搜索的分类技术应用于线程划分成为可能. 同时，由于在 CMP 上将线程划分为执行时间最优的多个推测线程是 NP 完全问题[13]，对线程划分解空间的完全搜索将可能会导致无法承担的巨大开销. 因此，需要一种折中的方案来平衡搜索所引发的开销和搜索所求取的线程划分解的加速比性能，即基于搜索的线程划分方法结合一定的启发式规则进行. 通过结合启发式规则将可以大大简化需要搜索的线程划分解空间，同时也可以进一步降低搜索过程的复杂性并加速更优线程划分解的求解过程. 另外，FCM 方法作为一种有效的聚类划分技术，目前，已被大量研究. 和通常将数据进行硬性划分的一般划分方法不同，FCM 方法实现了一种软划分，其聚类结果描述了样本属于各个类别的不确定性程度. 这种聚类结果的模糊性将非常适合应用于推测多线程的划分，即通过 FCM 确定各个不同基本块隶属于不同线程单元的隶属度，然后利用基于经验得来的启发式规则来确定最终的线程划分. 正基于以上分析，本文自然地提出一种基于模糊聚类的线程划分方法. 该方法首先对分类数目 c 构成的划分解空间进行搜索，然后根据聚类结果描述的样本属于类别的隶属度，并结合启发式规则来确定最佳的划分. 显然，每个类别包含的对象所代表的基本块就自然地构成了一个推测线程.

3.2　基于模糊聚类的线程划分

3.2.1　模糊聚类算法

FCM 算法是一种基于划分的聚类算法. 给定一含有 N 个向量的样本集 $S=\{x_j,\ j=1,\cdots,N\}$，$x_j\in R^P$，算法的目标就是将样本集 S 划分到 c 个类别集$\{1,2,\cdots,c\}$，使得被划分到同一类的对象之间相似度最大，而不同类之间的相似度最小. FCM 的目标函数的一般形式为

$$J_{\mathrm{FCM}}(\boldsymbol{U},\boldsymbol{V};X)=\sum_{i=1}^{c}\sum_{j=1}^{n}u_{ij}^{m}\|x_j-x_i\|^2 \tag{1}$$

其约束条件为

$$\forall i\in\{1,2,\cdots,c\},\ j\in\{1,2,\cdots,N\} \tag{2}$$

$$u_{ij}\in[0,1],\ \sum_{i=1}^{c}u_{ij}=1,\ \sum_{j=1}^{N}u_{ij}\le N \tag{3}$$

式中，$\boldsymbol{U}$ 为模糊隶属度矩阵，$\boldsymbol{V}$ 为聚类中心矩阵，X 为样本集合，c 为聚类数，$x_j\in R^P$ 为第 j 个数据，u_{ij} 为 x_j 属于第 i 类的隶属度，$m\in[1,+\infty]$是一个加权指数. 对式(1)用拉格朗日乘法构造最小化的目标函数，可得求其最优解的必要条件：

$$u_{ij}=\frac{\|x_j-v_i\|^{-\frac{2}{m-1}}}{\sum_{k=1}^{c}\|x_j-v_k\|^{-\frac{2}{m-1}}} \tag{4}$$

$$v_i=\frac{\sum_{j=1}^{N}u_{ij}^{m}x_j}{\sum_{j=1}^{N}u_{ij}^{m}} \tag{5}$$

FCM 算法的输出是 c 个聚类中心点向量和一个 $c\times N$(N 为样本个数)的模糊划分矩阵，这个矩阵表示的是每个样本属于每个类的隶属度. 根据这个划分矩阵按照模糊集合中的最大隶属原则就能够确定每个样本归为哪个类. 本文提出的模糊聚类划分方法正是通过将线程划分问题转化为聚类划分问题，然后通过判定聚类结果的有效性来求得最佳线程划分解. 在本文提出的线程划分方法中，FCM 过程是作为划分过程的其中一个处理环节，FCM 算法处理的是每一个对象程序的结构化和归一化结果，输出结果给出的是不同程序块对不同线程单元的隶属程度描述. 对于给定的某一聚类数，其线程划分解是根据启发式规则来最终确定. 和经典的人工智能、统计学习理论以及专家系统等领域使用的模糊理论相比，其主要区别在于确定最终线程划分解的知识并非来源于聚类分析，而是来源于程序行为分析的结果；某一程序对象的 FCM 过程输出的不同程序块对不同线程单元的隶属程度知识并不能被用以和启发式规则结合，并指导其它程序对象的聚类分析过程.

3.2.2 形式化预处理

(1)结构化分析

本文提出的线程方法是基于程序 CFG 进行推测多线程划分.因此,为了便于对推测路径上的代码构建样本集,需要对程序 CFG 进行结构化预处理.首先,利用程序剖析技术提取包括分支概率、循环和过程调用的动态指令数目、循环迭代次数等信息,建立基本块层级的加权控制流图 WCFG.然后,在 WCFG 基础上,进行结构化分析并建立超级控制流图 SCFG.在 SCFG 中,过程调用和循环区域均被归结为类似于基本块的形式,即超级块.对于过程调用,其指令数目大小可采用如下表达式计算:

过程调用=调用例程+后继,

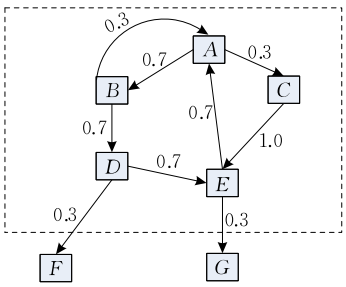
调用例程=例程路径.

对于循环区域,处理方式如图 3 例子所示.首先,通过对循环区域控制流图进行剖析,构造一个循环路径集合.每个循环路径由循环体中从循环入口节点到循环出口点的一个串行节点集组成.在循环区域控制流图中,一个循环出口节点可能有 3 种类型:(1)指向循环头节点;(2)指向循环区域外部节点;(3)节点调用了包含结束指令的过程调用.一个循环路径可以用如下式子表示:

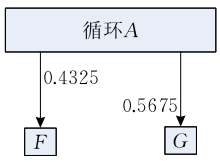
循环路径=路径上节点集+路径类型+

节点长度+分支概率,

路径类型={循环外节点 |循环头节点|退出节点}.



(a)循环控制流图



(b)循环体超级块

类型	路径	执行概率	后继
循环头节点	{A,B}	0.2100	A
循环头节点	{A,B,D,E}	0.2401	A
循环外节点	{A,B,D,E}	0.1029	G
循环外节点	{A,B,D}	0.1470	F
循环头节点	{A,C,E}	0.2100	A
循环外节点	{A,C,E}	0.0900	G

(c)循环路径

图 3 循环归结方法示意图

然后,通过计算此循环路径集合计算出循环区域的动态指令数目大小.其具体计算方式可用如下方式表示:

循环大小=循环例程的集合+后继+

路径个数+节点长度,

循环例程=循环体内路径的集合.

(2)归一化处理

进一步,为了应用 FCM 算法,首先利用结构化分析的结果来构造以基本块(或者超级块)为对象的样本集 $S=\{x_j, j=1,\cdots,N\}$,然后,根据 CFG 中的控制流关系,构造所有样本的邻接矩阵 $\boldsymbol{M}_{\text{adjacent}}$.此矩阵中的每个元素表示对应的两个样本之间的控制流关系:

$$M_{ij}=\begin{cases}1, & i\rightarrow j \text{ 有控制流}\\ 0, & i\rightarrow j \text{ 无控制流}\end{cases},\quad 1\leq i,\ j\leq N \tag{6}$$

接下来,根据 $\boldsymbol{M}_{\text{adjacent}}$ 和下述规则,求取任意样本之间的距离 D_{ij}:

① 如果 M_{ij} 值为 1,则 D_{ij} 为两个样本所代表的基本块包含的动态指令数之和的 1/2;

② 如果 M_{ij} 值为 0,则 D_{ij} 为两个样本之间所包含的路径的距离.

D_{ij} 值描述了样本之间的距离,经过归一化处理,可以得到样本集的距离矩阵 $\boldsymbol{D}_{\text{distance}}$:

$$\boldsymbol{D}_{\text{distance}}=\begin{bmatrix}D_{11} & \cdots & D_{1N}\\ \vdots & \ddots & \vdots\\ D_{1N} & \cdots & D_{NN}\end{bmatrix} \tag{7}$$

显然,根据 $\boldsymbol{D}_{\text{distance}}$ 可以很容易计算式(4)和式(5).

3.2.3 确定有效性函数

在利用 FCM 算法进行线程划分时,如何判定某一个聚类结果的有效性是一个关键问题.针对这一问题,本文提出了一个可以有效评估各种并行开销的评估模型,并利用该模型来判定聚类划分的有效性.图 4 给出了一个基于 SCFG 的线程划分的实例,图中实体粗线表示推测路径,实体细线表示线程划分的边界.图 4 中,在最可能路径上划分了 3 个线程,分别为线程 1,线程 2 和线程 3.线程 1 包含了超级块 A 和 B,线程 2 包含了超级块 C 和 E,线程 3 包含了超级块 G 和 I.根据 2.1 节描述的 Prophet 执行模型可知,这些线程将会被推测执行;当成功执行时这些线程可以对并行做出贡献,其执行时间将会等价成并行时间.由于线程是依据推测路径进行划分,推测路径上所有基本块(或者超级块)将以一定的概率参与并行,而不在推测路径上的基本块(或者超级块)则由于没有被推测(即使实际中被推测执行,由于在 Prophet 中 p-slice 只是沿推测路径进行提取,因此没有进行值预测,通常也会由于数据依赖而撤销,最终被串行执行)而只能串行执行.根据

Amdahl 定律，一个程序的加速比是由程序串行部分的运行时间和并行部分的运行时间决定的. 假设 T_Time 是串行程序执行时间，T_Seq 是程序中串行部分的执行时间，T_Par 是程序中并行部分的执行时间，那么加速比可以用如下的公式计算：

$$Speedup=\frac{T\_Time}{T\_Seq+T\_Par} \tag{8}$$

设推测路径上有 n 个节点，则可构造由这 n 个节点组成的集合 $N=\{n_t,1\leq t\leq n\}$，同时，假设 $Prob_t$ 为节点 n_t 沿推测路径的分支概率.

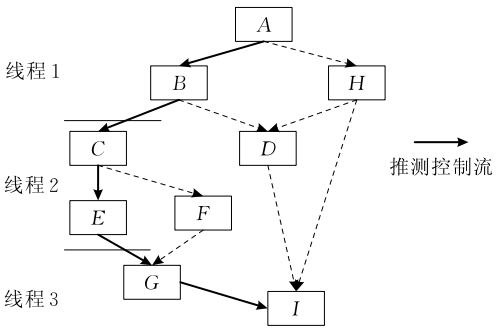


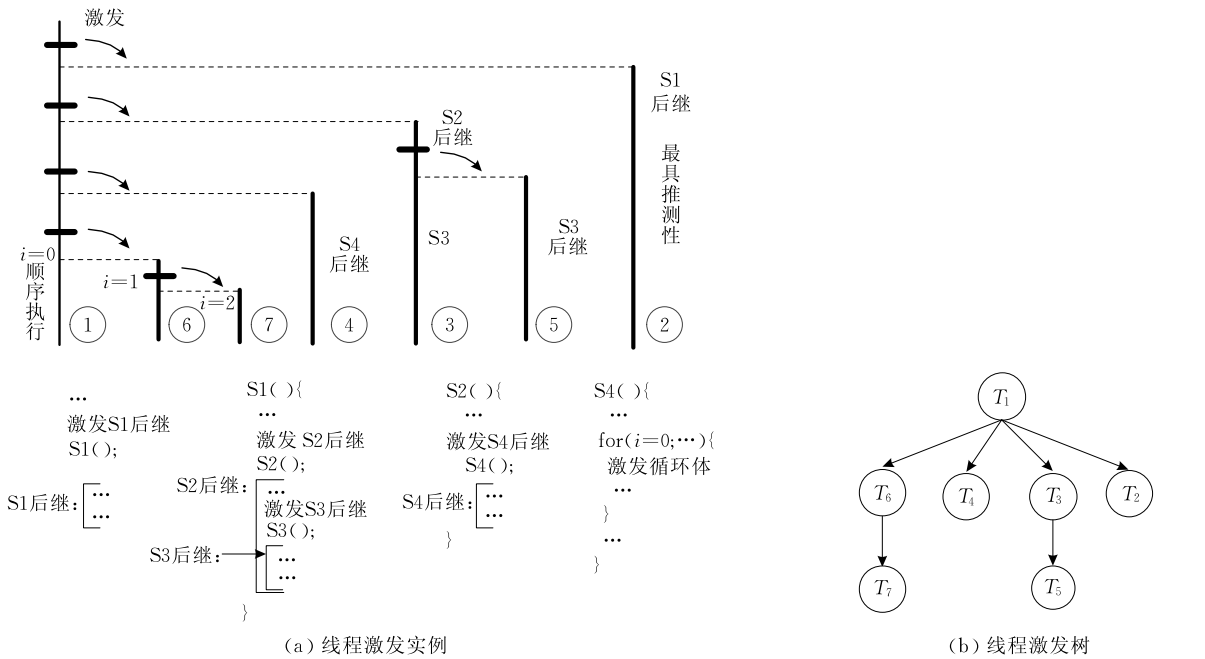
图 4　一个基于 SCFG 的线程划分实例

(1) 计算 T_Par

接下来，本文详细说明如何评估各种并行开销并最终求取某一划分的理论加速比. 对某一过程，假设共划分为 L 个线程，则对每个线程 T_l，$1\leq l\leq L$，根据集合 N 构建集合 $T_l\_path=\{n_m,1\leq t\leq M\}$，$n_m$ 为所有位于从线程 l 开始的节点到线程 l 的末尾节点路径上的节点. 则线程 l 被推测执行的概率可以由下面式子来计算：

$$P_l=\prod_{m=1}^{M}Prob_m \tag{9}$$

下面，我们通过构建并遍历线程激发树来计算程序的并行执行等价时间 T_Par. 图 5 给出了 1 个激发树构造的例子. 图 5(a)中，最左边的线程是确定线程，其它的都是推测线程. 1 个线程的推测级越高，则它越具有推测性，也越靠近右边. 线程发起的顺序为①→②→③→④→⑤→⑥→⑦，串行执行顺序为①→⑥→⑦→④→③→⑤→②. 图 5(b)中是程序对应的线程激发树，激发树中推测级别高的线程处于树的右侧.



(a) 线程激发实例　　(b) 线程激发树

图 5　构建线程激发树的例子

对于线程结点 T_l，假设 W_l 是线程 T_l 的执行时间，并且假设当推测成功时，W_l' 是此线程相对于其父线程结点的等价并行时间. 进一步假设 $T\_equ\_T_l$ 是以 T_l 为根结点的线程激发子树的等价并行执行时间.

以图 6 为例，对于线程结点 T_l，如果线程结点 T_l 是一个根结点，则 W_l' 等于 $T\_equ\_T_l$. 如果 T_l 不是根结点，假设 P_S_l 是 T_l 的预计算片段的执行时间. 对于 T_l 的父线程 T_{par}，SP_{par} 是线程的发起代价，C_p 是线程的验证代价，而 C_l 是线程的提交代价. 则对于 T_l，我们可以按照下面式子计算它的 W_l'：

$$W_l'=\begin{cases}T\_equ\_T_l, & \text{if } T_l \text{ is root}\\ (P\_S_l+T\_equ\_T_l)-(W_{par}-SP_{par}+C_p) & \\ C_l, & \text{if}(P\_S_l+T\_equ\_T_l)-(W_{par}-SP_{par}+C_p)\leq 0\end{cases} \tag{10}$$

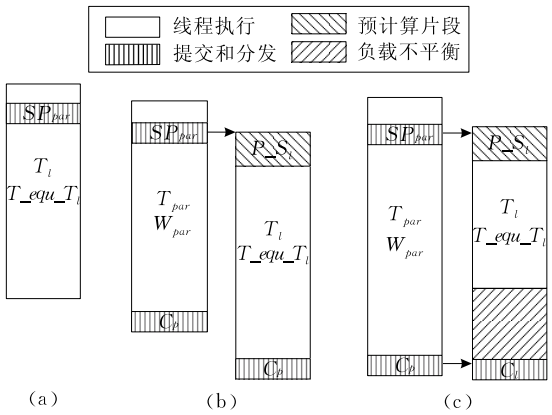


图 6 线程等价执行时间

根据式(10),我们通过广度遍历激发树来计算 $T\_equ\_T_l$. 假设它的孩子结点为 T_{li}, $i=0,1,\cdots,k$ ($k=1,2,\cdots$). 为了模拟撤销和重启代价,假设 α_l 是在 T_l 的执行过程中发生数据违规的概率. 那么我们可以用算法 1 的函数 $T\_equ(T_l)$ 来计算 $T\_equ\_T_l$. 显然 T_Par 可以通过如式(11)来确定:

$$T\_Par=T\_equ(T_{root}) \tag{11}$$

算法 1. 计算每个激发树的等价执行时间.

输入:激发树 T

输出:$T\_equ(T_l)$

1. T_equ(thread node T_l){
2. $T\_equ\_T_l=0$;
3. IF (T_l is leaf node)
4. $T\_equ\_T_l=P_lW_l'$;
5. ELSE{for each child node T_{li}
6. $T\_equ\_T_l=P_lW_l+$
7. $P_{li}\left\{\sum_{i=1}^{k}(1-\alpha_{li})\max\ T\_equ(T_{li})+\alpha_{li}T\_equ(T_{li})\right\}$;
8. return $T\_equ\_T_l$;}

在本文,我们利用程序切片技术对推测线程进行了值预测,因此,当线程 l 以概率 P_l 被推测时,我们近似认为线程将会成功执行而不会由于数据依赖发生撤销(此时 $\alpha=0$). 另外,由于我们采取了保守的预计算片段策略,对子程序和循环体等超级块没有进行值预测,因此,我们保守地认为由于复杂的数据依赖,对于包含超级块线程的推测始终是失败的(此时 $\alpha=1$).

(2)计算 T_Seq

在推测执行模式下,所有的线程 T_l 都在概率 P_l 下被推测并行执行. 设 T_Time 是程序串行的执行时间,则推测执行模式下的程序串行部分的执行时间可以通过如下的式子进行计算:

$$T\_Seq=T\_Time-\sum_{l=1}^{L}W_lP_l \tag{12}$$

根据式(9)、(11)、(12),对于任意一种线程划分,我们都可以计算出其理论加速比值. 即对某一个聚类划分 c_i,其聚类有效性函数可以由下式来定义:

$$f(c_i)=Speedup(c_i) \tag{13}$$

3.2.4 基于 FCM 的线程划分

在本节,本文给出详细的基于 FCM 算法的线程划分方法. 首先,为了进一步缩小搜索范围并减少搜索开销,基于 3.1 节的分析,我们可以摈弃一些显然会导致严重负载不平衡的线程划分,最终确定聚类数范围为[1,c],其中 c 由下式来确定:

$$c=\frac{\sum_{t=1,\cdots,N}x_t}{\max(x_t,t=1,\cdots,N)} \tag{14}$$

式(14)中,x_t 表示其代表的基本块的动态指令数目.

然后,对某一确定的 c_i,应用 FCM 算法对样本集 S 进行划分. 根据算法输出的隶属度矩阵,我们依据算法 2 来确定具体的推测线程,即确定线程边界 CQIP 点位置. 在算法 2 中,我们设定阈值 $\varepsilon=10^{-4}$,即当某一基本块对不同的线程中心隶属度相差在此阈值范围之内时,结合启发式规则,将按照更有利于减少控制依赖发生的原则来判定此基本块的归属类别.

算法 2. 确定候选推测线程.

输入:关系矩阵 $\boldsymbol{U}$,聚类中心矩阵 $\boldsymbol{V}$,阈值 ε

输出:二维矩阵 $\boldsymbol{R}[n,c_i]$,每个候选推测线程 T_{ci}

1. $Determine\_Threads(\boldsymbol{U},\boldsymbol{V},\varepsilon,\boldsymbol{R}[n,c_i])$
2. FOR(1=<j; j++; j<=n)
3. {temp=the first column number of the max($\boldsymbol{U}(j)$);
4. $\boldsymbol{R}[j,temp]=temp$;}
5. FOR (1=<j; j++; j<=n)
6. FOR (1=<k, k++, k<=c_i)
7. {IF ($\boldsymbol{R}[j,k]!=\boldsymbol{R}[j+1,k]$)&
($|\boldsymbol{U}(j,k+1)-\boldsymbol{U}(j,k)|<\varepsilon$) THEN
8. {if node j is the post-dominator of the start node in threads T_k
9. $\boldsymbol{R}[j,k]=k+1$;}}

接下来,对于已经确定的候选推测线程,分别在适当的位置插入其相应的激发点. 同时,由于线程划分的 NP 特性,即使在使用了前述一些典型启发式规则情况下,此插入过程仍然会导致很大的开销. 因此,为了进一步降低复杂性以减少开销,本文进一步引入几个典型的启发规则指导最终的线程划分,以达到更好地搜索开销和搜索精度的折中效果. 这些启发式规则主要包括:

(1)激发点 SP 可以在激发线程的任意位置;

(2)激发点 SP 和相应的 CQIP 点必须位于同一过程体或者循环体内;

(3)为了减少发生数据依赖的概率，数据依赖数目[14](用两个线程之间的数据依赖弧个数来量度)必须小于某一个阈值.

算法 3 给出了详细的激发点 SP 插入算法的描述.算法 3 中，*get_candidate_thread*()函数用来获取下一个有可能被激发的候选线程，*data_dependence_count*()则用来计算在当前激发点位置到候选线程之间的数据依赖数.

算法 3.　确定 SP 的位置.

输入：*curr_thread*, *future_thread*, *spawn_pos*

输出：*spawn_pos*

1. *insert_spawn_point*(*curr_thread*, *future_thread*, &*spawn_pos*){
2. *future_thread* := *get_candidate_thread*();
3. IF (*curr_thread.spawn_count*()==0)
4. *spawn_pos* := the position of first instruction contained in *curr_thread*;
5. WHILE(*spawn_pos* is not the last instruction) DO
6. *opt_ddc*=*data_dependence_count*(*curr_thread*, *future_thread*, *spawn_pos*);
7. IF(*opt_ddc*<DEP_THRESHOLD) THEN
8. *curr_thread*=*future_thread*;
9. *Insert_spawn_point*(*curr_thread*, *future_thread*, &*spawn_pos*);
10. END IF
11. *spawn_pos* := the position of the next instruction contained in *curr_thread*;
12. END WHILE
13. ELSE
14. *curr_thread*=*future_thread*;
15. *spawn_pos* := the position of first instruction contained in *curr_thread*;
16. *Insert_spawn_point*(*curr_thread*, *future_thread*, &*spawn_pos*);
17. END IF }

下面，利用 3.2.3 节确定的有效性函数 $f(c_i)$对每一个 c_i所对应线程划分进行有效性计算.基于 FCM 的线程划分算法描述如下：

1. 确定聚类分类数的范围$[c_{\min}, c_{\max}]$.
2. 对于聚类分类数 c 从 $c_{\min}$到 $c_{\max}$依次应用如下步骤：
2.1. 初始化聚类中心 $\boldsymbol{V}(0)$；
2.2. 应用基本的 FCM 算法更新模糊分类矩阵 $\boldsymbol{U}$ 和聚类中心 $\boldsymbol{V}$；
2.3. 判断是否收敛，如果没有，转步骤 2.2；否则转步骤 2.4；
2.4. 通过有效性指标函数计算有效性指标值 $f(c_i)$.
3. 比较对应每个 c 的有效性指标值，最大指标值 $f(c_i)$所对应的线程划分解即为所求的最优解.

3.3　算法复杂度分析

本文提出的基于 FCM 算法的线程划分方法时间复杂度主要包括 3 个方面：FCM 算法、聚类有效性函数计算和插入激发点算法.假设样本集中有 n 个节点，则插入处理激发点算法的时间复杂度不超过 $O(n)$.FCM 算法包括 2 个部分，即计算迭代计算隶属度矩阵和确定候选线程.其时间复杂度为 $O(n^2+Lnp)$，其中 L 为 FCM 的迭代次数，在本文中，由于样本集中的基本块只有一维距离属性，因此，其维数 p 的值为 1.根据 3.2.3 节的分析，聚类有效性函数计算的时间复杂度主要由式(9)和算法 1 组成，其复杂度为 $O(n\log n)$.因此，算法总的时间复杂度为 $O(n^2+Ln+n\log n)$.另外，基于启发式规则进行线程划分的算法时间复杂度通常为 $O(n\log n)$[14-15]，但是某些基于启发式规则进行动态评估进行线程划分的算法时间复杂度则通常达到 $O(n^3)$[13]，甚至更高的复杂度[16](由于需要通过模拟器模拟执行，因此将依赖于划分对象程序本身的复杂度).本文算法由于增加了 FCM 过程处理，因此时间开销将比基于传统启发式规则进行划分的算法大，但却明显小于利用动态评估进行线程划分的时间开销.

4　实验评估

4.1　模拟环境

本文提出的基于 FCM 的线程划分方法已经在 Prophet 编译系统平台上进行了实现.Prophet 模拟器采用 MIPS 指令集，通过扩展指令集实现对 SpMT 处理器模拟.每个处理单元 PE 有自己的程序计数器、取指令单元、解释指令单元和执行单元，用来从线程中取出指令并执行.同时，Prophet 模拟器包含了 ALU、Cache、流水线等部件，实现了超标量流水多核处理器的模拟.具体的配置信息见表 1.

表 1　模拟器配置参数

配置项	项目值
取指、发射和提交宽度	4 个指令周期
处理单元数	4 个
多版本一级 Cache	4 路组相联、64 KB，命中延迟：2 个时钟周期
激发开销	5 个时钟周期
验证开销	15 个时钟周期
本地寄存器	1 个时钟周期
访问内存	5 个时钟周期
提交开销	5 个时钟周期

本文选择 Olden 测试程序集的子集对论文提出

的划分方法进行性能测试. Olden 基准程序是由 Princeton 大学提供的一个测试集,具有复杂的数据结构以及对这些数据结构的操作,例如都是对树和链表进行合并、遍历等操作;另外,Olden 程序结构大多为递归,具有复杂的线程间依赖关系. 因此,Olden 基准程序测试集这些复杂的控制和数据依赖的特性,非常有利于测试本文提出的划分算法的性能. 同时,为了获取程序动态执行的信息,我们设计了一个训练集,并根据训练集的不同输入,对每个测试程序模拟执行大约 10 M 条指令.

4.2 实验及结果分析

为了展示本文方法的有效性,我们选择和基于一些典型的启发式规则的线程划分方法进行性能比较[14-16]. 这些启发式规则除本文使用的启发式规则(如 3.2.4 节描述)外,还有诸如线程划分点应尽可能位于程序的控制无关点处,人为设置线程粒度大小的范围等. 本文方法是一种结合启发式规则和搜索方法的线程划分方法,其具有的解空间搜索功能将可以有效地摒弃单纯基于启发式规则带来的自上而下或自下而上的顺序划分的局限,并进而求取更优的线程划分解. 另外,为了与基于这些启发式方法进行线程划分进行对比,我们已经在 Prophet 平台将基于这些启发式规则的划分算法进行了实现[15].

下面我们首先给出相比基于启发式规则[15]方法,本文方法在减少并行执行负载不平衡开销的信息统计,如表 2 所示.

表 2 并行执行负载不平衡开销信息统计

Olden 程序	基于启发式规则方法/%	基于 FCM 方法/%	开销减少比率/%
Mst	31.4	9.90	217.2
Bh	7.5	7.00	7.1
Power	3.1	2.50	24.0
Voronoi	7.2	4.70	53.2
Perimeter	59.0	34.30	72.0
Em3d	3.5	1.97	77.7
H. mean	18.6	10.10	84.2

表 2 中第 2 列和第 3 列分别给出的是基于两种方法生成的线程在并行执行过程中,由于负载不平衡导致的处理器处于等待状态占总处理器执行时间的比例. 从表 2 可以看出,负载不平衡所引起的开销占用相当大比例的处理器执行时间,尤其是 Mst 和 Perimeter 程序,其处理器等待状态所占比例分别达到了 31.4%和 59.0%,严重影响了并行执行的加速比性能. 同时,从第 3 列可以看出,基于本文方法生成的线程相比于基于启发式规则生成的线程所引起的负载不平衡开销均有明显的降低,特别是 Mst 程序,其负载不平衡开销减少比例高达 217.2%. 总体上,我们可以平均减少 84.2%负载不平衡开销,这说明本文方法可以有效地减少负载不平衡的影响.

表 3 给出了应用 FCM 算法时,Olden 程序聚类迭代次数的统计信息. 从表 3 中可以看出,所有程序的迭代次数都比较小,Olden 程序的平均迭代次数仅为 262.7 次. 这一方面说明,启发式规则的使用大大简化了搜索的复杂性,本文通过结合启发式规则来确定线程的解搜索空间是可行的;同时,这也说明了由于非规则程序包含的过程体通常较小[17],因此需要划分的样本集规模不大,FCM 算法可以被用来有效地进行推测多线程划分. 另外,由于 FCM 过程只是作为划分过程的其中一个处理环节,FCM 算法独立地处理的是每一个对象程序. 相较于经典的人工智能、统计学习理论以及专家系统等领域使用的模糊理论,本文方法中的 FCM 过程输出结果仅仅给出了不同程序块对不同线程单元的隶属程度描述,其聚类所得的知识并不能被用以和启发式规则结合以指导其它程序对象的聚类分析过程. 因此,表 3 同时也说明了进一步的工作应该着重于探索如何更好地结合 FCM 算法和启发式规则,并进而有效利用 FCM 聚类知识用于指导更好地进行线程划分.

表 3 Olden 程序聚类迭代次数信息统计

Olden 程序	FCM 迭代次数
Mst	212.0
Bh	476.0
Power	189.0
Voronoi	329.0
Perimeter	187.0
Em3d	183.0
H. mean	262.7

图 7 给出基于本文方法和基于启发式规则方法生成的推测线程的加速比性能对比. 从图 7 可以看到,本文方法在测试程序集上的加速比相比基于启发式规则方法均有不同程度的提高,特别是 Perimeter、Bh、Em3d、Mst 等程序提升相当明显. 同时,从图 7 可以看到,不同程序性能提升的幅度有较为明显的差距. 首先,这说明基于启发式规则方法对某些程序确实可以产生性能很好的推测线程;其次,这间接说明了本文提出的评估方法尚不够精确,导致某些潜在的开销信息不能被完全提取,致使对多种开销的评估出现一定程度的偏差,进而导致聚类有效性函数存在一定程度的不精确性. 但这种评估偏差可以通过提高评估模型的精确度来有效地消除或

者降低到可以接受的范围. 最后，这也说明根据启发式规则来确定的线程解空间存在一定程度的局限性，有可能不包含全局最优解. 另外，根据 3.3 节对算法复杂度分析可知，较启发式规则方法，在取得更好的加速比性能的同时，本文方法的不足之处在于其开销明显增大；另一方面，结合 3.3 节的分析可知，本文算法有效解决了线程划分的 NP 难题，算法开销被保持在一定的范围之内，这也说明了本文提出的通过结合使用启发式规则来寻求算法开销和最终线程划分解性能的折中方法是合理的. 同时，从图 7 还可以看出，Perimeter 程序的加速比性能较低，Em3d 程序则具有较高的加速比，这主要是是因为 Em3d 程序比 Perimeter 程序含有更多的适合推测并行的循环体.

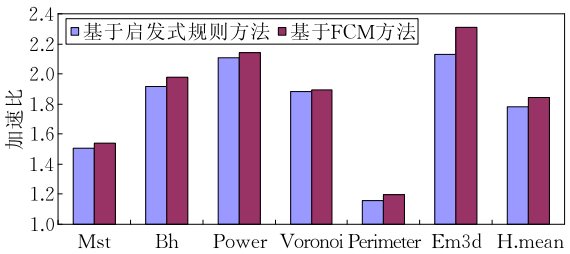


图 7　基于两种方法的加速比性能对比

为了进一步分析算法的有效性，在表 4，我们给出了基于本文方法和基于启发式规则方法生成的推测线程在编译器并行执行过程中的动态统计信息. 从表 4 可以看出，对于 Mst、Bh、Power、Em3d、Perimeter 等程序，基于 FCM 方法所激发的线程数目均有较为明显的增加；同时，虽然线程激发成功率有一定程度的下降，但是成功激发的线程数目都有较为明显的增加，特别是 Em3d 程序，激发的线程总数增加了近 2.2 倍，但激发成功率仅仅下降了 10%. 同时，从表 4 还可以看出，对于 Voronoi 程序，和其它测试程序不同，基于 FCM 方法所激发的线程数目有 0.8%的减少，但其激发成功率却有 2%的提升，且总的成功激发线程数目仍有一定程度的增加. 这说明，基于 FCM 方法产生的推测线程可以通过激发更多成功的推测线程来挖掘程序的并行性，并因此带来加速比性能提升. 进一步，定义加速比性能提升率为

$$\text{性能提升率}=\frac{\text{基于 FCM 加速比}-\text{基于启发式规则加速比}}{\text{基于启发式规则的加速比}-1}\times 100\% \quad (15)$$

表 4　基于两种方法的推测线程动态执行统计信息

Olden 程序	激发线程数目		激发成功率		成功激发线程数目		性能提升/%
	基于启发式规则方法	基于 FCM 方法	基于启发式规则方法	基于 FCM 方法	基于启发式规则方法	基于 FCM 方法	
Mst	639	651	0.7470	0.7405	477	486	6.5
Bh	120275	125321	0.9616	0.9383	115656	117588	6.6
Power	129359	136654	0.7817	0.7410	101119	101260	3.1
Voronoi	111907	110016	0.9189	0.9380	102831	103195	1.8
perimeter	9677	13619	0.6825	0.5709	6605	7775	25.3
Em3d	2360	5156	0.7466	0.6486	1762	3344	16.1

在表 4 的最后一列我们给出所有的 Olden 程序加速比性能的提升率. 从表 4 可以看出，所有的 Olden 程序都有从 1.8%到 25.3%范围内不同程度的加速比性能提升.

总之，上述结果表明，本文针对传统的基于启发式规则的线程划分方法所提出的基于 FCM 的线程划分方法可以有效地对非规则程序进行划分. 首先，FCM 方法可以在一定程度上克服单纯依靠具有经验性的启发式规则进行划分的局限，通过对线程划分解空间进行搜索将可以有效地寻求更优线程划分解. 其次，为了有效减少 FCM 过程的开销，本文方法在算法开销和获取程序加速比之间实现了一种折中方法，通过结合一些典型的启发式规则将复杂度限制在了一定的范围之内. 同时，由于本文提出的线程划分方法的划分对象是以过程为单位的非规则程序，而非规则程序的过程体大都规模较小，因此对以过程体构成的小样本集进行划分也非常适用于应用 FCM 算法. 总体上，相对于基于典型的启发式规则的划分方法，本文方法取得了平均 9.9%加速比性能提升.

5　相关工作

SpMT 技术作为一个能有效挖掘非规则程序并行性的线程级并行方法，目前已经得到了迅速发展. 在 SpMT 系统中，线程划分技术是最为关键的技术，其线程划分结果将直接决定 SpMT 编译器的最终性能. Bhowmik 等人[14]通过构建基于 SUIF-Mach SUIF 编译框架的推测多线程方案，利用控制流图和数据流图关联关系，提出一个基于相对最可

能路径的线程划分算法. Wang 等人[18]采用编译时的静态程序剖分技术提取程序的代码特征,评估循环结构的并行代价开销,寻找出最优线程边界和激发点位置,最终将整个程序分解为多个线程. 从整体上看,这些单纯基于启发式规则方法的划分算法都取得了一定的加速效果,但是由于启发式规则只能对多种并行开销进行定性评估,因而这些方法只能得到经验上较优的线程划分. 为了克服启发式规则只能定性评估多种并行开销的局限,Carlos 等人[16]提出一种通过实际动态执行模式来评估并行开销的评估模型,最终选取并行性能最佳的推测线程,实现程序的线程划分. 类似地,Luo 等人[19]通过引入硬件性能计数器,建立了一个软硬件评估模型来动态评估程序运行时的性能,并从中选取性能较好的线程作为推测线程,进而确定线程划分解. 这些基于动态评估的方法虽然取得了较好的效果,但是其低效的评估方法严重限制了算法的进一步应用.

为了克服上述划分方法的不足,一些人试图引入机器学习技术指导线程划分. Wang 等人[20]采用程序剖分技术提取循环结构特征,构建基于推测线程映射的性能调整模型,从推测线程执行的调度角度探讨如何进一步提高加速比性能. Tournavitis 等人[21]则试图通过收集运行时的信息作为训练样本,抽取控制和依赖的特征,建立一种分析模型来计算数据依赖值并进而指导线程划分. 和前述方法中仅仅应用机器学习方法间接用于提高加速比性能不同,在本文,我们通过引入模糊聚类方法,首次提出了一种基于对线程解空间进行搜索的线程划分方法. 该方法可以有效地克服传统的仅仅基于经验性的划分思想,更加深入探索线程划分影响程序加速比的内在规律. 这对于多核处理器研发、加速现有串行应用程序都将有着重要的理论意义和应用价值.

6 结论及下一步的工作

线程划分是 SpMT 技术的关键因素之一. 因此,有效评估诸如控制、数据依赖等因素导致的多种并行开销并实现最优线程划分是加速比性能提升的关键. 针对现有线程划分方法存在的不足,本文首次提出一种可以有效搜索线程划分解空间以寻求更优解的线程划分算法,即基于 FCM 的线程划分算法. 该算法首先通过深入分析各种影响加速比性能的并行开销,并结合启发式规则确定了有效的线程划分解的搜索空间. 然后,提出一种评估模型对各种并行开销进行定量评估,进而确定了聚类算法的有效性指标函数. 在此基础上,利用模糊聚类的方法对线程解空间进行搜索以求取更优的线程划分. 该线程划分方法的主要创新在于:(1)深入分析多种并行开销,并结合启发式规则有效地确立了推测线程划分解的搜索空间;(2)首次提出一种能定量评估各种并行开销的评估模型,并根据此模型求取任意线程划分解的理论加速比来判定聚类的有效性;(3)首次提出一种基于 FCM 算法的推测多线程划分方法,实现对线程划分解空间的有效搜索以求取更优的线程划分解. 基于 Olden 测试程序集的实验结果表明,本文提出的基于 FCM 方法的线程划分方法的平均加速比达到 1.85;相对于传统的基于启发式规则的线程划分方法,本文方法可以取得平均 9.9% 加速比性能提升. 这说明,本文提出的线程划分算法可以有效地对非规则程序进行推测多线程划分.

进一步地,由于本文中的线程解空间是结合启发式规则确定的一维解搜索空间,存在一定程度的局限性,有可能不包含全局最优解;同时,本文提出的评估模型尚不够精确,会导致某些潜在的并行开销信息不能被完全提取,致使对多种开销的评估出现一定程度的偏差,进而影响聚类有效性函数的精确性. 另外在划分过程中,某一程序对象的 FCM 过程输出的不同程序块对不同线程单元的隶属程度知识并不能被用以和启发式规则结合并指导其它程序对象的聚类分析过程,而这些聚类结果中蕴含的线程划分的规律性知识将对提升加速比性能有着重要意义. 因此,下一步的工作将从以下 3 个方面进行:(1)更深入分析影响线程划分的多种因素,以确定包含数据和控制依赖等因素在内的多维线程解搜索空间;(2)进一步完善评估模型,提高评估模型的精度,并设计更加精确的聚类有效性函数;(3)进一步探索如何更好地结合 FCM 算法和启发式规则,并进而有效利用 FCM 聚类知识用于指导更好地进行线程划分. 另外,由于本文提出的划分算法仅在 Prophet 编译系统平台上进行了相应的实现,因此,当将该算法与其它的并行体系结构建立的平台相结合应用时,尚需要注意以下问题:(1)针对不同的并行应用平台,分析和评估其影响并行性能的具体因素以确立相应的线程划分解的搜索空间;(2)针对不同的并行应用平台,也需要根据相应的影响并行性能的具体因素来建立相应的评估模型,并结合 FCM 算法来探索线程划分的更优解.

参 考 文 献

[1] Aliaga José I, Bollhöfer M, Martín A, et al. Exploiting thread-level parallelism in the iterative solution of sparse linear systems. Parallel Computing, 2011, 37(3): 183-202

[2] Liu W, Tuck J, Ceze L, et al. POSH: A TLS compiler that exploit program structure//Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York, USA, 2006: 158-167

[3] Pei Song-Wen, Wu Bai-Feng. SpMT WaveCache: Exploiting speculative multithreading for dataflow computer. Chinese Journal of Computers, 2009, 32(7): 1382-1392(in Chinese)
(裴颂文,吴百锋. SpMT WaveCache:开发数据流计算机中的推测多线程.计算机学报, 2009, 32(7): 1382-1392)

[4] Kulkarni M, Pingali K, Ramanarayanan G, et al. Optimistic parallelism benefits from data partitioning. ACM SIGPLAN Notices, 2008, 43(3): 233-243

[5] Madriles C, López P, Codina J M, et al. A fine-grain thread decomposition scheme for speculative multithreading// Proceedings of the 18th International Conference on Parallel Architectures and Compilation Techniques. Raleigh, USA, 2009: 15-25

[6] Ghandour W J, Akkary H, Masri W. The potential of using dynamic information flow analysis in data value prediction// Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques. Vienna, Austria, 2010: 431-422

[7] Ozturk C, Sendag R. An analysis of hard to predict branches//Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software. White Plains, USA, 2010: 213-222

[8] Chen Z, Zhao YL, Pan XY, et al. An overview of Prophet. LNCS 5574, 2009: 396-407

[9] Carlisle M C. Olden Benchmark Suite. Http://www.cs.princeton.edu/~mcc/olden.html, 2009, 06, 30

[10] Lee J, Park H, Kim H, et al. Adaptive execution techniques of parallel programs for multiprocessors. Journal of Parallel and Distributed Computing, 2010, 70(5): 467-480

[11] Renau J, Tuck J, Liu W, et al. Tasking with out-of-order spawn in TLS chip multiprocessors: microarchitecture and compilation//Proceedings of the 19th ACM International Conference on Supercomputing. Cambridge, USA, 2005: 179-188

[12] Giffhorn D, Hammer C. Precise slicing of concurrent programs: An evaluation of static slicing algorithms for concurrent Programs. Automated Software Engineering, 2009, 16(2): 197-234

[13] Johnson T A, Eigenmann R, Vijaykumar T N. Speculative thread decomposition through empirical optimization// Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. San Jose, USA, 2007: 205-214

[14] Bhowmik A, Franklin M. A general compiler framework for speculative multithreaded processors. IEEE transactions on Parallel and Distributed Systems, 2004, 15(2): 713-724

[15] Pan XY, Zhao, YL, Chen Z, et al. A thread partitioning method for speculative multithreading//Proceedings of the 8th International Conference on Embedded Computing. Dalian, China, 2009: 285-290

[16] Carlos M, Carlos G, Jesu's S, et al. Mitosis: A speculative multithreaded processor based on precomputation slices. IEEE Transactions on Parallel and Distributed Systems, 2008, 19(7): 914-925

[17] Sherwood T, Perelman E, Calder B. Basic block distribution analysis to find periodic behavior and simulation points in applications//Proceedings of the 10th International Conference on Parallel Architectures and Compilation Techniques. Barcelona, Spain, 2001: 3-14

[18] Wang S, Dai X, Yellajyosula K S, et al. Loop selection for thread-level speculation//Proceedings of the 18th International Workshop on Languages and Compilers for Parallel Computing. New Orleans, USA, 2006: 289-303

[19] Luo Y, Packirisamy V, Hsu W C, et al. A dynamic performance tuning for speculative threads//Proceedings of the 36th ACM/IEEE Annual International Symposium on Computer Architecture. Austin, USA, 2009: 462-473

[20] Wang Z, O'Boyle M F P. Mapping parallelism to multicores: A machine learning based approach//Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. Raleigh, USA, 2009: 75-84

[21] Tournavitis G, Wang Z, Franke B, et al. Towards a holistic approach to auto-parallelization integrating profile-driven parallelism detection and machine-learning based mapping// Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation. Dublin, Ireland, 2009: 177-187

LI Yuan-Cheng, born in 1981, Ph. D. candidate. His research interests include high performance computer architecture, parallel computing and machine learning.

YIN Pei-Pei, born in 1985, M. S. candidate. Her research interests focus on parallel computing.

ZHAO Yin-Liang, born in 1960, Ph. D., professor. His research interests include design theory and implementation of programming languages, parallel computing and artificial intelligence.

Background

Chip multiprocessors (CMPs), or multi-core processors, have become a common way of reducing chip complexity and power consumption while maintaining high performance. However, maximizing the utilization of the computing resources provided by multi-core processors requires adjustments both to the operating system (OS) support and to existing application software. Programming truly multithreaded code often requires complex co-ordination of threads and can easily introduce subtle and difficult-to-find bugs due to the interleaving of processing on data shared between threads (thread-safety). Consequently, such code is much more difficult to debug than single-threaded code when it breaks. Speculative multithreading (SpMT) technology is an effective mechanism for automatic parallelization of irregular programs. Compared to manual methods such as parallel programming, it is hopeful to accelerate the large serial applications with zero cost and can also be combined with the multi-core parallel programming to further improve the utilization of core resources.

While speculative parallelization can potentially deliver significant speedup for programs that would otherwise be executed sequentially, several overheads associated with the technique limit these speedups in practice. In SpMT system, thread partitioning is a critical factor in determining the performance achieved by a speculatively threaded program, and the thread partitioning method used by the compiler is key issue to the success of the speculative approach. Many thread partitioning algorithms have been proposed for SpMT systems which support some sorts of speculative thread execution. Overall, these algorithms have achieved certain acceleration. However, most existing thread partitioning methods are still based on simple heuristics rules to select speculative threads. Because these heuristics rules can only be used to indirectly estimate the speculative multithreaded execution overheads and simply tackle individual overheads, these methods can only get the empirical optimal speculative thread solution. For the limitation of these existing methods, in this paper, we first propose a fuzzy c-means (FCM) algorithm based thread partitioning method which can be used to effectively search the solution space of speculative threads and get the optimal solution. In this method, through combining the heuristic rules, we effectively compress the solution space of speculative threads based on the analysis of several overheads. Meanwhile, in order to apply the FCM algorithm, we design and implement the cluster validity function by introducing a cost estimation model. The experimental results show that the proposed method can effectively search the solution space of speculative threads and we can indeed get better performance.

This work is supported by the National High Technology Research and Development Program (863 Program) of China under Grant No. 2008AA01Z136, and the National Natural Science Foundation of China under Grant No. 61173040. The research activities were accomplished at the cooperative of the all members of Prophet Compiler team.