软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(8):1950−1964 [doi: 10.3724/SP.J.1001.2012.04148]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种基于路径优化的推测多线程划分算法*

李远成, 赵银亮+, 李美蓉, 杜延宁

(西安交通大学 计算机科学与技术系,陕西 西安 710049)

Thread Partitioning Algorithm for Speculative Multithreading Based on Path Optimization

LI Yuan-Cheng, ZHAO Yin-Liang+, LI Mei-Rong, DU Yan-Ning

(Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China)

+ Corresponding author: E-mail: zhaoy@mail.xjtu.edu.cn

Li YC, Zhao YL, Li MR, Du YN. Thread partitioning algorithm for speculative multithreading based on path optimization. *Journal of Software*, 2012,23(8):1950−1964 (in Chinese). http://www.jos.org.cn/1000-9825/4148.htm

Abstract: Speculative multithreading (SpMT) technology is an effective mechanism for automatic parallelization of irregular programs. However, just generating speculative threads based on the control flow graph which only contains branch probability information is not enough. It is inevitable that there may be excessive constraints resulting from control dependence and data dependence in practice. In the traditional thread partitioning algorithms, a key problem is that these existing evaluation methods do not integrate the data dependence with the control speculation in order to obtain the maximum benefit. In order to solve the inefficiency of evaluation for control dependence and data dependence, this paper proposes a path optimization based thread partitioning algorithm. In this algorithm, by introducing pre-computation technique and discussing the trade-offs between different speculative paths the study establishes an evaluation model to evaluate the control dependence and data dependence. Meanwhile, in order to reduce the side-effect of workload imbalance, some heuristics rules are designed. The experimental results show that there are interesting trade-offs between different speculative paths and it is possible to achieve a better performance. On average, the study achieved an average speedup of 1.83 on Olden benchmark suits.

Key words: speculative multithreading; thread-level parallelism; thread partitioning; path optimization; automatic parallelization

摘 要: 推测多线程(speculative multithreading,简称 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,基于控制流图和分支预测技术的线程划分方法,不可避免地会受到划分路径上所存在的控制依赖和数据依赖的制约.目前,在传统的线程划分算法中存在的一个重要问题是,在对划分路径进行选取时只考虑了控制依赖影响却不能有效地综合考虑数据依赖的影响,进而导致不能选取最佳的划分路径.因此,针对传统方法中这种依赖评估方法效率低下的问题,设计并实现了一种基于路径优化的线程划分算法.该算法通过引入基于程序切片技术的预计算方法,建立一种路径评估方法来评估程序间的控制和数据依赖.同时,引入控制线程体大小的启发式规则,以便有效地

* 基金项目: 国家自然科学基金(61173040); 国家高技术研究发展计划(863)(2008AA01Z136)
收稿时间: 2011-04-20; 修改时间: 2011-07-21, 2011-09-02; 定稿时间: 2011-11-02

解决负载不平衡的问题.基于 Olden 测试集的测试结果表明,所提出的算法可以有效地对非规则程序进行划分,其平均加速比可以达到 1.83.

关键词: 推测多线程;线程级并行;线程划分;路径优化;自动并行化

中图法分类号: TP314 **文献标识码**: A

挖掘并行性是提高串行程序执行性能的有效方法之一.随着超标量和指令级并行(instruction-level parallelism,简称 ILP)等技术遇到越来越多的瓶颈以及片上多处理器(chip multi-processor,简称 CMP)的迅速发展,线程级并行(thread level parallelism,简称 TLP)逐渐成为一个更佳的选择.目前,包括诸如科学计算、数字信号处理等应用领域,以数据处理为核心的规则程序已经得到了很好的并行处理;但是,对于如桌面应用、多媒体及服务器应用等含有大量控制和数据依赖的非规则程序的并行处理,仍是一个非常棘手的问题.

近年来,推测多线程(speculative multithreading,简称 SpMT)技术[1−3]作为一项能够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 技术在允许存在大量控制和数据依赖的情况下,以激进的方式发掘线程级并行性.执行过程中,每一个推测线程执行程序的不同部分,并在运行时由执行模型检测控制和数据等依赖的发生.如果出现依赖违规,采取撤销并重新运行等硬件手段来保证程序执行的正确性.在 SpMT 系统中,线程划分方法是最为关键的因素,其线程划分结果将直接决定 SpMT 编译器的最终性能.Wang 等人[4]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.Johnson 等人[5]从图论的角度提出平衡最小割算法,采用基于编译器和基于程序剖分的手段动态调整依赖边的权值,搜索最优的候选线程,最终将整个程序分解为多个线程.在此基础上,Johnson 等人[6]又提出了能够解决敏感程序输入以及适应程序运行时阶段特征造成性能变化的方法,实现程序的动态划分.Luo 等人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[7].文献[8]则主要针对程序的子程序结构进行了深入的研究,以较大的粒度挖掘线程级推测并行性.另外,与前述基于启发式规则的线程划分策略相比,文献[9]首先将机器学习用于线程划分.通过利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,以较小的编译时搜索开销来实现推测线程的划分.

在 SpMT 系统中,编译器一般利用程序剖析技术来分析程序的行为并构造程序的加权控制流图(weighted control flow graph,简称 WCFG),然后,基于该 WCFG 来进行线程的划分.同时,利用诸如同步和值预测等机制进一步消减线程间的数据依赖造成的影响.在基于 WCFG 的线程划分时,由于沿不同的划分路径会引起不同的控制和数据依赖,而控制和数据依赖又是制约加速比性能的重要因素,因此,如何权衡不同的路径带来的开销并最终选取最优的划分路径进行划分显得至关重要.基于路径优化技术进行线程划分已经得到了普遍的应用[10,11],目前主要有两种路径选择策略:一种是贪婪选取分支概率最大路径,即最可能路径[12].这种策略只是贪婪地选取了局部控制依赖代价最小的路径,实现简单.但是由于追求了局部代价最小,同时又由于忽略了数据依赖带来的开销影响,将难以确立全局代价最小的路径.在比较极端的情形下,程序很难取得明显的加速性能.另一种是建立路径评估模型,选取代价最小的路径[13,14].由于综合考虑了控制和数据依赖等综合因素的影响,这种策略具有更大潜力.在文献[13]中,Carlos 等人首先依据不同的路径进行划分,并确立一个候选线程集,然后,通过建立一个基于实际动态执行模式的评估模型来评估多个不同候选线程,并最终选取并行性能最佳的推测线程,实现程序的线程划分.然而,基于动态执行进行评估的模式虽然可以提供更加精确的运行时信息以指导更好的划分,但由于线程划分是一个 NP 完全问题[15],对于线程解空间的复杂搜索将会是一个漫长的过程.因此,该方法将不可避免地受到过多时间开销的严重限制.文献[14]则通过分析程序间不同路径的数据依赖数进行数据依赖影响评估,提出一种基于静态信息评估的路径优化方法.然而,此方法只是利用不同路径上数据依赖数目来简单评估数据依赖的影响.但是,数据依赖的数量并不必然和由依赖导致的开销成正比,因此不可避免地会面临评估精确性不足的问题.

为了解决上述评估方法效率低下的问题,本文提出了一种基于路径优化的线程划分算法.该算法通过引入

基于程序切片技术的预计算方法,首先提出一种有效的路径评估方法来评估程序间控制和数据依赖;其次,在线程划分过程中引入启发式规则,进一步有效地解决线程负载不平衡的影响;最后,本文将算法在课题组开发的Prophet编译系统平台[16]上进行了实现.基于Olden测试程序集[17]的实验结果表明,本文提出的基于优化路径的线程划分算法可以有效地提升程序加速比性能,其平均加速比性能达到了1.83.

1 推测多线程

1.1 推测多线程执行模型

在推测多线程模式下,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也即确定线程,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;而其他线程则均为推测线程,线程间以前驱和后继的形式保持串行程序的语义.当程序执行到激发指令时,如果现有资源允许激发,则将激发一个新的推测线程.每个线程均可以激发多个线程,且严格遵循乱序激发模式[18].当确定线程执行结束时,将验证其直接后继线程.若验证正确,则确定线程向安全存储器(通常为内存或者共享的高级缓存区)提交其执行结果,然后将确定执行的权限传递给其直接后继线程;若验证失败,则撤销所有的推测子线程,并重新执行其直接后继线程.推测线程产生的数据则存储于推测缓冲区,当遇到此缓冲区溢出或线程执行完毕时,此线程将进入等待状态,直到其被验证正确得到确定执行权限或者验证失败并被重启.

在Prophet编译系统中,引入SP(spawn point)和CQIP(control quasi-independent point)指令对来唯一确定一个激发线程对.同时,引入预计算片段(pre-computation slice,简称 p-slice)来进行线程的值预测.p-slice 是由编译器根据程序切片技术沿推测路径生成的一小段代码,用来预测推测线程使用的 live-ins(指线程体使用但并非由该线程定义的值).串行程序代码片段及其预计算片段一起构成推测线程.如图1所示,当程序执行到SP时,如果现有资源允许激发,则将激发一个新的推测线程.当确定线程(线程0)执行到CQIP时,将验证其直接后继线程(线程1)在p-slice中产生的live-ins数据.若验证正确,则确定线程提交其执行结果,然后将确定执行的权限传递给其直接后继线程;若验证失败,则撤销此所有推测子线程,然后跳过p-slice片段,确定执行此直接后继线程.

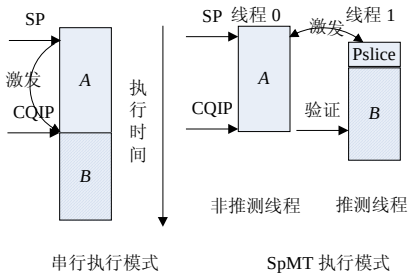


Fig.1 Sequential vs. SpMT execution pattern

图1 串行执行模式和SpMT执行模式

1.2 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.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[19].

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,此开销主要与缓冲区物理大小、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是指由于发生控制和数据依赖推测失败引起推测线程回滚并重启等操作引起的时间开销.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 PE2 和 PE3 被释放,并重新分配给新激发的推测线程(线程 2′和线程 3′可能是线程 2 和线程 3,也可能是其他推测线程).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指由于线程粒度大小不同,推测线程在执行结束后等待成为确定性线程进而提交所引起的时间开销.如图 2 所示,线程 1 的粒度小于线程 0,因此在线程 0 执行结束并验证线程 1 及将确定状态交给线程 1 之前,处理单元 PE1 将处于等待状态.图示中的等待时间就构成了负载不平衡开销.这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可以有效地减少这两种开销所造成的影响.

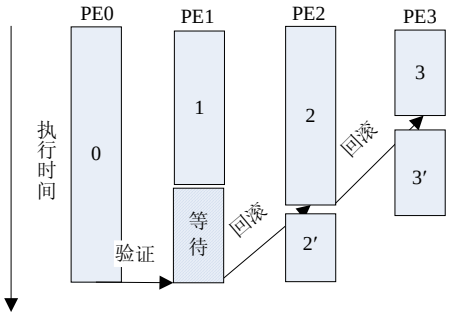


Fig.2 Speculative parallelization with data and control dependence, load imbalance overheads

图 2 推测并行中的控制、数据依赖和负载不平衡等开销

本文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地减少后两种主要开销,并通过提出一种路径评估方法来优化推测路径,最终提出一种基于路径优化的线程划分算法.

2 线程划分算法

2.1 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,试图以激进的方式发掘线程级并行性.值预测技术的发展又进一步打破了这种依赖关系束缚,即便明确存在依赖,如果能准确地预测后继相关线程所需的输入值,则可以大大减少由于出现依赖违规造成的撤销和重启开销,更大地加速串行程序的执行速度.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,则推测并行能够获得远远高于超标量获得的加速比,这将极大地提高系统的性能.

在 SpMT 系统中,由于编译器一般是基于程序的 WCFG 来进行线程划分,而不同的划分路径会导致不同的控制和数据依赖,因此,如何权衡不同的路径带来的开销并最终选取最优的划分路径进行划分显得至关重要.在图 3 中,我们给出一个程序片段的 CFG,图中大写字母标示的方框代表基本块,P_{AB} 和 P_{AC} 则分别表示路径的分支概率,黑色直线表示可能的线程边界.下面以图 3 为例来说明基于路径优化进行线程划分的思想.我们分两种情形进行讨论.

当推测执行路径 *A*-*B* 时,在概率 P_{AB} 下,*A*-*B* 路径上的代码将被成功并行执行;在概率 P_{AC} 下,由于预测值是针对 *A*-*B* 路径上代码进行的,因此程序将推测失败,*A*-*C* 路径上代码将只能被串行执行.类似地,当推测执行路径 *A*-*C* 时,在概率 P_{AB} 下,*A*-*B* 路径上代码被串行执行;而在概率 P_{AC} 下,*A*-*C* 路径上的代码将被成功并行执行.设 T_{AB} 和 T_{AC} 分别是在推测路径 *A*-*B* 和 *A*-*C* 下该段程序代码的执行时间,则若能量化 T_{AB} 和 T_{AC},即可确定最优的推测路径.

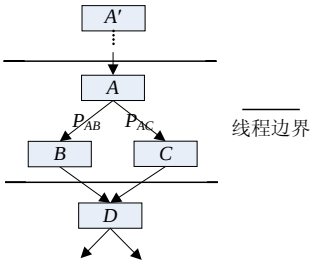


Fig.3 An example of speculative thread partitioning

图 3 一个推测线程划分的例子

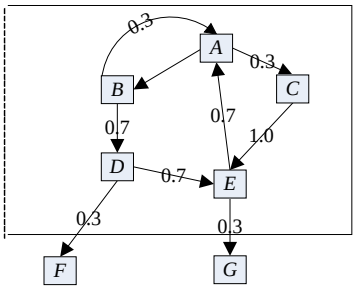
2.2 结构化预处理

为了易于实现对于不同路径的时间开销评估,本文对程序的控制流图 CFG 做进一步的结构化处理.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.和 CFG 不同的是,在 SCFG 中将不存在循环回溯边,其中,循环区域和过程调用均被归结为类似于基本块的形式,即超级块.超级控制流图 $SCFG=\langle CN,CE,Cins,Cfreq\rangle$是一个有向图,$CN$ 为超级块集,$SCFG$ 中的一个节点代表一个基本块或者一个超级块;边集 $CE=\{\langle cn_i,cn_j\rangle|cn_j$ 是 cn_i 的后继超级块$\}$,对应超级块的入口边和出口边由归结区域的入口边和出口边代替.以节点 cn 为起点的边个数称为 cn 的出度;$Cins$ 为节点的权值函数,$Cins(n_i)=\{cn_i$ 中包含对应超级块的动态指令数$\}$;$Cfreq$ 为边的分支概率,其值可以通过动态运行来确定,即 $Cfreq(C_{ij})=\{$动态执行中经过 C_{ij} 的次数/总的动态执行次数$\}$.

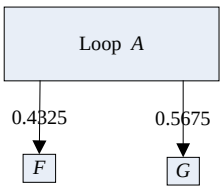
对于过程调用,其指令数目大小可采用如下表达式计算:

- CALL=CALL INSTANCE+POINTER;
- CALL INSTANCE=INSTANCE PATH.

对于循环区域,处理方式如图 4 的例子所示.



(a) 循环控制流图



(b) 循环体超级块

Type	Path	Prob	Next
Continue	{*A*,*B*}	0.2100	*A*
Continue	{*A*,*B*,*D*,*E*}	0.2401	*A*
Break	{*A*,*B*,*D*,*E*}	0.1029	*G*
Break	{*A*,*B*,*D*}	0.1470	*F*
Continue	{*A*,*C*,*E*}	0.2100	*A*
Break	{*A*,*C*,*E*}	0.0900	*G*

(c) 循环路径

Fig.4 Construction method of loop macronode

图 4 循环归结构建方法

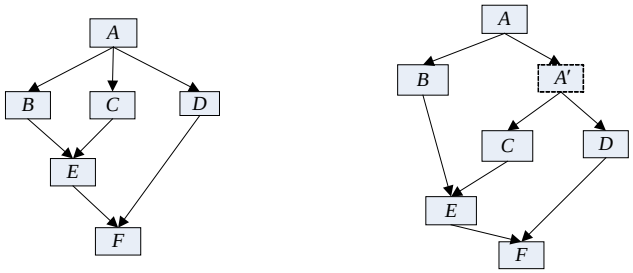
首先,通过对循环区域控制流图进行剖析,构造一个循环路径集合.每条循环路径由循环体中从循环入口节点到循环出口点的一个串行节点集组成.在循环区域控制流图中,一个循环出口节点可能有 3 种情形:一是指向循环头节点(continue path),二是指向循环区域外部节点(break path),三是节点调用了包含结束指令的过程调用(exit path).一条循环路径可以用如下式子表示:

- LOOP PATH=LIST OF {NODE}+TYPE+LENGTH+PROB;
- LOOP PATH TYPE={CONTINUE|BREAK|EXIT}.

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,其具体计算方式用如下方式表示:

- LOOP=CIRCULAR LIST OF {LOOP INSTANCE}+POINTER+TRIP COUNT+LENGTH;
- LOOP INSTANCE=LIST OF {LOOP PATH}.

另外,为了进一步简化路径评估的复杂性,对 SCFG 进行等价转换,通过插入虚拟基本块的形式,构建其相应的等价超级控制流图 E-SCFG.与 SCFG 相比,E-SCFG 的不同之处在于,其所有的分支路径都不超过 2 个,即 CG 中每个节点的出度不超过 2.其转换方式如图 5 所示.在图 5(a)中,基本块 *A* 有 3 条分支跳转,因为将其转换成两路分支,我们可以插入一个虚拟的基本块 *A′*,如图 5(b)所示,即可将基本块 *A* 的 3 个分支路径转换成两路,其中,虚拟基本块 *A′*包含的指令数目设置为 0 条.在图 5(b)中,*A′C* 和 *A′D* 分支的相应分支概率即为图 5(a)中 *AC* 和 *AD* 分支的分支概率.



(a) 超级控制流图 (b) 等价的超级控制流图

Fig.5 Equivalent transformation of SCFG

图 5 等价超级控制流图转换

2.3 依赖评估

在 SpMT 系统中,编译器一般是基于程序控制流图进行划分的;同时,利用诸如同步和值预测等机制进一步消减线程间的数据依赖造成的影响.在基于 WCFG 的线程划分时,由于沿着不同的划分路径会引起不同的控制和数据依赖,因此,如何权衡不同路径带来的这些依赖开销并最终选取最优的划分路径进行划分显得至关重要.

2.3.1 值预测技术

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认可.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[20]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来实现推测值的预测.构建预计算片段主要分为以下 3 个步骤:

(1) 构建基于程序控制流图的数据依赖图.并发线程之间的数据依赖冲突主要指 RAW(read-after-write)冲突,为了解决线程之间的 RAW 问题,就需要检测线程数据的定值和引用关系,构造引用-定值链.这些信息将被提供给线程划分器,用来指导线程划分,以避免或尽量减少并发线程之间的 RAW 冲突.引用-定值链或者 ud(use-defined)链是关于变量数据流信息的稀疏表示.一个变量的 ud 链是一个列表,假设在程序中某点 *u* 引用了变量 *a* 的值,则将能够到达该引用的 *a* 的所有定值点的全体称为 *a* 在引用点 *u* 的引用-定值链,简称 ud 链.也就是说,ud 链连接一个变量的引用到所有可能流经到该引用的定值.

由于线程划分以基本块为单位,因此,数据依赖分析转化为对基本块之间的数据依赖进行分析.为了评估基

本块之间的数据依赖关系,将这里的线程之间的引用-定值链转化为依赖弧.依赖弧定义为一条有向弧,每一条依赖弧连接一个变量的引用-定值位置偶对.从程序段 T_1 流入程序段 T_2 的依赖弧表示 T_1 中引用(读)了 T_2 中定值(写)的某个变量,说明两个程序段之间发生读写依赖.依赖弧数目的多少则反映了两个程序段之间数据依赖大小,因此,两个程序段之间的数据依赖程度问题就可以用二者之间的依赖弧数目来进行评估.

(2) 确定推测线程的 live-in 变量.在循环域中寻找 live-ins 变量,主要由决定下一次迭代过程体是否执行的那些活跃变量组成.在循环体的出口与入口之间有一条回边,只需分析循环体的出口基本块中经过回边这条路径时所要用的变量即可.在非循环域中寻找 live-ins 变量,需要对将来可能会成为线程体的一个连通子图进行分析,从该连通子图的第 1 个超级块开始,采用数据流分析得出每个超级块的变量 *Def* 集合和 *Use* 集合.然后在该连通子图的最可能路径上迭代每一个超级块,直到遇到该连通子图出口为止.迭代方程如下:

$$Def(SG)=Def(CB_i)\cup Def(SG) \tag{1}$$

$$Use(SG)=Use(SG)\cup(\forall u(u\in Use(SB_i),u\notin Def(SG))) \tag{2}$$

公式(1)、公式(2)中:

- SG——可能划分线程的连通子图;
- CB_i——连通子图 SG 中可能路径上的每个超级块.

初始化时,$Def(SG)$和 $Use(SG)$为空;迭代收敛后,$Use(SG)$集合即为 live-ins 变量集合.

(3) 构建基于 live-ins 的程序切片.输入推测线程的 live-ins 向量集,在控制流图中从 CQIP 到 SP 沿推测路径前向遍历,构建推测线程的预计算片段.为了有效减少预计算开销,编译器只是沿推测路径进行预计算片段构造,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.

2.3.2 线程划分策略

推测编译器试图将程序划分为多个推测线程并行执行来挖掘程序的并行性.线程划分的最终目的是综合程序间控制和数据依赖等信息,将程序划分为最佳的线程组合,以取得最优的执行时间.然而,在多线程结构上将程序划分为执行时间最优的多个推测线程却是 NP 完全问题.因此,线程划分一般依据启发式规则进行.在线程划分时,为了减少程序间控制和数据依赖的影响,通常遵循下述启发规则.

启发规则.在线程划分时,为了尽可能地减少控制依赖,线程划分均在程序控制流图的控制无关点进行划分;在依据此规则划分时,对于过大从而引起严重负载不平衡的线程,将依据此规则将此线程进一步划分成多个线程;同时,采取值预测技术预测推测线程使用的数据值,以减少数据依赖的影响.

图 3 给出了一个在此启发式规则指导下进行划分的一个例子.在图 3 中,节点 D 是 A 的准控制无关点,因此,在 D 节点前进行划分,作为以 A 为起始块的线程边界.基于此启发式规则的线程划分,可以有效地减少推测执行过程中控制依赖的发生,因而减少由于控制依赖违规而导致的开销.然而,简单的启发式规则只能定性分析控制依赖开销并以此来指导线程划分[21],并不能有效地对不同路径进行评估并进而确立最优的推测路径.因此,为了更精确地进行定量分析数据和控制等依赖的开销,并更精确地指导线程划分,本文首先根据当前基于启发规则的划分算法[1,13]给出两个重要的发现.这两个发现将作为下一步进行定量依赖评估的重要前提.

- 发现 1:在线程划分时,对非循环区域均是顺序激发线程,仅当遇到函数调用和循环区域时才进行乱序激发线程.
- 发现 2:在线程划分时,对某一个推测线程 T 来说,在 E-SCFG 中,其激发线程都是以此线程 T 起始块的前向准控制无关点起始的线程.

2.3.3 依赖评估方法

根据第 2.3.1 节的分析可知,预计算片段的开销正是数据依赖所引起的额外开销.因此,当数据依赖模型足够精确时,此预计算片段开销天然地可以用来作为此推测线程由于数据依赖所导致的时间开销.基于第 2.3.2 节

的两个发现,本文提出一种基于代价评估的路径优化方法.下面以图 3 为例,我们详细分析如何对不同的路径进行依赖开销评估并确定最优的划分路径.

为了评估控制依赖的因素,引入路径的分支概率,假设分别为 P_{AB} 和 P_{AC}.同时,假设 $I(AB)$和 $I(AC)$分别表示路径 *A-B* 和路径 *A-C* 的动态指令数目,每条指令的指令周期设为 *T*.对于数据依赖的因素,基于以上讨论,自然地我们可以利用预计算片段来评估数据依赖的开销.根据第 2.3.2 节的两个重要发现,对于以块 *A* 到其准控制无关点 *D* 之间的某一条路径来说,其预计算片段可以近似等价为此路径上的 live-in 变量在 *A* 到 *A′*(在图 3 中,假设 *A′*是 *A* 的前向准控制无关点)之间程序段的程序切片.对于路径 *A-B* 和路径 *A-C*,分别假设其相应的预计算片段为 PS_{AB} 和 PS_{AC}.另外,由于假设硬件体系结构是基于良好设计的,因此可以理想地认为,在相同的软硬件环境下,当控制推测成功且没有数据依赖发生时,程序段可以获得相同的加速比性能,假设此加速比为 *Sp*.接下来,根据第 2.1 节的分析,分别计算 T_{AB} 和 T_{AC}.

(1) 推测划分路径 *A-B*.此时,路径 *A-B* 将会被划分并被推测并行执行,同时将基于 *A-B* 路径上的 live-in 变量生成的预计算片段插入到 *A-B* 路径的开始处.考虑实际运行时发生的两种情况,当在分支概率 P_{AB} 下沿该路径运行时,路径 *A-B* 成功被推测执行;当在分支概率 P_{AC} 下,由于 *A-C* 路径没有被推测划分并进行值预测,因此,该路径上的程序代码将只能串行执行(即使被激发,也将由于发生数据依赖违规而被撤销,最终被串行执行).于是,该段代码等价执行时间 T_{AB} 可以计算为

$$T_{AB}=T\left\{I(AC)P_{AC}+\frac{P_{AB}(I(AB)+PS_{AB})}{Sp}\right\} \tag{3}$$

(2) 推测划分路径 *A-C*.此时,路径 *A-C* 将会被划分并被推测并行执行,同时将基于 *A-C* 路径的 live-in 变量生成预计算片段插入到 *A-C* 路径的开始处.同样考虑实际运行时发生的两种情况:当在分支概率 P_{AC} 下沿该路径运行时,路径 *A-C* 成功被推测执行;当在分支概率 P_{AB} 下沿该路径运行时,由于 *A-B* 路径没有被推测划分并进行值预测,因此,该路径上的程序代码将串行执行.于是,该段代码等价执行时间 T_{AC} 可以计算为

$$T_{AC}=T\left\{I(AB)P_{AB}+\frac{P_{AC}(I(AC)+PS_{AC})}{Sp}\right\} \tag{4}$$

进一步地,假设

$$f()=T_{AC}-T_{AB} \tag{5}$$

则由公式(3)~公式(5)可得:

$$f()=T\left\{E_1\left(1-\frac{1}{Sp}\right)+\frac{1}{Sp}E_2\right\} \tag{6}$$

其中,

$$E_1=I(AB)P_{AB}-I(AC)P_{AC} \tag{7}$$

$$E_2=PS_{AC}P_{AC}-PS_{AB}P_{AB} \tag{8}$$

接下来,根据公式(5)~公式(8),分 3 种情况讨论 $f()$的值的正负:

(1) 当 $E_1E_2\geqslant 0$,且 $E_1+E_2\geqslant 0$ 时,有 $f()=T_{AC}-T_{AB}\geqslant 0$,即 $T_{AC}\geqslant T_{AB}$.

(2) 当 $E_1E_2\geqslant 0$,且 $E_1+E_2<0$ 时,有 $f()=T_{AC}-T_{AB}<0$,即 $T_{AC}<T_{AB}$.

(3) 当 $E_1E_2<0$ 时,由于 *Sp* 真实值未知,因此,$f()$值的正负无法精确判定.也就是说,在这种情况下无法精确判断 T_{AB} 和 T_{AC} 的大小.

由于 T_{AB} 和 T_{AC} 表示的是推测 *A-B* 和 *A-C* 路径的相应执行时间,因此根据上述分析,对如图 3 所示的 E-SCFG 中的一个节点 *A* 到其准控制无关点 *D* 之间的多条路径,可以得出如下优化规则:

(1) 若 $E_1E_2\geqslant 0$,且 $E_1+E_2\geqslant 0$,则选取路径 *A-B*.

(2) 若 $E_1E_2\geqslant 0$,且 $E_1+E_2<0$,则选取路径 *A-C*.

(3) 若 $E_1E_2<0$,则分为两种情况:当 $P_{AB}>P_{AC}$ 时,选取路径 *A-B*;当 $P_{AB}<P_{AC}$ 时,选取路径 *A-C*.

从上述优化规则(3)可以看出,当无法精确判断时,优化方法采取了保守的做法,直接选取分支概率较大的路

径分支.另外,根据第 2.2 节的描述可知,E-SCFG 已经进行了二路转化,因此,此优化规则可以直接应用于本文提出的路径优化.此规则实现如图 6 中的算法 1-3 所示.

```
算法 1-1. Partition_procedure(procedure P).
    For all loop L in p do
    Partition_loop(L);
    End for
    start_block:=P 的入口节点;
        end_block:=P 的出口节点;
        curr_thread:=create_new_thread(start_block,null,null);
        curr_thread:=partition_thread(start_block,end_block,curr_thread);
算法 1-2. partition_loop(loop L).{
    start_block:=entry block of loop L;
    end_block:=exit block of loop L;
    optimum_path:=Path_Optimization(start_block,end_block);
    loop_size:=循环域 L 子图的动态指令数;
    if (loop_size>LOOP_SIZE_THRESHOLD)
            curr_thread:=create_new_thread(end_block,spawn_pos,optimum_path);
    else if (loop_size≤LOOP_SIZE_THRESHOLD)
            unroll(loop L);
    end if
    return curr_thread;
}
算法1-3. Path_Optimization(start_block,end_block).{
   if (start_block==end_block)
     return 1;
   pdom_block==postdominator of start_block
   if (start_block.leftchild==pdom_block)
    {denoting the path between start_block and pdom_block;
       strat_block=pdom_block;
    Path_Determination(start_block,end_block);
       }
     else {determine the path according to the optimization rules;
        start_block=pdom_block;
    Path_Determination(start_block,end_block);
        }
算法 1-4. partition_thread(start_block,end_block,curr_thread).{
    if (start_block==end_block)
            return curr_thread;
    end if
    pdom_block:=start_block 的最近后向支配节点;
    path:=从 start_block 到 pdom_block 的优化路径;
    thread_size:=curr_thread+path 的动态指令数;
    if (is_medium(thread_size)) then
        curr_thread:=curr_thread+path;
            curr_thread:=create_new_thread(pdom_block,spawn_pos);
    else if (is_big(thread_size))
            curr_thread:=curr_thread+path.first_block;
            If (!is_small(curr_thread))
                curr_thread_1:=create_new_thread(path.first_block,spawn_pos);
            else
                curr_thread:=curr_thread+path.first_block;
                curr_thread_1:=curr_thread;
            end if
              curr_thread_1:=partition_thread(path.first_block,pdom_block,curr_thread_1);
        else
            curr_thread:=curr_thread+path;
         curr_thread:=curr_thread+pdom_block;
    end if
    curr_thread:=partition_thread(pdom_block,end_block,curr_thread);
    return curr_thread;
}
```

Fig.6 Pseudo code of the thread partitioning algorithm

图 6 线程划分算法伪代码

2.4 基于路径优化的线程划分

线程划分是影响 SpMT 系统性能最为关键的因素之一.在线程划分时,需要综合考虑第 1.2 节所讨论的影响并行性的各种开销,并最终选取最佳的推测线程.循环区域是开发程序并行性的重要部分,现有的划分算法大都是重点关注对循环区域的划分[22,23].然而对于非规则程序来讲,非循环区域也是提供并行性能的重要组成部分.表 1 给出了 Olden 测试程序集动态执行的统计信息.

Table 1 Comparison of dynamic instruction numbers between loop and non-loop region on Olden benchmarks (%)

表 1 Olden 测试程序集循环、非循环动态指令数对比 (%)

	Olden 程序								
	Bh	Bisort	Em3d	Health	Perimeter	Tsp	Voronoi	Power	Mst
非循环区域	47.3	73.7	28	22.2	100	99.8	55.8	40.5	55.5
循环区域	52.7	26.3	72	77.8	0	0.2	44.2	59.5	44.5

从表 1 中可以明显看出,非循环执行占了相当大的比例,特别是对于 bh,em3d,health 等程序,非循环区域都超过了 50%以上.因此,本文提出的算法将考虑包括循环区域和非循环区域在内的所有程序代码,以过程为划分单位,采用自上而下的方法实现线程划分.线程划分算法的伪代码如图 6 所示.

2.4.1 循环区域的划分

循环区域是开发程序并行性的重要部分.对于归结出来的每个循环节点,则根据图 6 中的算法 1-2 来指导循环域划分.在线程划分时,同时结合循环的迭代次数和循环体的动态指令数等信息.对于循环体较小迭代次数少的循环不考虑推测执行.若循环体较小迭代次数过多,则采取循环展开扩展循环体大小,之后进行非循环域划分.如图 6 中算法 1-2 所示,循环域在进入当前迭代后遇到激发点时,激发下一次的迭代过程.当循环体过大时,将对每个循环体按照非循环域的线程划分算法划分.初始化当前候选线程子图为空,从超级块控制流图的入口节点作为当前参考点开始递归遍历,在每一步递归中,将当前参考点的控制无关节点作为当前候选线程的开始点,调用 *partition_thread* 划分线程,直到当前区域的出口点结束.

2.4.2 非循环区域的划分

partition_thread 完成超级块控制流图中非循环域线程划分,它以两个超级块节点和当前候选线程子图为输入,尽可能地通过递归调用将两个节点间的程序段子图划分为多个线程,划分的策略是严格乱序的.另外,本文的线程划分是基于启发式规则进行的划分,其线程边界将尽可能地被置于程序 CFG 的控制无关点处,这将可能会导致推测线程大小差距较大.因此,在对循环区域和非循环区域的划分时,为了尽量减少负载不平衡的影响,线程划分框架引入线程大小(thread size)的限制,即一个线程中的动态指令数应严格大于最小限制 *LOWER_LIMITE* 和尽量小于最大限制 *UPPER_LIMITE*(考虑可能某些基本块会出现包含过多指令的情形).在图 6 所示的算法 1-4 中,*curr_thread* 表示当前候选线程子图,它包括了上一个候选线程的结束点到 *start_block* 之间的所有的基本块.*curr_thread* 为空,则意味着 *start_block* 就是上一个候选线程的结束点;否则,意味着 *curr_thread* 中的子图由于线程体过小而不能单独成为一个线程.*pdom_block* 是 *start_block* 的最近后向支配节点,它作为与当前参考点控制无关的线程分界点.*path* 是 *start_block* 到 *pdom_block* 的优化路径.

2.4.3 生成预计算片段

基于上述线程划分结果,依据第 2.3.1 节预计算的生成方式,对生成的线程构建各自的预计算片段.构建预计算片段时,输入来自于推测路径的 live-ins 向量集,然后在控制流图中沿 CQIP 到 SP 的推测路径后向遍历,进行程序切片,生成预计算片段.在程序切中过程中,当涉及到一个函数调用指令时,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,将会导致预计算片段过大.考虑到子程序的副作用是其返回值被分析,而其中的子程序代码可能是不需要的,因此,本文算法采取了在产生的预计算片段中裁剪函数调用指令的保守方案.这种保守方案会在一定程度上影响值预测的精确程度,但是随着函数返回值预测技术的发展,这种负面影响将会逐渐被消除.

2.4.4 算法复杂度分析

本文提出的基于路径优化的线程划分算法复杂度主要包括 3 个方面:结构化分析、路径优化及线程划分和预计算片段生成.结构化分析的运行时间主要由控制流图中规约迭代的次数决定.迭代的次数依赖于区域匹配的策略,结构化分析的最大运行时间将小于等于 $O(n\cdot\log n)$.路径优化的运行时间主要来源于程序切片的时间开销,极限情况下不超过 $O(n^2)$.而线程划分和预计算片段生成的运行开销则主要来自于对控制流图的遍历,其时间开销是 $O(n)$.因此,算法总的时间复杂度为 $O(n^2)$.

3 实验和性能分析

3.1 模拟环境

本文提出的基于路径优化的线程划分算法已经在 Prophet 编译系统平台上进行了实现.该编译系统是课题组基于开源的 SUIF/MACHSUIF 编译框架[24]实现的并行编译器原型.该系统已经实现了包括编译器、汇编链接器、模拟器、可视化视图等在内的完整工具链.Prophet 编译器具有包括控制流分析库和数据流分析库等编译优化工具,并采用了开放的程序结构,易于多种划分算法的集成和测试.Prophet 模拟器[25]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,用来从线程中取指并执行指令.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 2.

Table 2 Prophet processor configuration

表 2 Prophet 模拟器配置

配置项	项目值
发射/提交宽度	4
取指令队列	4
处理单元数	4
ALU 功能单元(整数、浮点数)	ALU(4 个):1 个时钟周期;MULT(1 个):4 个时钟周期;DIV(1 个):12 个时钟周期
多版本一级 Cache	4 路组相联、64KB,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本文提出的算法进行性能测试.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.同时,从表 1 可以看出,Olden 程序中非循环区域占有较大的比重,因此,Olden 基准程序测试集这些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 10M 条指令.另外,线程大小的阈值 *LOWER_LIMITE* 和 *UPPER_LIMITE* 分别被限定为 15 和 80 条指令[1,13].

3.2 实验结果及性能分析

为了展示本文提出的线程划分算法的有效性,本文首先给出线程划分的一些统计信息,见表 3.

在表 3 中,第 2 列给出的是线程的大小.从中可以看出,线程大小被限制在相当接近的范围内,这将可以有效地减少负载不平衡的影响.从表 3 的第 3 列和第 4 列给出的 P-slice 信息可以看出,P-slice 平均大小仅为 3.7,占线程比例只有 8.9%,这说明只根据推测路径进行预计算片段抽取可以有效地减少预计算的开销.同时,从第 5 列可以看出,线程仍然可以取得平均 76.2%的较高推测成功率.最后,从表 3 最后一列可以看出,本文提出的算法取得了重要的加速性能,基于 Olden 测试程序集的加速比平均值达到了 1.83.

接下来,为了便于直接进行算法的比较,更有效地说明本文方法的有效性,我们在 Prophet 编译系统平台上分别对基于最可能路径的传统方法[26]和基于路径评估的简单评估方法[14]进行了实现.在基于传统方法进行线

程划分时,线程划分过程只是简单地选取分支概率最大路径(即最可能路径).此种策略最大的特点就是贪婪地选取了局部控制依赖代价最小的路径,追求局部控制依赖代价最小,实现简单.基于简单评估方法的线程划分方法则是在线程划分时,通过建立评估模型对推测路径进行粗略评估,然后选取代价较小的推测路径进行划分.在文献[14]中,李远成等人就是采取了利用数据依赖数目(live-ins 变量)来大体评估数据依赖所引起的开销,同时结合控制依赖的影响,选取较优的推测路径.但由于不同的数据依赖引起的开销并不等同,因此,这种简单的评估方法具有评估精度不高的局限性.图 7 和图 8 分别给出了基于最可能路径的传统方法[19]和基于路径评估的简单评估方法[9]与本文提出的基于路径优化方法所得的测试程序加速比性能对比.

Table 3 Information statistics of thread partitioning

表 3 线程划分的相关统计信息

Olden 程序	线程大小	P-slice 大小	P-slice 比例(%)	推测成功率(%)	加速比
Bh	65.5	4.3	6.6	93.9	1.97
Bisort	35.4	3.5	9.9	56.3	1.37
Em3d	40.6	4.9	12.1	64.4	2.30
Health	38.8	3.4	8.8	94.5	1.87
Perimeter	36.5	3.2	8.7	68.3	1.30
Tsp	32	3.1	9.6	77.4	1.86
Power	56.4	3.8	6.7	78.5	2.13
平均值	43.6	3.7	8.9	76.2	1.83

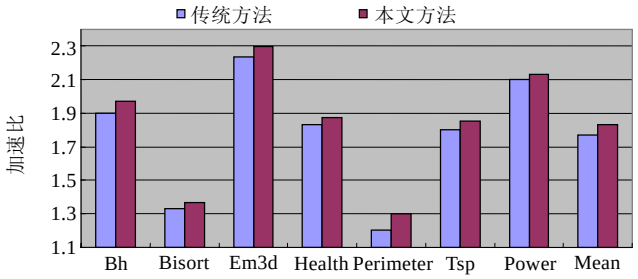


Fig.7 Comparison of speedup between our method and traditional method

图 7 本文方法与传统方法的加速比对比

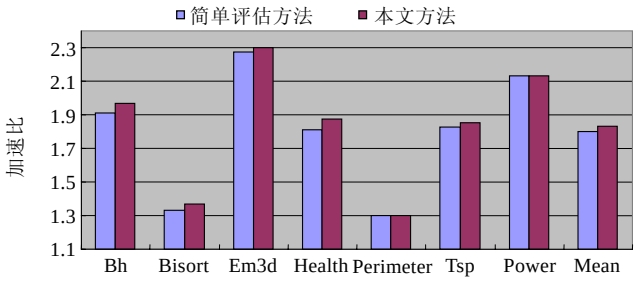


Fig.8 Comparison of speedup between our method and simple evaluation method

图 8 本文方法与简单评估方法的加速比对比

从图 7 可以看出,本文提出的划分算法在测试程序集上的加速比与基于最可能路径方法相比,均有不同程度的提高.对于 Perimeter,Bisort 和 Tsp,根据对程序的 WCFG 观察发现,这主要是由于这些程序都含有较大量的非循环区域,且其程序行为较为复杂,有相当一部分最可能路径的执行概率并不具有明显的优势;同时,相对于最可能路径,程序在概率较小的路径分支执行了更多的指令.对于 Mst,Power,Voronoi,Em3d 和 Health,根据对相应程序的 WCFG 观察发现,这些程序都含有较大量的循环区域.同时,这些循环区域大都具有复杂的程序行为;特别是 Em3d 和 Health,其循环区域的不同循环路径分支概率差距都比较小,而不同的循环路径长度差距则较大

(循环概率较小的路径拥有更长的路径长度).因此,针对程序的这些特点,本文方法通过综合考虑不同分支路径的数据和控制因素来进行推测,可以取得更好的加速性能.从图 8 可以看出,与基于简单路径评估方法相比,本文提出的划分算法在测试程序集上的加速比性能也都有一定程度的提升.特别是 Bh,Health,Bisort 等程序,均有较为明显的加速比性能提升.根据数据依赖模型的分析可知,这主要是因为这些程序中含有较多的预计算开销差距较为明显的 live-in 变量(如有的变量仅仅需要 1 条预计算指令,而有的则需要 3 条甚至更多);对于 Em3d 和 Tsp 程序,根据数据依赖模型的分析可知,虽然绝大部分 live-in 变量需要的预计算开销相差不大,但是这两个程序中均存在少量依赖距离较大且引起预计算开销都比较大(至少 3 条指令以上)的 live-in 变量,这些变量的错误推测将会引起严重的数据依赖开销.简单评估方法仅仅根据 live-in 变量个数来评估数据依赖开销,将使对不同路径的评估不够精确,因而导致不能选取更优的推测路径进行划分,从而最终影响程序的加速比性能.与上述程序不同,数据依赖分析显示,Perimeter 和 Power 程序中的 live-in 变量分布较为均匀,因而基于本文方法和简单评估方法的优化结果趋于一致,程序最终也因此加速比基本相同.总之,上述结果有效地说明,本文提出的基于预计算片段技术的路径优化的方法可以有效地选取更好的推测路径进行划分.另外,对于 Health 程序,其加速比提升程度明显小于预期(基于手工选取的最优路径其加速比可以达到 1.92 以上).这说明了本文所采用的数据依赖模型尚不够精确,可能会导致某些潜在的依赖信息不能被完全提取,并因此会导致数据依赖影响因素被弱化,导致评估出现一定程度的偏差.但是,这种评估偏差可以通过提高数据依赖模型的精确度来有效地消除或者降低.

表 4 给出基于 3 种方法编译器产生的推测线程的动态信息,以进一步说明算法的有效性.首先,对传统方法和本文方法进行对比和分析.对于 Bh,Perimeter,Bisort,Em3d,Power 等程序,基于本文方法所激发的线程数目均有较为明显增加;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 Bisort 程序,激发成功率只是微弱下降,但是激发的线程总数增加了近 3 倍.这说明,虽然牺牲一定程度上的推测成功率会增加线程撤销的时间开销,但是相对于传统方法,本文方法能够成功地激发更多的推测线程,通过推测包含更多代码的路径,可以实现更高程度上的程序并行,并因此得到加速比性能的提升.对于测试程序 Health 和 Tsp 等程序,从表 4 可以看出:相对于传统方法,基于本文方法的线程激发数目虽有不同程度的减少,但是线程激发的成功率却有较为明显的提高.同时,成功激发线程数目也略有增加.这说明,本文方法可以更有效地评估不同路径分支的开销,成功地选取了数据依赖程度更小的路径,并通过激发数据依赖程度更小的线程,从而实现更高的推测成功率.虽然激发线程数目一定程度的减少会降低程序的并行性,但是更高的推测成功率可以更有效地减少线程撤销所带来的时间开销,并因此使程序得到加速比性能的提升.

Table 4 Dynamic information statistics of three methods on Olden benchmarks

表 4 3 种方法基于 Olden benchmarks 的推测线程动态统计信息

		Olden 程序						
		Bh	Bisort	Em3d	Health	Perimeter	Tsp	Power
传统方法	线程发起数目	120 587	15 748	2 397	4 455	8 076	197 951	126 724
	推测成功率	0.960	0.570	0.735	0.904	0.757	0.683	0.797
简单评估方法	线程发起数目	121 054	16 154	3 486	4 258	9 677	183 844	128 883
	推测成功率	0.951	0.564	0.684	0.916	0.683	0.725	0.785
本文方法	线程发起数目	125 205	81 366	5 111	4 160	9 677	174 006	128 883
	推测成功率	0.939	0.563	0.644	0.946	0.683	0.774	0.785

进一步来说,对本文方法和简单评估方法进行性能对比和分析.对 Bh,Bisort,Em3d 等程序,基于本文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.对于 Tsp,Health 等程序,基于本文方法的线程激发数目虽有不同程度的减少,但是线程激发的成功率却有较为明显的提高;同时,成功激发线程数目也略有增加.类似于前述对传统方法和本文方法的分析,这些程序或是通过牺牲一定程度上的推测成功率来成功地激发更多的推测线程,以实现更高程度上的程序并行,或是通过降低一定的并行性,以带来更高的推测成功率,从而提升加速比性能.对于 Perimeter 和 Power 程序,从表 4 可以看到,线程动态信息没有变化.这说明在这些测试程序中,基于两种方法划分的情形一致,因此,程序的加速比保持了原来的性能.

4 结论和进一步的工作

线程划分是 SpMT 技术的关键因素之一.因此,如何有效地综合评估程序间控制依赖、数据依赖以及负载不平衡等因素至关重要.在本文中,首先对基于启发式规则进行线程划分的传统算法进行分析,并得出两个重要发现;然后,通过引入基于程序切片技术的预计算方法,建立了一种路径评估方法来评估程序间控制和数据依赖.进一步地,通过设置线程体粒度范围来有效地减小负载不平衡的影响.在此基础上,本文提出了一种基于路径优化的线程划分算法.本文提出的基于路径优化的线程方法的主要创新在于:(1) 在深入分析传统的基于启发式规则进行划分的基础上,得出两个重要发现来指导线程划分;(2) 提出一种基于预计算片段技术的路径评估方法来综合评估控制和数据依赖等因素,在此基础上提出一种基于路径优化的线程划分方法.基于 Olden 测试程序集最终的平均加速比性能达到了 1.83,这表明本文提出的算法可以有效地对非规则程序进行划分,并取得良好的加速比性能.

另外,由于代价评估所采用的数据依赖模型固有的不精确性,导致了某些数据依赖信息不能完全提取;同时,由于在生成预计算片段时对函数调用的处理采取了保守策略,这些都将会在一定程度上影响一些程序的加速比性能.因此,进一步的工作将从以下两个方面进行:(1) 从更接近机器的目标代码层面上进行寄存器数据依赖和内存数据依赖分析,提高数据依赖模型的精确度;(2) 进行函数返回值预测技术研究,进一步减少数据依赖的影响.

References:

[1] Bhowmik A, Franklin M. A general compiler framework for speculative multithreaded processors. IEEE Trans. on Parallel and Distributed Systems, 2004,15(2):713−724. [doi: 10.1109/TPDS.2004.26]

[2] Liu W, Tuck J, Ceze L, Ahn W, Strauss K, Renau J, Torrellas J. POSH: A TLS compiler that exploit program structure. In: Proc. of the ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming. New York: ACM Press, 2006. 158−167. [doi: 10.1145/1122971.1122997]

[3] Pei SW, Wu BF. SpMT WaveCache: Exploiting speculative multithreading for dataflow computer. Chinese Journal of Computers, 2009,32(7):1382−1392 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2009.01382]

[4] Wang S, Dai X, Yellajyosula KS, Zhai A, Yew PC. Loop selection for thread-level speculation. In: Proc. of 18th Int'l Workshop on Languages and Compilers for Parallel Computing (LCPC 2006). Berlin: Springer-Verlag, 2006. 289−303. [doi: 10.1007/978-3-540-69330-7_20]

[5] Johnson TA, Eigenmann R, Vijaykumar TN. Min-Cut program decomposition for thread-level speculation. In: Proc. of the ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI 2004). New York: ACM Press, 2004. 59−70. [doi: 10.1145/996893.996851]

[6] Johnson TA, Eigenmann R, Vijaykumar TN. Speculative thread decomposition through empirical optimization. In: Proc. of the ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PpoPP 2007). New York: ACM Press, 2007. 205−214. [doi: 10.1145/1229428.1229474]

[7] Luo Y, Packirisamy V, Hsu WC, Zhai A, Mungre N, Tarkas A. Dynamic performance tuning for speculative threads. In: Proc. of the ACM/IEEE 36th Annual Int'l Symp. on Computer Architecture (ISCA 2009). New York: ACM Press, 2009. 462−473. [doi: 10.1145/1555815.1555812]

[8] Liang B, An H, Wang L, Wang YB. Exploring of speculative thread-level parallelism from subroutine. Journal of Chinese Computer Systems, 2009,30(2):230−235 (in Chinese with English abstract).

[9] Wang Z, O'Boyle MFP. Mapping parallelism to multi-cores: A machine learning based approach. In: Proc. of the ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPOPP 2009). New York: ACM Press, 2009. 75−84. [doi: 10.1145/1594835.1504189]

[10] Mameesh RH, Franklin M. Speculative-Aware execution: A simple and efficient technique for utilizing multi-cores to improve single-thread performance. In: Proc. of the 19th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT 2010). New York: ACM Press, 2010. 421−430. [doi: 10.1145/1854273.1854326]

[11] Wang C, Wu YF, Borin E, Hu WL, Sager D, Ngai T, Fang J. Dynamic parallelization of single-threaded binary programs using speculative slicing. In: Proc. of the 2009 ACM SIGARCH Int'l Conf. on Supercomputing (ICS 2009). New York: ACM Press, 2009. 158−168. [doi: 10.1145/1542275.1542302]

[12] Bhowmik A, Franklin M. A general compiler framework for speculative multithreading. In: Proc. of the Annual ACM Symp. on Parallel Algorithms and Architectures (SPAA 2002). New York: ACM Press, 2002. 99−108. [doi: 10.1145/564870.564885]

[13] Carlos M, Carlos G, Jesu´s S, Pedro M, Antonio G, Dean MT, John PS. Mitosis: A speculative multithreaded processor based on precomputation slices. IEEE Trans. on Parallel and Distributed Systems, 2008,19(7):914−925. [doi: 10.1109/TPDS.2007.70797]

[14] Li YC, Zhao YL, Yin PP, Han B. A cost estimation based speculative path prediction method for speculative multithreading. Journal of Xi'an Jiaotong University, 2010,44(12):22−27 (in Chinese with English abstract).

[15] Sarkar V, Hennessy J. Partitioning parallel programs for macro-dataflow. In: Proc. of the Conf. on LISP and Functional Programming. New York: ACM Press, 1986. 202−211. [doi: 10.1145/319838.319863]

[16] Chen Z, Zhao YL, Pan XY, Dong ZY, Gao B. An overview of Prophet. In: Proc. of the 9th Int'l Conf. on Algorithms and Architectures for Parallel Processing (ICA3PP 2009). LNCS 5574, Berlin: Springer-Verlag, 2009. 396−407. [doi: 10.1007/978-3-642-03095-6_38]

[17] Carlisle MC. Olden benchmark suite. 1995. http://www.martincarlisle.com/olden.html

[18] Renau J, Tuck J, Liu W, Ceze L, Strauss K, Torrellas J. Tasking with out-of-order spawn in TLS chip multiprocessors: microarchitecture and compilation. In: Proc. of the 19th ACM Int'l Conf. on Supercomputing (ICS 2005). New York: ACM Press, 2005. 179−188. [doi: 10.1145/1088149.1088173]

[19] Lee J, Park H, Kim H, Jung C, Lim D, Han SY. Adaptive execution techniques of parallel programs for multiprocessors. Journal of Parallel and Distributed Computing, 2010,70(5):467−480. [doi: 10.1016/j.jpdc.2009.10.008]

[20] Giffhorn D, Hammer C. Precise slicing of concurrent programs: An evaluation of static slicing algorithms for concurrent programs. Journal of Automated Software Engineering, 2009,16(2):197−234. [doi:10.1007/s10515-009-0048-x]

[21] Zhai A, Colohan CB, Steffan JG, Andmowry TC. Compiler optimization of memory resident value communication between speculative threads. In: Proc. of the Int'l Symp. on Code Generation and Optimization (CGO 2004). Piscataway: IEEE Press, 2004. 39−50.

[22] Gao L, Xue JL, Ngai TF. Loop recreation for thread-level speculation on multicore processors. Software: Practice and Experience, 2010,40(1):45−72. [doi: 10.1002/spe.947]

[23] Prakash R, Akshay KB, Hariprasad K, Mohan M, Praveen J, Srivatsa S, Thejus VM, Prabhu V. A study of performance scalability by parallelizing loop iterations on multicore SMPs. In: Proc. of the 10th Int'l Conf. on Algorithms and Architectures for Parallel Processing (ICA3PP 2010). LNCS 6081, Berlin: Springer-Verlag, 2010. 476−486. [doi: 10.1007/978-3-642-13119-6_41]

[24] Hall M, Anderson J, Amarasinghe SP, Murphy BR, Liao SW, Bugnion E, Lam MS. Maximizing multiprocessor performance with the SUIF compiler. Computer, 1996,29(12):84−89. [doi: 10.1109/2.546613]

[25] Dong ZY, Zhao YL, Wei YK, Wang XH. Prophet: A speculative multi-threading execution model with architectural support based on CMP. In: Proc. of the 8th Int'l Conf. on Embedded Computing. Piscataway: IEEE Press, 2009. 103−108. [doi: 10.1109/EmbeddedCom-ScalCom.2009.128]

[26] Pan XY, Zhao YL, Chen Z, Wang XH, Wei YK, Du YN. A thread partitioning method for speculative multithreading. In: Proc. of the 8th Int'l Conf. on Embedded Computing. Piscataway: IEEE Press, 2009. 285−290. [doi: 10.1109/EmbeddedCom-ScalCom.2009.58]

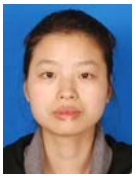
附中文参考文献:

[3] 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.

[8] 梁博,安虹,王莉,王耀彬.针对子程序结构的线程级推测并行性分析.小型微型计算机系统,2009,30(2):230−235.

[14] 李远成,赵银亮,阴培培,韩博.一种应用代价评估的推测多线程路径预测方法.西安交通大学学报,2010,44(12):22−27.

李远成(1981－),男,河南杞县人,博士生,主要研究领域为并行计算,体系结构,机器学习.



李美蓉(1984－),女,博士生,主要研究领域为并行计算,体系结构,机器学习.

赵银亮(1960－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为语言与编译系统,并行计算与机器学习.

杜延宁(1978－),男,博士生,主要研究领域为并行计算,体系结构,编程语言.