基于交互信息的投机并行化方法 *

李 莹, 孙煦雪, 袁新宇, 徐印成
(浙江大学 计算机科学与技术系, 杭州 310027)

摘 要: 针对投机并行化中如何权衡策略并确定合适的执行模型来获取理想性能的问题,提出了一种基于交互信息的投机并行化方法,利用交互信息来确定投机并行化的执行模型,建立相关评价模型,并着重从线程抽取创建角度提出了相应的策略及对应的性能评价。通过实验表明,基于交互信息进行"按需"并行化,可以达到所需的性能要求。

关键词: 投机多线程; 线程抽取创建; 执行模型; 并行化
中图分类号: TP314 **文献标志码**: A **文章编号**: 1001-3695(2010)06-2123-04
doi:10.3969/j.issn.1001-3695.2010.06.037

Interaction-based speculative thread-level parallelization

LI Ying, SUN Xu-xue, YUAN Xin-yu, XU Ying-cheng
(*College of Computer Science & Technology, Zhejiang University, Hangzhou* 310027, *China*)

Abstract: Aiming at the problem about how to trade off between different policies and execution models to get ideal performance in the exploration of speculative thread-level parallelism, this paper proposed an interaction-based method to exploit speculative thread level parallelism. This solution used interactive information to set up the execution model and evaluation model, especially from the view of thread extraction and spawning. The experimental results show that on demand parallelization based on interactive information can achieve the performance goal.

Key words: speculative multi-threading; thread extraction and thread spawn; execution model; parallelization

在并行编译中,当存在大量模糊依赖时,传统并行编译器不得不采用保守的策略来保证程序执行的正确性,这大大限制了串行程序可以挖掘的并发程度。为解决这些问题,研究人员提出了投机并行化[1],它将串行程序转换成多个投机执行的并行线程,并在运行时检查数据依赖。

投机并行化系统可以由纯软件实现[2,3],也可以由硬件支持[4~7]或软硬件相结合[8~10]来完成。其主要任务包括多线程抽取和多线程的投机执行。投机并行化中的开销有多种来源,找到最优的程序分解来获得性能最优是一个 NP 完全问题[11],不同的执行模型及实现机制会涉及不同的性能考虑,也会产生不同的开销代价。如何在各种策略中进行权衡,确定合适的执行模型以及建立对应的开销模型和性能评估,使用投机并行化来获取理想性能,这就是本文研究的问题域。

1 交互式投机并行化

1.1 概述

本文使用的方法是结合交互式和投机并行化,利用与用户交互获取的信息来确定投机并行化的执行模型,建立相关评价模型,"按需"并行化来达到性能要求。

本文使用软硬件相结合来实现投机并行化系统,其框架结构如图 1 所示。多线程抽取主要由静态时刻编译系统支持,包括对串行程序进行划分、多线程识别标记、进行程序变换和编译优化、生成多线程并行化代码,而多线程执行由运行时系统支持,包括多线程产生、多线程提交,使用值预测、调度及同步等技术来进行性能优化;通过硬件支持来实现对内存一致性的维护,包括检测数据依赖冲突、投机运行状态和投机数据的缓存、误投机的回滚和恢复等。

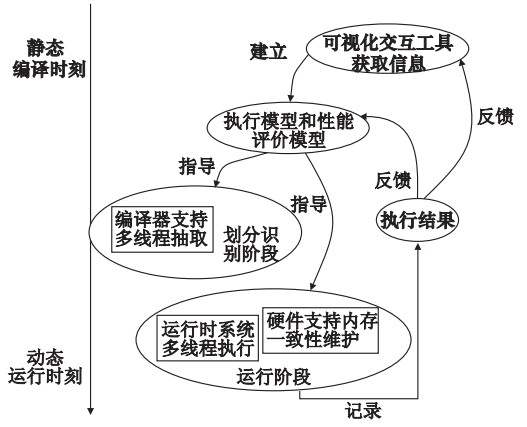


图1 基于交互信息的投机多线程并行化系统

投机多线程并行化中存在的性能和开销评估对象主要有并行粒度、负载、内存、时间和实现复杂度五个方面,详细的性能和开销考虑点如表 1 所示。

收稿日期: 2009-11-05; **修回日期**: 2009-12-28 **基金项目**: 国家自然科学基金资助项目(60873045)

作者简介:李莹(1973-),男,浙江衢州人,副教授,博士,主要研究方向为编译、中间件(cnliying@zju.edu.cn);孙煦雪(1984-),女,硕士研究生,主要研究方向为并行计算、编译优化;袁新宇(1978-),男,博士研究生,主要研究方向为多核并行编译;徐印成(1985-),男,硕士研究生,主要研究方向为编译优化.

本文的主要贡献在于:a)提出交互方式进行投机多线程并行化,充分挖掘性能与执行模型及投机策略间的对应关联,以达到所需的性能要求;b)从线程抽取创建角度提出了相应的策略及对应的性能评价,提供交互工具来确定策略设计。

表 1　投机多线程并行化的性能评估

评估对象	优质性能考虑点	开销代价考虑点
并行粒度	粗细粒度结合,并行性充分挖掘	并行性挖掘不充分,粒度不符合需要
负载	负载均衡	负载不平衡
内存	内存冲突数量少 内存一致性高 投机缓冲区有效使用 缓存局部性高	内存冲突多 内存不一致 投机缓冲区低效使用 缺乏缓存局部性
时间	误投机时线程挤压重启时间少 降低投机缓冲区溢出率及溢出带来的时间开销 线程间通信花费时间少 线程分派和提交花费时间少 线程同步开销少 并行代码指令执行时间开销少 运行时值预测、调度、预取等优化技术的时间开销少 线程新建的时间开销少 依赖冲突检测时间开销少	误投机时线程挤压重启时间开销 投机缓冲区溢出的时间开销 线程间通信的时间开销 线程分派和提交产生的时间开销 线程同步产生的时间开销 代码并行执行的时间开销,包括 load、store 指令延迟的开销 运行时值预测、调度、预取等优化措施产生的时间开销 线程创建启动带来的时间开销 依赖冲突检测的时间开销
复杂度	软硬件实现复杂度可接受	软硬件实现复杂度高

1.2　“按需”投机并行化

1.2.1　基于交互信息的执行模型

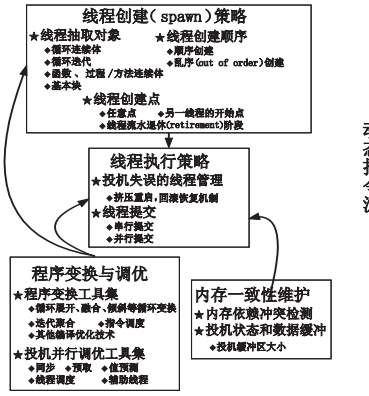
在投机并行化系统中,编程人员通过可视化交互工具输入相关机制设计和策略选择信息。根据这些交互信息,可变的投机多线程并行化执行模型经过类实例化过程成为确定的执行模型。本文提供的可变的投机多线程执行模型包括线程抽取创建策略、线程执行策略、程序变换与调优机制以及内存一致性维护机制,如图 2 所示。其中,程序变换工具集提供了多种编译优化技术,如循环展开、迭代聚合等,可由程序员定制,在线程抽取时选用以对程序进行变换优化;投机并行调优工具集提供了值预测、预取、强制同步、线程调度及辅助线程等机制,可由程序员定制启用,在多线程投机执行时进行相关优化来达到性能目的;内存一致性维护机制主要涉及内存依赖冲突检测机制和缓冲投机状态、投机数据的策略设计,其中包括投机缓冲区大小等的设置。

为了更清晰地进行本阶段的研究,即主要在线程抽取创建策略的设计与对应性能评价模型的探索空间中进行研究,本文对研究模型进行一定的简化,限定其中部分可变量因素。该研究执行模型使用硬件来检测内存依赖冲突并对投机状态和数据进行缓冲,维护内存的一致性,默认采用循环展开和迭代聚合的程序变换技术以及值预测机制,线程执行中采用顺序提交,而将线程抽取创建作为可变量。下面分别对线程抽取创建及其对应的性能评价模型进行建模研究。

1.2.2　线程的抽取与创建

线程抽取创建机制包括投机线程抽取对象的选择、投机线程创建顺序和创建点。线程抽取创建机制决定投机线程的大小及程序划分的结果,影响执行时间开销和内存开销,与投机成功率及误投机产生的开销相关联。

线程抽取包括循环的选择和非循环部分的划分。抽取创建策略决定了投机线程执行的代码以及其开始执行的时间,主要涉及两个点的选定,即创建点 SP(spawning point,表示新投机线程创建点)和控制准独立点 CQIP(control quasi-independent point,表示新创建的投机线程开始执行点)[12]。如图 3 所示的投机多线程创建和运行中,标志了 SP 和 CQIP 的位置。线程 n 照常执行指令流,直到到达一个创建点;在这个创建点,处理器标志一条在未来很有可能执行的指令(即指定控制准独立点 CQIP)。接着,线程 $n+1$ 创建一个投机新线程并在 CQIP 开始执行,而线程 n 继续执行指令直到线程的汇合点,即 CQIP。运行时刻硬件会进行依赖检测,若检测到内存依赖冲突,则投机线程将被挤压销毁(squash)。



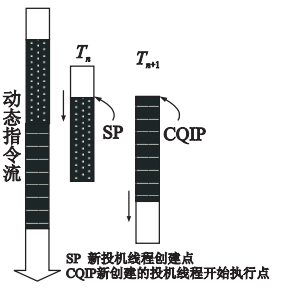


图2　灵活可变的执行模型　　图3　投机多线程的创建运行

有效地选定指令作为创建点和控制准独立点来组成创建对(spawning pairs)才能提高线程级并行度,充分挖掘程序的并行性。线程抽取的对象主要有循环连续体,循环迭代,函数、过程/方法连续体和基本块。针对这些对象选择合适的 SP 和 CQIP,使得运行时访问 SP 后到达 CQIP 的概率非常高,以此作为有效选择创建对的首要标准[12]。将循环迭代(loop iteration)作为投机多线程抽取的对象,是因为在不考虑跳转结果的情况下,一个循环迭代开始后,下一个迭代极其有可能还在循环体中,即迭代会再次被执行。与循环迭代相关的线程创建机制叫做循环迭代创建策略,这种策略默认将循环中静态顺序的第一条指令作为 SP 和 CQIP。另一个高可预测性的控制流点是循环连续体(loop continuation),将其作为投机多线程抽取对象,是因为在不考虑循环中的控制流结果的情况下,开始循环后,结束循环的后向跳转(backward branch)指令后面的一条指令极其有可能被执行。与循环连续体相关的线程创建机制叫做循环连续体创建策略,这种策略默认将循环中静态顺序的第一条指令作为 SP,结束循环的后向跳转的下一条指令作为 CQIP。除以上所述外,子过程连续体(subroutine continuation)也可作为线程抽取的考虑对象,是因为在不考虑过程中路径的情况下,过程调用返回后的下一条指令极其有可能被执行。与子过程连续体相关的线程创建机制叫做子过程连续体创建策略,这种策略默认将子过程调用作为 SP,子过程调用后的下一条指令作为 CQIP。每次在一个过程调用指令执行时就创建一个投机线程,并在调用后的下一条指令处开始执行。

为了清晰描述几种线程抽取对象及其标志,现规定如下符号规则:

$\triangleright$ definition of;　$\prec$ instance of;　$\Leftrightarrow$ equivalence of

其中:$A \triangleright B$ 表示 A 是 B 的定义,$A \prec B$ 表示 A 是 B 的一个实例,$A \Leftrightarrow B$ 表示 A 与 B 等价。

线程抽取对象可形式化描述为:应用程序中的过程由路径组成,$\{p \mid p \prec \mathrm{RP}\} \triangleright \mathrm{routine}$。程序中的路径是一个由节点链、路径类型、长度和概率组成的四元组$(X, \tau\langle ? \rangle, \lambda, \rho) \triangleright P$。其中 X

是一组节点,每个节点可以是基本块或循环体或调用,list(n) $\wedge n \prec \{BB \mid \Delta_l \mid \Delta_c\} \rhd X$;$\tau\langle ? \rangle$ 是模板路径类型,可以是循环路径类型 $\tau_l \prec \{\text{continue} \mid \text{break} \mid \text{exit}\}$,或过程路径类型 $\tau_r \prec \{\text{return} \mid \text{exit}\}$。将路径类型具体化后,就可得到循环路径 apply(P,τ_l) $\rhd$LP 或过程路径 apply(P,τ_r) $\rhd$RP。一组循环路径即构成循环实例 list(LP) $\rhd \varphi_l$,而一条过程路径就是一个过程实例 RP$\Leftrightarrow \varphi_c$。循环实例和过程实例分别可构成循环实例环行链和过程实例环行链 cList($\varphi\langle ? \rangle$) $\rhd B$,apply(B,φ_c) $\rhd B_c$,apply(B,φ_l) $\rhd B_l$。对于循环节点,它是由循环实例环行链、指向链首实例的指针、循环计数及长度来表征(B_l,θ,t,λ) $\rhd \Delta_l$。对于过程节点,它是由过程实例环行链、指针及长度来表征(B_c,θ,λ) $\rhd \Delta_c$。根据控制流图提供的信息,对一个程序进行遍历,生成路径和相关节点,同时分析可识别出循环连续体、循环迭代和过程连续体。循环迭代作为线程抽取对象时,要考虑一类循环节点,其循环实例环行链中的元素的路径类型 $\tau_l \Leftrightarrow$continue。循环连续体作为线程抽取对象时,要考虑那些循环实例环行链中元素的路径类型为 $\tau_l \Leftrightarrow$break 的循环节点;过程连续体作为线程抽取对象时,要考虑那些过程实例环行链中元素的路径类型 $\tau_r \Leftrightarrow$return 的过程节点。

分析出线程抽取对象后,需要计算和确定 SP 和 CQIP。SP 的选择可以选在线程开始时,对于循环和调用而言分别对应于循环主体开始和函数调用之前。文献[13]就是采用这种策略:对于循环,SP 位于循环主体的开始,每一个循环迭代将下一个迭代创建为投机线程;对于函数调用,SP 位于函数调用之前,非投机线程执行函数体,而从函数后续体创建投机线程。这种选择策略能够最大化处理器利用率,减少空闲时间,但会导致投机线程产生太早,不能充分考虑运行时信息,而且线程过早产生,导致没有空闲可用的处理器可供分配。另一种 SP 的选择是在线程中任意点可创建,具有较大灵活性,这种将创建投机线程延迟的方法可以充分利用运行时信息,直至能够解析一个特定跳转或数据依赖时才创建,能够提高投机成功概率,缺点是可能会导致处理器空闲时间过大。确定了 SP 后,需要选择 CQIP。除了上文提到的高概率可达点标准外,还需考虑 SP 和 CQIP 间的距离大小,需要保持合适的距离以维持线程的大小在限定范围内。线程粒度太小将不能充分挖掘程序并行性,导致过度的线程初始化开销;而大线程将会导致工作负载不均衡、投机数据和状态缓冲区过大以及误投机后的恢复开销大。另外,应该选择合适的 CQIP,使得其后面的指令与之前的指令存在较少依赖关系,即使存在依赖关系,其依赖关系中的数值应该可预测。标志 CQIP 的标准,就是保证 SP 和 CQIP 间的指令与 CQIP 后面的指令尽可能相互独立无依赖[12]。

本方法提供可视化交互工具,通过用户交互来获取线程抽取创建机制中参数的相关信息,如投机线程抽取对象、投机线程创建点、投机线程粒度选择等。根据这些信息,投机多线程并行化执行模型采用如下所述的线程抽取创建的算法来确定 SP 和 CQIP,从而确定线程抽取创建策略。

```
Input:cfg<control flow information.
Output: list(pr) where pair(snode, cnode) ▷ pr.
Declaration
choice extobj < p
where p ∈ (P({LI,LC,SC}) - ∅)
LI: loopIteration;
LC:loopContinuation;
SC:subrouinteContinuation
choice point < {firstAtOnce, delay}
suggestion grain < {coarse, fine, mid}
node sp
nodeList cCandi
float prob
Process
phasel: interact with programmer, instantiate extObj, point, grain and
prob(especially considering those branches)
phase 2:
one traversal and mark to generate list(Δ1) and list(Δc);
case loopIteration ∈ extObj
    compute(list(Δ1), τ1, continue);
case loopContinuation ∈ extObj
    compute(list(Δ1),τ1, break);
case subrouinte Continuation ∈ extObj
    compute(list(Δc), τc, return);
end
function compute(list(Δ⟨?⟩) li, τ⟨?⟩t, type ty)
foreach elem in li
  if(t = = ty){
    sp←analyzeSP(elem, point);
    cCandi←analyzeCandi(sp);
    pr←(sp, refine(cCandi));
    add pr to output list;
  }
```

完成 SP 和 CQIP 的计算后,就确定了一组需要创建的投机线程的序列。一个投机多线程执行模型可以支持顺序或乱序创建投机线程。如果允许乱序创建,则线程不需要按程序顺序进行创建,对于两段顺序执行中相隔较远的代码,可以在其交叉的代码区创建线程之前并行执行。文献[13]采用的投机多线程协议就是采用乱序创建,同时在创建一个克隆线程时传递寄存器状态。乱序创建投机线程挖掘了更多的任务级并行性,但也可能导致死锁或系统开销过大等问题。为了防止乱序创建中可能出现的死锁情况,投机多线程处理器需要对一些线程采取抢占机制;同时,投机执行模型中的乱序创建的深度应该是有限的。例如,乱序深度为 1 的情况下,一个线程被创建后至多一个前驱线程可以被创建,当前驱线程随后被创建而导致当前线程被抢占,处理器必须保存至多一个线程的状态。

本机制中,程序员可通过可视化交互工具来输入创建次序的选择以及支持乱序创建时的乱序深度,以此来确定运行时刻投机线程的创建策略。

2　实验结果与分析

本文通过用户交互信息来确定投机多线程执行模型,其运行时刻总的时间开销可建模为

$$T^{\infty} = \max(T_{i-s}) + T_{\text{disp/comm}}(P+1)/2 + \max_{i=1..p}(T_{i-e}) + T_f \times \partial_{\text{fail}} + T_o \times \partial_{\text{overflow}} + T_{\text{pred}}$$

其中:T_{i-s}是第 i 个线程的创建时间;T_{i-e}是第 i 个线程的主体代码指令执行时间;∂_{fail}是投机失败的概率;T_o 是投机缓冲区溢出的时间开销;$\partial_{\text{overflow}}$是投机缓冲区溢出的概率;$T_{\text{pred}}$是运行时值预测产生的时间开销。

投机多线程并行化方案的加速比为

$$\text{speedUp} = \frac{\sum_{i=1}^{\text{num}} T_i}{T^{\infty}}$$

其中:T_i 是线程 i 原始执行的时间。

图 4 所示为投机线程创建机制设定的一个应用场景。本文

的实验环境使用 SESC 模拟器[14]模拟支持投机多线程的 4 核体系环境。图 5 ~7 从加速比的性能需求角度显示了投机线程创建机制中不同策略的选择所产生的影响。其中,图 5 为选择不同的投机线程创建顺序时的性能结果;图 6 为选择不同的投机线程抽取对象时的性能结果;图 7 为选择不同的投机线程粒度大小时的性能结果。图 8 是投机线程乱序创建的系统开销。

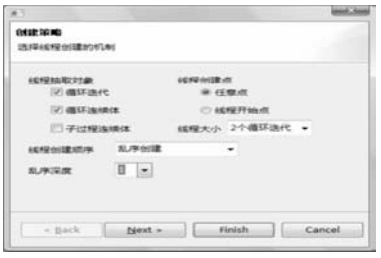


图4　策略选择应用实例

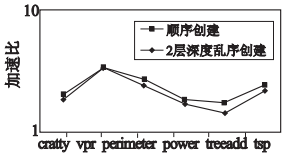


图5　不同的投机线程创建顺序的性能结果

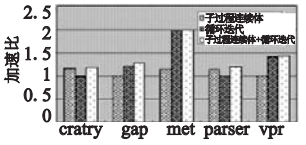


图6　不同的投机线程抽取对象的性能结果

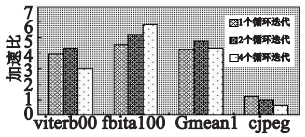


图7　不同的投机线程粒度大小的性能结果

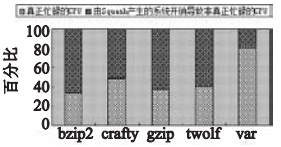


图8　投机线程乱序创建的系统开销

3　相关工作

近年来有不少研究从软件和硬件角度来支持投机并行化[10,15,16],这些框架采用了不同的执行模型来进行投机并行化。对于投机执行模型中的线程抽取创建策略,Bhowmik 等人[17]在 SUIF-MACHSUIF 平台上设计了一个投机多线程框架,使用基于依赖的任务选择算法设计线程创建策略来进行投机并行化,他们使用基于简单启发式的静态编译分析来选取投机线程。除了静态编译优化方法,另外一些研究[11,12,16]使用评测(profiling)来识别好的线程划分以进行投机执行。文献[11]通过评测运行时执行次数选取线程的方法来避免静态分解评估准确性的问题,同时在评测时刻实现分解来避免动态分解开销。对于线程抽取对象,循环迭代是程序中创建线程最明显的部分,很多投机多线程的研究都专注于循环迭代[16,18,19],也有针对子程序结构开发线程级并行性[20]。Du Zhao-hui 等人[19]提出了一个开销驱动的编译框架,静态确定哪个循环可以并行化,他们从控制流图和数据依赖图中计算开销图,并评估误投机的概率。文献[20]总结出使用循环迭代创建策略、增量值预测器以及一个无限连接体系结构的设计,对于挖掘投机线程级并行性是非常有效的。对于投机执行模型中的线程提交策略,文献[3]通过实验验证了串行提交会成为性能瓶颈,而部分提交和并行提交能有效降低开销。对于投机并行化中内存一致性的维护,文献[2]提出了一种激进的滑动窗口机制,同时在投机并行化框架中实现了利用规约同步约束对投机存储操作进行数据依赖冲突检测。

4　结束语

本文提出了基于交互信息的投机并行化方法,利用交互信息确定投机执行模型,并将线程抽取创建作为可变因素对线程抽取创建策略及其对应的性能评价模型进行了研究。

在未来研究中,笔者将继续探索投机并行化执行模型,并从以下几方面对当前工作进行扩展和改进:

a)目前设计的方法对开发人员有一定投机执行的背景要求,在未来的工作中,将设计更为通用简便的交互方式,将投机并行化的性能需求和执行策略映射为更明晰的表达方式,为普通使用者提供友好的交互支持。

b)将线程提交机制以及程序变换与调优作为可变因素,提供交互来支持更为灵活的投机执行模型。

参考文献:

[1] OPLINGER J T, HEINE D L, LAM M S. In search of speculative thread-level parallelism[C]//Proc of the 8th International Conference on Parallel Architectures and Compilation Techniques. Washington DC: IEEE Computer Society, 1999: 303-313.

[2] CINTRA M, LLANOS D R. Toward efficient and robust software speculative parallelization on multiprocessors[C]//Proc of the 9th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York: ACM Press, 2003: 13-24.

[3] CINTRA M, LLANOS D. Design space exploration of a software speculative parallelization scheme[J]. IEEE Trans on Parallel and Distributed Systems, 2005, 16(6): 562-576.

[4] PRVULOVIC M, GARZARÁN M J, RAUCHWERGER L, *et al*. Removing architectural bottlenecks to the scalability of speculative parallelization[C]//Proc of the 28th Annual International Symposium on Computer Architecture. New York: ACM Press, 2001: 204-215.

[5] SOHI G S, BREACH S E. VIJAYKUMAR T N. Multiscalar processors[C]//Proc of the 22nd Annual International Symposium on Computer Architecture. New York: ACM Press, 1995: 414-425.

[6] TSAI J Y, HUANG Jian, AMLO C, *et al*. The superthreaded processor architecture[J]. IEEE Trans on Computers, 1999, 48(9): 881-902.

[7] STEFFAN J G, COLOHAN C, ZHAI A, *et al*. The STAMPede approach to thread-level speculation[J]. ACM Trans on Computer System, 2005, 23(3): 253-300.

[8] CHEN M, OLUKOTUN K. TEST: a tracer for extracting speculative threads[C]//Proceedings of International Symposium on Code Generation and Optimization. Washington DC: IEEE Computer Society, 2003: 301-312.

[9] OOI C L, KIM W S, PACK I I, *et al*. Multiplex: unifying conventional and speculative thread-level parallelism on a chip multiprocessor [C]//Proc of the 15th International Conference on Supercomputing. New York: ACM Press, 2001: 368-380.

[10] OHSAWA T, TAKAGI M, KAWAHARA S, *et al*. Pinot: speculative multi-threading processor architecture exploiting parallelism over a wide range of granularities[C]//Proc of the 38th Annual IEEE/ACM International Symposium on Microarchitecture. Washington DC: IEEE Computer Society, 2005: 81-92.

[11] JOHNSON T A, EIGENMANN R, VIJAYKUMAR T N. Speculative thread decomposition through empirical optimization[C]//Proc of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York: ACM Press, 2007: 205-214.

(下转第 2139 页)

参考文献:

[1] CANFORA G, CERULO L. How software repositories can help in reoslving a new change request[C]//Proc of Workshop on Empirical Studies in Reverse Engineering. 2005.

[2] ANVIK J, HIEW L, MURPHY G C. Who should fix this bug[C]//Proc the 28th of International Conference on Software Engineering. New York: ACM Press, 2006:361-370.

[3] FISCHER M, PINZGER M, GALL H. Analyzing and relating bug report data for feature tracking[C]//Proc of the 10th WCRE IEEE. Washington DC: IEEE Compnter Society, 2003:90-99.

[4] 韩卫岗,周红建,赵禄丰. 软件缺陷信息分析研究[J]. 计算机工程与设计,2008,29(13):3381-3383,3387.

[5] 刘英博,王建民. 面向缺陷分析的软件库挖掘方法综述[J]. 计算机科学,2007,34(9):1-4,11.

[6] 刘海 郝克刚. 软件缺陷数据的分析方法及其实现[J]. 计算机科学,2008,35(8):262-264.

[7] 李宁,李战怀. 软件缺陷数据处理研究综述[J]. 计算机科学,2009,36(8):21-25,78.

[8] 尹相乐,马力,关昕. 软件缺陷分类的研究[J]. 计算机工程与设计,2008,29(19):4910-4913.

[9] JALBERT N, WEIMER W. Automated duplicate detection for bug tracking systems[C]//Proc of IEEE International Conference on Dependable Systems and Networks with FTCS and DCC, DSN. 2008: 52-61,24-27.

[10] CUBRANIC D, MURPHY G C. Automatic bug triage using text classification[C]//Proc of SEKE. [S.l.]: KSI Press, 2004: 92-97.

[11] ANVIK J K. Assisting bug report triage through recommendation [R]. [S.l.]: University of British Columbia, 2007.

[12] WANG Xiao-yin, ZHANG Lu, ANVIK J, *et al*. An approach to detecting duplicate bug reports using natural language and execution information[C]//Proc of the 30th International Conference on Software Engineering. New York: ACM Press, 2008:461-470.

[13] WEISS C, PREMRAJ R, ZIMMERMANN T, *et al*. How long will it take to fix this bug[C]//Proc of the 4th International Workshop on Mining Software Repositories. Washington DC: IEEE Computer Society, 2007.

[14] ANVIK J. Automating bug report Assignment[C]//Proc of the 28th International Conference on Software Engineering. New York: ACM Press,2006:937-940.

[15] BETTENBURG N, JUST S, SCHRTER A, *et al*. Quality of bug reports in Eclipse[C]//Proc of OOPSLA Workshop on Eclipse Technology Exchange. New York: ACM Press, 2007: 21-25.

[16] BETTENBURG N, JUST S, SCHRTER A, *et al*. What makes a good bug report[C]//Proc of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York: ACM Press, 2008:308-318.

[17] KO A J, MYERS B A, CHAU D H. A linguistic analysis of how people describe software problems[C]//Proc of IEEE Conference on Visual Language and Human-Centric Computing. 2006:127-134.

[18] BETTENBURG N, PREMRAJ R, ZIMMERMANN T, *et al*. Extracting structural information from bug reports[C]//Proc of the 5th Working Conference on Mining Software Repositories. New York: ACM Press,2008:27-30.

[19] BETTENBURG N, PREMRAJ R, ZIMMERMANN T, *et al*. Duplicate bug reports considered harmful... really[C]//Proc of the 24th IEEE International Conference on Software Maintenance. 2008:337-345.

[20] BAEZA-YATES R, RIBEIRO-NETO B. Modern information retrieval [M]. New York: Addison Wesley, 1999.

[21] GUNN S R. Support vector machines for classification and regression [R]. [S.l.]: University of Southampton, Faculty of Engineering, Science and Mathematics; School of Electronics and Computer Science, 1998.

[22] RUNESON P, ALEXANDERSON M, NYHOLM O. Detection of duplicate defect reports using natural language processing[C]//Proc of ICSE. Washington DC: IEEE Computer Society,2007:499-510.

[23] 庞剑锋,卜东波, 白硕. 基于向量空间模型的文本自动分类系统的研究与实现[J]. 计算机应用研究,2001,18(9):23-26.

[24] ALLAN J, ASLAM J, BELKIN N, *et al*. Challenges in information retrieval and language modeling[J]. ACM SIGIR Forum, 2003,37(1):31-47..

(上接第 2126 页)

[12] MARCUELLO P, GONZÁLEZ A. Thread-spawning schemes for speculative multithreading[C]//Proc of the 8th International Symposium on High-performance Computer Architecture. Washington DC: IEEE Computer Society, 2002: 55-64.

[13] XEKALAKIS P, IOANNOU N, CINTRA M. Combining thread level speculation helper threads and runahead execution[C]//Proc of the 23rd International Conference on Supercomputing. New York: ACM Press,2009: 410-420.

[14] SACK P. SESC: SuperESCalar simulator [EB/OL]. (2004-12-20). http://iacoma.cs.uiue.edu/~pau/sack/sescdoc/sescdoc.pdf.

[15] MADRILES C, GARCÍA-QUIÑONES C, NCHEZ J. Mitosis: a speculative multithreaded processor based on precomputation slices[J]. IEEE Trans on Parallel and Distributed Systems, 2008, 19(7): 914-925.

[16] LIU Wei, TUCK J, CEZE L, *et al*. POSH: a TLS compiler that exploits program structure[C]//Proc of the 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York: ACM Press,2006: 158-167.

[17] BHOWMIK A, FRANKLIN M. A general compiler framework for speculative multithreading[C]//Proc of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures. New York: ACM Press, 2002: 99-108.

[18] 刘圆,安虹,汪芳,等. 利用连续两阶段在线剖析优化多线程推测执行[J]. 小型微型计算机系统,2009,30(3):385-390.

[19] DU Zhao-hui, LIM C C, LI Xiao-feng, *et al*. A cost-driven compilation framework for speculative parallelization of sequential programs [C]//Proc of ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM Press, 2004: 71-81.

[20] 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[J]. 小型微型计算机系统,2009,30(2):230-235.