帮助线程预取技术研究综述

张建勋[1,2] 古志民[1]

(北京理工大学计算机学院 北京 100081)[1] (天津中医药大学网络中心 天津 300913)[2]

摘 要 帮助线程预取是当前多核平台提高非规则数据密集应用预取效果性能的关键技术之一,近年来已成为国内外的研究热点。针对非规则数据密集应用访存规律的非连续局部性特征,帮助线程预取技术利用CMP平台的最后一级共享缓存(LLC)将应用的非连续局部性转换为瞬时的连续时空局部性(即时局部性),从而达到通过线程级数据预取提高程序性能的目的。归纳了帮助线程预取技术的分类,概括和比较了不同帮助线程实现技术的优势和局限性,深入分析和探讨了现有的几种典型帮助线程技术的预取控制策略。最后从帮助线程实时控制、参数动态选取和优化方面指出了帮助线程预取技术的研究方向。

关键词 帮助线程,数据预取,CMP(Chip Multi-Processor)平台,非规则数据密集应用

中图法分类号 TP303 **文献标识码** A

Survey of Helper Thread Prefetching

ZHANG Jian-xun[1,2] GU Zhi-min[1]

(School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China)[1]

(Network Center, Tianjin University of TCM, Tianjin 300913, China)[2]

Abstract Helper thread prefetching is one of the key techniques to improve the prefetch effect of non-irregular data intensive applications. It has become a hot research topic at all over the world in recent years. Aiming at the memory access characteristic of discontinuous locality of non-irregular data intensive applications, helper threading could effectively convert discontinuous locality into continuous-instant spatial or temporal locality by using the shared LLC of CMP platform. And as a result, the application's performance can be improved. In this paper, the classification of helper thread prefetching techniques was summarized from the perspective of implementation method. The limitation and superiority of different types of prefetching were compared and surveyed. The current helper thread prefetching control policy was systematically analyzed and compared. Finally, several major issues and research directions of helper thread prefetching for further exploration were also pointed out.

Keywords Helper thread, Data prefetching, CMP platform, Non-irregular data intensive application

随着多核和云计算技术的发展,数据量呈指数型、爆炸式增长趋势。数据密集计算(Data Intensive Computing)[1]应用遍布于如天文物理、人工智能、信息安全、社会网络分析等整个科学计算领域,已经成为当前学术界和工业界的研究重点和热点。对于非规则数据密集型计算[2],其应用程序算法通常依赖于图、树或者链表等存储结构来实现。由于数据规模庞大,DIC应用的访存行为呈现非规则性,其访存模式不能在静态编译或编程时进行预测,使得传统的动态延迟隐藏技术(如数据缓存技术和传统的数据预取技术)失效,其可利用的空间局部性和时间局部性受到极大的限制。然而,利用帮助线程预取技术来构造多核缓存的即时局部性成为解决这一问题的有效方法。帮助线程预取技术主要源自于预执行[3-13]的思想,针对源程序的程序热点区域,通过构造一个精简版本的数据预取线程,利用线程级并行(Thread Level Parallel)通过多核平台的最后一级共享缓存(Last Level Cache)达到数据预取的目的。帮助线程只起到数据预取作用,并不影响源程序的逻辑计算结果。针对当前CMP平台,如何对帮助线程进行有效的预取控制,提高预取质量,最大限度地减少帮助线程的负获益是帮助线程预取技术研究的主要焦点问题之一。

1 帮助线程预取技术的思想渊源

为了充分利用多核处理器的处理核,学术界提出了多种程序执行模型,如TLS(Thread Level Speculative)、RA(Run Ahead Execution)、HT(Helper Thread)等。其中,基于预执行思想的帮助线程预取技术是近年来多核平台的延迟隐藏技术研究的一个热点问题之一。帮助线程思想最早可追溯到辅助执行思想[14]、Leader/Follower思想、预执行和预计算思想。

1.1 Leader/Follower思想

同一个程序分别在两个不同的处理器上运行,一个运行在前,充当"领导者"角色,另外一个运行在后,充当"跟随者"

到稿日期:2012-09-17 返修日期:2012-12-13 本文受国家自然科学基金项目(61070029)资助。

张建勋(1978—),男,博士生,讲师,主要研究方向为并行计算与多核缓存优化,E-mail:zhangjx@tjutcm.edu.cn;古志民(1964—),男,博士,教授,博士生导师,主要研究方向为并行计算及缓存优化,E-mail:zmgu@x263.net(通信作者)。

角色，这种结构称为 Leader/Follower 结构。Leader/Follower 结构将一个程序分成内存访问和计算两个模块，使访存和计算在执行上解耦[15]，从而加速程序的执行性能。

1.2 预计算与预执行思想

预计算和预执行可以看作是多线程结构的 Leader/Follower 结构。其主要思想是预取线程和原有线程同时运行。由于预取线程是原有线程的“精简版本”，它往往比原有线程运行得快，因此预取线程提前于主线程发出长延迟访存请求，并将预取结果服务于主线程以达到加速程序执行速度的目的。预取线程仅仅起到预取的作用，不修改主线程的体系结构状态。如果预取线程成功预先执行了长延迟访存指令，就能够达到加速效果；反之，如预取线程没有预先执行关键的延迟访存指令或执行结果有错时，就会降低主线程的执行速度，但不影响整个程序执行的正确性。因此，基于线程的预取机制可以在提高程序性能和保持执行结果正确性两个目标上解耦，从而大大放松了划分并行线程时的诸多限制，降低了系统实现的复杂度。

1.3 未来执行思想(Future Execution)[16]

未来执行也可以看作是一种 leader/follower 结构。未来执行的工作机制如图 1 所示：原始的没有经过改动的程序在第一个处理核上运行，当重排序缓冲区中的指令提交时，用其执行结果更新值预测器(Value Predictior)。值预测器通过特定算法预测出其在第 n 个循环之后要产生的值。如果预测的准确率高，那么就用一个 load 指令替代当前的指令，而对那些预测准确率低的指令就不用修改，保持不变。之后将处理后的指令流送到第二个处理核心上，直接将指令注入到流水线的派发阶段。指令的注入是以第一个核提交的顺序为准，从而保持原来程序的执行语义。由于指令是以译码之后的格式传到第二个处理核，因此在第二个核上可以将取指和译码过程旁路。一般在第二核上运行的程序会比第一核上运行的程序快 n 个循环。指令在第二个核上的执行按照正常模式执行即可，如果是 load 指令就用预测的地址去取数据，在执行过程中忽略掉异常的发生，从而避免因引入未来执行机制导致第二个核流水线的停顿。

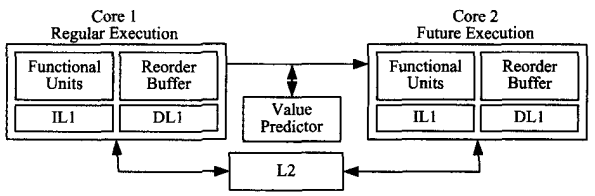

图 1 Future Execution 系统结构[31]

1.4 CMP 平台帮助线程预取思想

帮助线程预取技术实质上仍然是一种 leader/Follower 结构，帮助线程预取的目的地是 CMP 平台的最后一级共享缓存。如图 2 所示，CMP 体系结构的帮助线程思想[17] 主要是利用一个空闲核 CPU1，去运行一个帮助线程(精简版本的主线程)来帮助主线程进行数据预取，帮助线程在数据流和控制流上都不会对主线程造成干扰，只起到数据预取的作用。因为构造帮助线程时剔除了主线程的计算任务，只剩下了访存任务和必要的控制流，所以它可以和主线程按照计算和访存来进行分工，主线程负责计算，帮助线程负责访存，从而有效地隐藏访存延迟。根据主线程的计算量和访存量之间的比例，可以通过在帮助线程中设置一定的提前预取距离来提高帮助线程的性能。

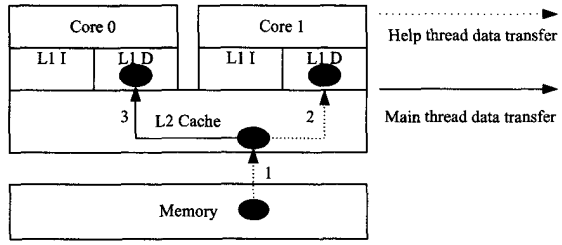

图 2 CMP 平台帮助线程数据预取原理

2 帮助线程预取技术分类

1)根据帮助线程的构造机制，帮助线程预取可以分为动态帮助线程预取技术和静态帮助线程预取技术两大类。动态帮助线程的产生和管理均由硬件自动生成，主要是由硬件利用指令 trace 缓冲区所收集的指令，根据寄存器依赖关系，反向生成一个指令序列来构造帮助线程。其优点在于帮助线程的构造相对于编译程序和程序开发人员来讲是透明的，无需程序开发人员改动源程序。但是，动态帮助线程技术需要更改现有的硬件体系结构设计，同时受限于硬件实现的复杂度和成本，指令 trace 缓冲区一般设计得很小，帮助线程构造的指令序列通常只有十几条，因此不能很好地理解程序的执行语义，从而降低预取的准确性。静态帮助线程技术需要事先对程序进行离线 profiling，然后由编译器在源码级或二进制级文件中显式地插入预取线程代码，其优点在于技术易于实现，程序分析的范围大大增加，可以更好地理解程序的语义信息和结构信息；缺点是不能观察到程序的动态行为，对帮助线程不能够实时进行控制。表 1 总结了动态帮助线程和静态帮助线程技术的主要优缺点。

表 1 不同帮助线程预取技术的比较

帮助线程预取分类	构造机制	优点	缺点
动态帮助线程技术	硬件	易于捕获程序动态行为，线程启动开销小	需更改硬件，实现困难，程序分析范围小
静态帮助线程技术	编译器	易实现，程序分析范围大，程序语义和结构理解好	软件开销大，不能理解程序动态行为

2)根据帮助线程实现的目标平台，可以将其分为基于 SMT 平台的帮助线程技术、基于 CMP 平台的帮助线程技术、基于单核平台的虚拟多线程技术。在 SMT 结构的处理器上实现帮助线程的主要性能瓶颈在于主线程和帮助线程共享相同的计算执行部件，如果帮助线程控制不当，会与主线程竞争计算资源，从而降低程序的性能。基于 VMT 技术的帮助线程不需要空闲的硬件上下文来支持帮助线程进行数据预取，而是通过操作系统的调度完成主线程与帮助线程之间的同步和切换工作。SMT 与 VMT 的区别在于，VMT 在时间上只有一个线程占有当前全部的硬件资源，而基于 SMT 的帮助线程却始终占有主线程未利用的硬件资源进行数据预取工作。基于 CMP 平台实现的帮助线程运行在不同的处理器物理核，计算资源竞争小，但是如果帮助线程的控制策略不当，会给内存带宽带来压力。

3 典型的帮助线程的构造和控制技术

帮助线程的关键技术主要包括长延迟指令的识别机制、

初始化机制、帮助线程的构造机制、触发机制以及控制机制。如何使帮助线程既能及时预取主线程所需要的数据，又能保证帮助线程运行不能超过主线程太远而造成预取的数据污染缓存，是 CMP 平台帮助线程预取技术实现的关键。

3.1 利用辅助硬件构造帮助线程片断(slice)

这类研究早期主要是针对 SMT 结构，采用反向数据流分析技术，通过辅助的硬件动态实现帮助线程的构造和触发。于考虑到硬件的成本，一般硬件实现的帮助线程的 slice 片断很小，只有少数 10 几条汇编指令，因此在帮助线程的控制策略上，不需要与主线程同步。硬件实现帮助线程的优势在于帮助线程输入的活跃变量(live-in variables)是通过寄存器拷贝实现的，从而克服了帮助线程与主线程的同步开销，其缺点是硬件实现成本较高。

3.1.1 动态投机预计算(Dynamic Speculative Precomputation)[4]

图 3 所示为实现 DSP 的微处理器结构，图中灰色部分为增加硬件部分，Slice Cache 为可选部件，主要是为了提高性能增设的缓存。新增硬件主要完成 3 个基本的功能：1)识别出长延迟 load 指令；2)针对这些长延迟访存指令构造线程执行片断 p-slice；3)触发和管理帮助线程的执行。

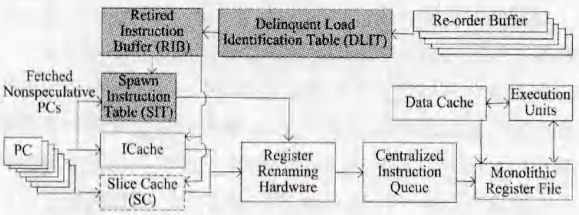

图 3 基于 SMT 微结构的 DSP 硬件实现[4]

长延迟访存指令的识别主要通过 DLIT 表来实现，DLIT 表项以先来先服务的方式分配给程序执行过程中发生 L2 缓存缺失的 load 指令。当程序提交指令数达到 128k 时，若仍然有一个 load 指令占有 DLIT 表项，并且该 load 指令至少被执行 100 次，每次执行平均超过 4 个或更多 CPU 时钟，则该 load 指令被标识为长延迟访存指令。

提交指令缓冲(Retired Instruction Buffer)主要用来保存已经提交指令的踪迹(trace)，以 FIFO 队列形式保存。RIB 中保存的信息包括每条指令的 PC 值、逻辑的源和目标寄存器号和一个标志位(用于标识该指令是否在 p-slice 片断中)。当识别出长延迟访存指令时，RIB 进入创建 slice 模式，长延迟访存指令本身自动加入到 p-slice 中，然后依次向上追踪该长延迟访存指令所依赖的指令，确定活跃变量集。

Slice 信息表 SIT 主要用于初始化帮助线程和管理 p-slice。在每个时钟周期内通过译码指令的地址查询 SIT 表，当检测到触发帮助线程的指令时，帮助线程在下一个时钟周期触发。SIT 表的初始化活跃变量功能以直接拷贝寄存器方式实现。帮助线程的取指可以从 Slice 缓存中进行，这样可以避免和主线程取指部件发生冲突。

3.1.2 动态预取线程 DPT[18]

图 4 为基于 CMP 结构的动态预取线程硬件实现，图中灰色部分为所添加硬件，其中 DPT 产生器主要完成长延迟访存指令的识别、动态预取线程的构造以及预取线程的触发等功能。设计的 Shadow Register 主要用来快速初始化预取线程的执行现场。

长延迟访存指令的识别和预取线程的构造策略与文献[4]相似，长延迟访存指令主要通过一个简单的 MISS 预测器(DLT 表)来实现，DLT 表是一个共 128 项以 PC 为索引的 5 位计数器，每当 L2 缓存 load 缺失时就增加相应的计数器，当计数器超过 31 时，即被识别为长延迟访存指令。与文献[4]不同的是预取线程的触发机制。长延迟访存指令本身即为预取线程的触发点，触发时间选定为该条指令提交时的时间点，其原因在于指令提交时寄存器的值已经准备好，而在指令派发时，寄存器的值还不确定。在 CMP 平台上的多个核心之间传送寄存器的上下文是一件十分耗时的事情，因此设计了 shadow register 来保留主线程执行核寄存器的状态，主线程核对 shadow 寄存器具有写的权限，而预取线程核对 shadow 寄存器只具有读的权限。

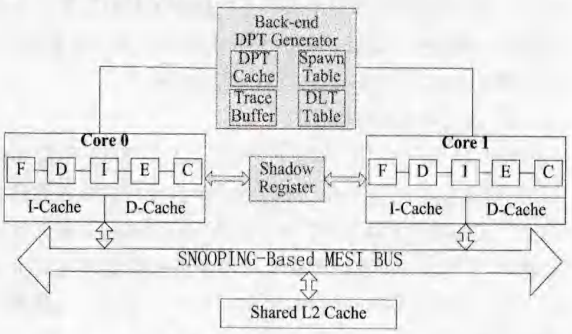

图 4 基于 CMP 结构的动态预取线程 DPT 硬件实现[33]

3.2 基于动态优化的实时动态帮助线程构造和控制策略

基于动态优化框架的帮助线程构造方法主要基于硬件性能监视计数器 PMC[19]和硬件优化事件[20]动态监测程序的运行，得到程序运行的一个热点踪迹(trace)，然后根据得到的程序 trace 构造帮助线程。Jiwei Lu 等人[19]基于 CMP 平台的硬件性能计数器实现了动态优化框架，其实现的帮助线程管理控制策略主要是在主线程和帮助线程中分别设置计数器，当计数器超过预先设置的阈值时，主线程和帮助线程同步一次。Weifeng Zhang 等人[20]提出的 Trident 是一个软硬件结合的动态优化框架，其在硬件的支持下支持低开销的优化预计算线程，因帮助线程的代码数量较少，在帮助线程实现过程中无需同步。Yangchun Luo 等人[21]针对当前出现的各种 TLS 执行模型提出了一个动态性能调节框架，它可以利用现代处理器的 PMC 性能计数器，动态收集程序的执行信息，然后对投机线程的性能做出预测，从而进行实时动态性能调节，其动态调节的思想可以指导用来优化帮助线程的参数选择。

3.3 基于编译技术和手动的帮助线程构造和控制策略

基于编译技术的帮助线程构造策略主要依赖于静态的离线分析信息(offline profiling)，编译程序根据 profiling 信息，在原始程序中插入帮助线程触发点，以及帮助线程的控制代码。目前基于 CMP 平台上帮助线程的实现大多是基于编译技术生成帮助线程，帮助线程的控制策略主要通过主线程和帮助线程之间的同步来解决。一方面保证帮助线程能够及时地预取主线程所需要的数据，另一方面保证帮助线程不会落后或超前于主线程太长的距离，从而替换掉主线程所需的有用数据，造成多核平台的最后一级缓存污染。因此关于主线程和帮助线程之间的同步技术研究是当前一个比较活跃的研究领域，目前提出的同步方法主要有基于循环的同步[22]、基

于采样的同步机制[22]、双计数器同步[7]以及"PV"[23,24]同步方法。

3.3.1 二进制级编译实现

Liao 等人[6]首次在二进制代码级别实现了预执行线程的动态编译生成技术。帮助线程的触发方式有两种:基本触发方式和链式触发方式。基本触发方式是主线程在某个固定的触发指令点触发帮助线程;而链式触发方式是指一个帮助线程还可以触发另外一个帮助线程。Post-pass 工具通过利用缓存 profiling 信息来确定长延迟访存指令,通过使用程序控制流(Control Flow Graph)信息来生成 Slice 片断。针对帮助线程的链式触发方式,线程之间同步只进行一次,即在触发点将所有活跃变量传递给帮助线程;针对基本触发方式,如果生成 Slice 包括一个循环体,那么主线程在每一次循环迭代时触发一个帮助线程,即帮助线程每次所做的预取工作只是循环体的一次迭代。无疑这种同步方式增加了同步次数,当同步开销很大时,会严重影响帮助线程的性能。

3.3.2 源代码级编译实现

Kim 等人[7]利用 Unravel[25]切片工具和斯坦福大学 SUIF 编译框架在源代码一级完成了预执行帮助线程的自动构造。其编译器工具主要针对 3 种情况对预执行线程进行初始化:1)只生成一个预执行线程,2)针对 DoAll 循环同时生成多个并行执行的预执行线程,3)由预执行线程派生出预执行线程的情况。

在该方案中,主线程维护一个全局的计数器,当主线程执行完一个迭代之后,执行 V 操作,全局计数器加 1。而在每个预执行线程中维护一个局部的计数器,当执行完一个迭代之后,执行 P 操作,局部计数器加 1 并且和全局计数器进行比较,只有当两个计数器之间的差异大于 PD(预取距离)时,预执行线程才继续运行,否则预执行线程进入阻塞状态,进行等待。

Kim 等人[22]首次将帮助线程的实现移植到真实的 SMT 同时多线程平台,并使用编译器利用 profiling 信息实现了帮助线程的构造。其实现的帮助线程有两种情况,一种是基于循环的触发,即在循环的入口处触发帮助线程,在此期间主线程和帮助线程不同步,只在触发时同步一次。另外一种是基于采样的触发,帮助线程每隔一定的循环数触发一次,两次触发间隔称为采样周期。每次触发帮助线程后,要么执行与采样周期一样的循环数,要么是执行到循环终止。实质上,此处的采样周期就是一个固定的循环数,即帮助线程每隔固定循环数和主线程同步一次,因此称其为基于循环计数的同步方法。Yonghong Song 等人[26]在 SUN SPARC 平台上基于编译实现了帮助线程的构造方法,其构造方法基于对帮助线程的获益判断构造,具体帮助线程的控制策略没有提及。

Jaejin Lee 等人[23,24]在多核平台上手动实现了帮助线程的构造,并提出了一种"PV"帮助线程控制方法。该方法主要是在主线程和帮助线程中分别维护一个同步计数器(Loop_SYNC_Interval),在每个同步间隔,主线程执行"V"操作,表示主线程已经消费掉一个块的数据。在帮助线程到达同步间隔后,首先判断它所生产的"货物"是不是超过了一个阈值(MAX_DIST),如果超过,那么表明帮助线程落后于主线程,帮助线程需要做的操作是同步当前变量,追上主线程,并重新初始化阈值,如果没有超过,那么帮助线程简单地执行"P"操作,表示此时帮助线程仍然快于主线程。因此,在"PV"方案中,帮助线程能够领先主线程的最大距离是 Loop_SYNC_Interval * MAX_DIST 个循环数。该方案存在的主要问题是当主线程计算工作量很小时,帮助线程不能够保证始终领导主线程的执行,这时就会产生频繁的同步操作,从而带来性能的下降。

3.4 综合比较

表 2 是目前典型的帮助线程预取技术的综合比较,除硬件实现外,目前关于帮助线程预取的控制策略大多集中于主线程和帮助线程的同步机制。之前的大部分线程控制策略总是基于帮助线程是一个精简版本的主线程,其执行速度快于主线程的假设设计的,但是经过实验表明,帮助线程在执行过程中的执行步调并不总是优先于主线程。其本质原因在于一方面是主线程的计算工作量比较小,另外一方面在于帮助线程在预取数据的同时,大量的 LLC 缺失也会造成帮助线程停顿不前,尽管在帮助线程中针对经常发生缺失的指令用到的是非阻塞预取指令,但是在链式数据结构中,对链式数据结构本身的遍历往往会发生 LLC 缺失的情况,因此帮助线程常常会落后于主线程。针对这种情况,Zhimin Gu 课题组[27-32]基于 CMP 平台提出了 KPB(sKip-Push-Block)多参数帮助线程控制策略。该策略在考虑主线程计算工作量的前提下,通过调整参数 K-P-B,使得帮助线程能够适应不同计算量大小的主线程情况,对帮助线程做出同步或不同步的控制。目前 K-P-B 参数是通过实验来确定的。这和基于编译技术的静态生成帮助线程的方法存在同样的缺点,即不能根据程序运行时的动态阶段行为进行调整,比如同步间隔、参数选择和控制等等。笔者在课题组工作的基础上,提出了一个帮助线程预取控制框架(Helper thread Prefetching Control Framework)[33],即通过利用处理器硬件性能计数器 PMC 采集帮助线程执行过程中主线程性能的变化,通过参数生成算法实时对帮助线程的参数进行控制,从而克服帮助线程静态参数选择所存在的问题。另外,将 HPCF 应用于非规则大数据分析应用程序中常用到的中间介质核心性算法,取得了较好的性能加速比。

表 2 典型帮助线程预取技术比较

帮助线程预取技术	长延迟指令识别	初始化机制	构造机制	触发机制	控制机制
硬件动态 HT 预取技术	MISS 预测器	活跃变量直接拷贝至寄存器	反向指令 Trace	长延迟指令触发	Slice 较小无需与主线程同步
实时动态优化 HT 技术	动态监测 PMC	提前于主线程启动	反向指令 Trace	程序热点区前触发	全局变量同步
基于编译的 HT 技术	离线 profiling	由主线程启动	二进制代码级由 CFG 生成 slice	程序热点循环前触发	变量同步

结束语 帮助线程预取技术克服了传统预取技术针对非规则访存行为的预测困难问题,通过利用 CMP 平台的最后一级共享缓存架构来达到线程级数据预取的目的。当前,CMP 平台作为提高单线程应用性能的一种手段,帮助线程技术已经突显其重要性。本文对帮助线程预取技术的实现方法和控制策略进行了综述,并深入分析和对比了各种帮助线程预取方法。

就目前来看,当前国内外关于帮助线程技术的研究存在

以下不足：

1)帮助线程静态参数控制策略不能适应程序的动态阶段缓存行为

“PV”控制策略和“KPB”控制策略均属于静态控制策略，参数的选择通过枚举实验来完成。而在程序执行过程中预定参数不会发生变化。以 MST 程序为例，对于不同的参数集，程序的性能不同，最大的差距在 30%左右[30]。

因此，静态参数控制策略存在的问题主要是帮助线程在程序执行期间的收益难以达到最大化，不能够根据程序的动态阶段行为[21]做出调节。预先获得的 profiling 信息，是程序的整个执行过程累积的结果，例如，发生 LLC 后缺失的个数是整个循环执行完成以后得到的结果。而在程序执行过程中，被确认为长延迟的指令或语句并不总是发生 LLC 缺失，因此有的时间段帮助线程会落后于主线程，此时就不会带来获益或获益较小，因此在编译时，确定好的静态触发帮助线程并不总是有效的，帮助线程参数的有效动态调整能够达到较好的预取效果，因此参数动态调整的算法需要进一步进行研究。

2)帮助线程的性能依赖于特定的硬件系统结构

因为帮助线程所带来的性能提升来源于共享的硬件缓存资源，当相应的硬件缓存资源配置发生变化时，程序的执行行为也会发生变化。而与帮助线程的控制相关的一些阈值，如同步间隔就需要重新确定。之前的研究工作表明，在不同的平台 Q6600 和 Core i7，帮助线程的性能不同，并且对帮助线程的参数要求也不相同[30]。

3)数据输入集不同时对帮助线程的参数控制策略要求不同

帮助线程所带来的性能提升取决于数据输入集的不同。例如文献[22]利用 Intel 的同时多线程处理器上实现的基于循环的帮助线程，在选择循环时，首先从包含长延迟指令或语句的循环开始搜索，当循环次数超过一定阈值时，选定该循环作为帮助线程构造的基础代码。而这个循环次数阈值的选择会成为帮助线程是否带来获益的关键。如果选择的阈值过高，那么可能会丢失一些帮助线程带来性能提升的机会；如果阈值过低，帮助线程的获益并不会抵销其本身的开销。而当数据输入集变化时，这些程序的动态执行行为也会发生变化，也就是说，帮助线程在数据输入集发生变化的情况下，可能并不会带来性能的提升。

基于以上分析，最后指出下一步的工作重点是要解决帮助线程的实时动态控制和管理问题，降低帮助线程的参数选择和优化开销，从而保证帮助线程的预取质量。

参 考 文 献

[1] Bryant R E. Data-Intensive Supercomputing: The case for DISC [EB/OL]. http://www.cs.cmu.edu/~bryant, 2012-12-13

[2] 谭光明. 非规则计算中的局部性和并行性[D]. 北京：中国科学院计算技术研究所，2008

[3] Annavaram M, Patel J M, Davidson E S. Data prefetching by dependence graph pre-computation[A]// Proceedings of the 28th Annual International Symposium on Computer Architecture (Goteborg, Sweden), 2001[C]. New York: ACM, 2001: 52-61

[4] Collins J D, Tullsen D M, Wang H, et al. Dynamic speculative precomputation[A] // Proceedings of the 34th International Symposium on Microarchitecture (Austin, Tex.), 2001[C]. New York: ACM, 2001: 306-317

[5] Collins J D, Wang H, Tullsen D M, et al. Speculative precomputation: Long-range prefetching of delinquent loads[A] // Proceedings of the 28th Annual International Symposium on Computer Architecture (Goteborg, Sweden), 2001[C]. New York: ACM, 2001: 14-25

[6] Liao S S W, Wang P H, Wang H, et al. Post-pass binary adaptation for software-based speculative precomputation[A] // Proceedings of the ACM SIGPLANConference on Programming Language Design and Implementation (Berlin, Germany), 2002 [C]. New York: ACM, 2002: 117-128

[7] Kim D , Yeung D. Design and evaluation of compiler algorithms for pre-execution[A] // Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (San Jose, Calif.), 2002[C]. New York: ACM, 2002: 159-170

[8] Luk C-K. Tolerating Memory Latency through software-controlled pre-execution in simultaneous multithreading processors [A]//Proceedings of the 28th Annual International Symposium on Computer Architecture (Goteborg, Sweden), 2001[C]. New York: ACM, 2001: 40-51

[9] Moshovos A, Pnevmatikatos D N, Baniasadi A. Slice-processors: An implementation of operation-based prediction[A] // Proceedings of the International Conference on Supercomputing (Sorrento, Italy), 2001[C]. New York: ACM, 2001: 321-334

[10] Roth A, Sohi G S. Speculative data-driven multithreading [A]// Proceedings of the 7th International Conference on High Performance Computer Architecture (Monterrey, Mexico), 2001 [C]. Los Alamitos, Calif: IEEE Computer Society Press, 2001: 191-202

[11] Roth A, Sohi G S. A quantitative framework for automated pre-execution thread selection[A]//Proceedings of the 35th Annual International Symposium on Microarchitecture (Istanbul, Turkey), 2002[C]. New York: ACM, 2002: 430-441

[12] Zilles C B, Sohi G. Execution-based prediction using speculative slices[C]// Proceedings of the 28th Annual International Symposium on Computer Architecture (Goteborg, Sweden), 2001. New York: ACM, 2001: 2-13

[13] Collins J D, Tullsen D M, Wang Hong, et al. Dynamic Speculative Precomputation[A]//Proceedings of the 34th International Symposium on Microarchitecture(MICRO'01), 2001[C]. New York: IEEE, 2001: 306-317

[14] Dubois M, Song Y. Assisted Execution [R]. University of Southern California, October 1998

[15] Smith J E. Decoupled access/ execute computer architectures [A] // Proc. of the 9th Int. Symp. on Comp. Arch. (ISCA-9), 1982[C]. Los Alamitos: IEEE, 1982: 112-119

[16] Ganusov I, Burtscher M. Future Execution: A Prefetching Mechanism that Uses Multiple Cores to Speed up Single Threads[J]. ACM Transactions on Architecture and Code Optimization, 2006, 4(3): 424-449

(下转第 39 页)

行为和所具有的物理特性两方面平衡考虑。这种平衡机制对所有的节点在判断是否是搭便车节点时，更公平合理。同时对于搭便车者的处罚机制，是通过限制其下载资源的速度来实现的，从而可以延长其等待下载资源的在线时长。从仿真结果来看，基于平衡机制的算法可以在一定程度上有效地抑制搭便车行为，提高成功下载率以及系统的稳定性。

参 考 文 献

[1] Steinmetz R, Wehrle K. P2P 系统及其应用[M]. 王玲芳，陈焱，译. 北京：机械工业出版社，2008：1-22

[2] 许晓东，邹宝军，朱士瑞. 信任模型中搭便车节点的抑制[J]. 计算机科学，2012，39(3)：88-92

[3] 康江，房鼎益，陈晓江. 一种无结构 P2P 网络中对抗 Free-rider 的新方法[J]. 小型微型计算机系统，2010，31(8)：1538-1541

[4] Michal F, Christos P, John C, et al. Free Riding and Whitewashing in Peer to Peer Systems[J]. IEEE Journal on Selected Areas in Communications, 2006, 24(5): 1010-1019

[5] Xiong L, Liu L. PeerTrust: supporting reputation-based trust for peer-to-peer electronic communities[J]. IEEE Transactions on Knowledge and Data Engineering, 2004, 16(7): 843-857

[6] 余一娇，金海. 对等网络中的搭便车行为分析与抑制机制综述[J]. 计算机学报，2008，31(1)：1-15

[7] Hua J-S, Huang S-M, Yen D C, et al. A dynamic game theory approach to solve the free riding problem in the peer-to-peer networks[J]. Journal of Simulation, 2012(6): 43-55

[8] Lakshmish R, Liu L. Free Riding: A New Challenge to Peer-to-Peer File Sharing Systems[C]// Proceedings of The 36th Hawaii International Conference on System Sciences. IEEE Computer Society, 2003: 1-10

(上接第 23 页)

[17] Byna S, Chen Yong, Sun Xian-he. A Taxonomy of Data Prefetching Mechanisms [J]. Journal of Computer Science and Technology, 2009, 24(3): 405-417

[18] Rui Hou, Zhang Long-bing, Hu Wei-wu. Accelerating sequential programs on Chip Multiprocessors via Dynamic Prefetching Thread[J]. Microprocesors and Microsystems, 2007 (31): 200-211

[19] Lu Ji-wei, Das A, Hsu W-C, et al. Dynamic Helper Threaded Prefetching on the Sun UltraSparc CMP Processor[A]// Proc. 38th Ann. IEEE/ACM Int'l Symp. Microarchitecture(MICRO '05), 2005[C]. New York: ACM, 2005: 93-104

[20] Zhang Wei-feng, Calder B, Tullsen D M. A Self-Repairing Prefetcher in an Event-Driven Dynamic Optimization Framework [A]// IEEE Proceedings of the International Symposium on Code Generation and Optimization(CGO'06)[C]. 2006

[21] Luo Yang-chun, Packirisamy V, Hsu Wei-Chung, et al. Dynamic Performance Tuning for Speculative Threads[A]// Proc. of the 36th Int. Symp. on Comp. Arch. (ISCA-09)[C]. 2009

[22] Kim D, Liao S S-W, Wang P H, et al. Physical Experimentation with Prefetching Helper Threads on Intel's Hyper-Threaded Processors[A]// Proceedings of the International Symposium on Code Generation and Optimization[C]. Mar. 2004

[23] Jung C, Lim D, Lee Jaejin, et al. Helper Thread Prefetching for Loosely-Coupled Multiprocessor Systems[A]// IPDPS[C]. 2006

[24] Lee J, Jung C, Lim D, et al. Prefetching with Helper Threads for Loosely Coupled Multiprocessor Systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2009, 20(9): 1309-1324

[25] Lyle J, Wallace D, Graham J, et al. Unravel: A CASE Tool to Assist Evaluation of High Integrity Software[EB/OL]. http://hissa.ncsl.nist.gov/publications/nistir5691/vol1/, 2012-08-15

[26] Song Yong-hong, Kalogeropulos S, Tirumalai P. Design and Implementation of A Compiler Framework for Helper Threading on Multi-Core Processors[A]// Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques (PACT'05)[C]. 2005

[27] Zhang Jian-xun, Gu Zhi-min, et al. Performance Evaluation of data-push Thread on Commercial CMP Platform[A]// Proceedings of the International Network Computing Conference[C]. Korea, 2010

[28] Gu Zhi-min, Zheng Ning-han, Zhang Yi, et al. The Stable Conditions of a Task-Pair with Helper-Thread in CMP[A]// Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, 2009. Las Vegas, Nevada, USA: IEEE, 2009: 125-130

[29] Gu Z, Fu Y, Zheng N, et al. Improving Performance of the Irregular Data Intensive Application with small workload for CMPs [A]// International Conference on Parallel Processing Workshops[C]. Taiwan, China, 2011

[30] Zhang J, Gu Z. Exposing the Shared Cache Behavior of Helper Thread on Commercial CMP Platforms[A]// 11th International Symposium on Pervasive Systems, Algorithms, and Networks. Dalian, China (I-SPAN'2011)[C]. 2011

[31] Huang Y, Tang J, et al. The Performance Optimization of Threaded Prefetching for Linked Data Structures[J]. Intern. Journal of Parallel Programming, 2011, 4(20): 141-163

[32] Huang Y, Gu Z, et al. Reducing cache pollution of threaded prefetching by controlling prefetch distance[A]// Proc. IPDPS[C]. 2012

[33] Zhang J, Gu Z, et al. Solving Prameter Selection Problem of Helper Thread Prefetching via Realtime Hardware Performance Monitoring[A]// 13th International Conference on Parallel and Distrbuted Computing, Applications and Technologies (PDCAT 2012)[C]. 2012