第 35 卷第 8 期 通 信 学 报 Vol.35 No. 8
2014 年 8 月 Journal on Communications August 2014
doi:10.3969/j.issn.1000-436x.2014.08.017

面向非规则大数据分析应用的多核帮助线程预取方法

张建勋[1,2]，古志民[1]，胡潇涵[1]，蔡旻[1]
（1. 北京理工大学 计算机学院，北京 100081；2. 天津中医药大学 网络中心，天津 300193）

摘　要：大数据分析应用往往采用基于大型稀疏图的遍历算法，其主要特点是非规则数据密集访存。以频繁使用的具有大型稀疏图遍历特征的介度中心算法为例，提出一种基于帮助线程的多参数预取控制模型和参数优化方法，从而达到提高非规则数据密集程序性能的目的。在商用多核平台 Q6600 和 I7 上运用该方法后，介度中心算法在不同规模输入下平均性能加速比分别为 1.20 和 1.11。实验结果表明，帮助线程预取能够有效提升该类非规则应用程序的性能。

关键词：帮助线程预取；非规则数据密集应用；介度中心性
中图分类号：TP311　　**文献标识码**：A　　**文章编号**：1000-436X(2014)08-0137-10

Multi-core helper thread prefetching for irregular data intensive applications

ZHANG Jian-xun[1,2] GU Zhi-min[1] HU Xiao-han[1] CAI Min[1]
(1.School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China;
2. Network Center, Tianjin University of Traditional Chinese Medicine, Tianjin 300193, China)

Abstract: Big data analysis applications often use sparse graph traversal algorithm which characterized by irregular data intensive memory access. For improving performance of memory access in sparse graph traversal algorithm, helper thread prefetching could convert discontinuous locality into continuous-instant spatial-temporal locality effectively by using the shared last level cache of chip multi-processor platforms. Betweenness centrality algorithm was used as a case study, the multi-parameter prefetching model of helper thread and optimized instances were presented and evaluated on commercial CMP platforms Q6600 and I7, the average speedup of betweenness centrality algorithm at different input scale is 1.20 and 1.11 respectively. The experiment results show that helper thread prefetching can improve the performance of irregular applications effectively.

Key words: helper thread prefetching; irregular data intensive applications; betweenness centrality

1　引言

随着社交网络、微博等互联网应用的不断普及，海量非结构化数据将在未来 10 年里增长到现有数据量的 44 倍[1]。这些大数据以其多样性（variety）、海量性（volume）和快变性（velocity）的 3V 特性给数据分析应用领域带来严峻挑战。如何利用现有多核平台实时挖掘大数据所蕴含的价值（value）、提高系统性能是大数据时代亟待解决的难题之一。非规则大数据分析应用程序通常依赖于图、树或者链表等存储结构来实现，如计算介度中心（betweenness centrality）算法[2]等。这类应用访存行为具有动态性、非规则性等特点，访存模式往往不能在静态编译时精确预测，从而导致基于时空局部性的传统数据预取技术无法发挥作用。

针对非规则大数据分析和帮助线程预取技术的特点，为提升非规则数据密集应用的性能，研究帮助线程预取技术在多核平台上的缓存优化设

收稿日期：2013-05-02；修回日期：2013-07-22
基金项目：国家自然科学基金资助项目(61070029, 61370062)
Foundation Item: The National Natural Science Foundation of China (61070029, 61370062)

计，提出了一个通用的帮助线程预取构造和控制模型，并以广泛应用的介度中心算法为例，在商用多核平台实现了帮助线程的构造、分析和实验验证。实验结果表明，针对 SSCA2[3]基准测试程序，在不同规模输入下，帮助线程能够有效提高 BC 算法的执行性能，在 Q6600 平台， BC 算法在不同规模输入下平均性能加速比为 1.20，最高性能加速比为 1.25。在 I7 平台，BC 算法在不同规模输入下平均性能加速比为 1.11，最高性能加速比为 1.15。

2 相关工作

近年，提升非规则数据密集应用在多核平台上的性能已成为数据密集计算领域研究热点之一。Luk[4]指出，由于传统硬件/软件预取机制在预测准确性和及时性方面均存在缺陷，非规则访存模式只能通过执行代码本身来进行预测。目前，基于预执行思想[5]的帮助线程预取是解决这一难题的重要技术，帮助线程通过提前预取主线程所需的数据到多核最后一级共享缓存（LLC），以减少主线程 LLC 缺失，达到实现延迟隐藏的目的。帮助线程预取技术主要有硬件和软件 2 种实现方案，其中软件实现方案[6~9]具有无需改动硬件设计、实验评测可以在真实机器上进行、程序语义分析范围广等优势。例如，Zhou 等人[10]将预执行帮助线程用于提高关系型数据库的性能，使数据库吞吐率有 30%~70%的性能提升，Huang 等人[11]利用帮助线程预取技术来提高链表指针密集型应用的性能，Akos 等人[12]利用预执行帮助线程加速基于散列函数实现的数据密集应用的性能等。

计算介度中心算法是非规则大数据网络分析问题中具有代表性的一个算法，算法核心是图理论中典型的广度优先搜索（BFS）算法，其代表着一大类具有非规则访存行为的应用。然而，当前利用帮助线程预取技术提升 BC 算法性能的研究较少，大多研究是从 BC 算法的并行设计实现来提高其性能[13~15]。Tu 等人[16]在商用多核架构上对 BC 算法上的特征进行了深入细致的评测和分析，最后得出 BC 算法具有非规则访存、数据密集和非结构化并行 3 个特征。Tan 等人[17]在 IBM Cyclops64 模拟器上对 BC 算法进行了性能分析，通过利用系统结构支持的细粒度同步机制和多线程部件实现了一个细粒度并行 BC 算法。Tan 等人[18]提出一个延迟容忍的并行计算模型，通过在用户级将计算和访存操作分离，将并行线程划分为计算线程和访存线程（帮助线程）2 个组，通过计算线程和帮助线程的流水调度操作实现细粒度的并行执行。与 Tan[18]工作相比，本文帮助线程是基于商用多核平台，通常不需对计算任务进行再分解，而是直接对其热点模块构造帮助线程，并通过适当的控制策略来保证帮助线程的预取及时性和预取覆盖率。

3 介度中心性（BC）算法简介

介度中心性是大规模网络分析中常用的量化指标，该指标主要考察有向加权图中一个点落在其他任意两点最短路径中的程度，中间度越大，说明这个点越处于信息枢纽的位置，对信息的控制能力也就越强。下面简要介绍 Brandes[2]改进的快速 BC 算法。

3.1 BC 算法概念及原理

定义 1 顶点介度中心性 $BC(v)$

给定图 $G=(V, E)$，V 和 E 分别是顶点和边的集合。σ_{st} 表示顶点 s 和顶点 t 之间的最短路径的数量。$\sigma_{st(v)}$ 表示顶点 s 和顶点 t 之间最短路径中经过顶点 v 的路径的个数。

顶点 v 的介度中心性定义为

$$BC(v)=\sum_{s\neq v\neq t\in V}\frac{\sigma_{st(v)}}{\sigma_{st}} \tag{1}$$

由 $BC(v)$的定义可以看出，顶点 v 的介度中心性就是顶点 v 落在图中任意两顶点之间最短路径上的次数。Brandes 改进后的 BC 算法引入特定顶点依赖的概念 $\delta_s(v)$。

定义 2 顶点依赖性 $\delta_s(v)$

$$\delta_s(v)=\sum_{t\in V}\delta_{st}(v) \tag{2}$$

$\delta_s(v)$ 表示顶点 $s \ni V$ 对于顶点 $v \ni V$ 的依赖性。式(2)中，$\delta_{st}(v)=\frac{\sigma_{st(v)}}{\sigma_{st}}$，表示顶点 s 和顶点 t 之间通过顶点 v 的最短路径数占顶点 s 和 t 之间最短路径总数的比例，也即顶点 v 对顶点对(s, t)的依赖。顶点对(s,t)表示从顶点 s 出发到达图 G 中任意顶点 t，并且经过顶点 v 的最短路径的首尾顶点。顶点 v 对所有顶点对(s,t)的依赖性总和，即为顶点 s 对于顶点 v 的依赖性。

根据式(1)和式(2)，顶点 v 的介度中心性可以表

示为 $BC(v)=\sum_{s\neq v\in V}\delta_s(v)$。

Brandes 通过扩展针对有权图的单源点最短路径 Dijkstra 算法和针对无权图的 BFS 算法来计算顶点的依赖性。在最短路径算法中，如果顶点 v 在顶点 s 和顶点 t 之间的最短路径上，那么当且仅当满足条件 $d(s,t)=d(s,v)+d(v,t)$，$d(s,t)$ 表示从顶点 s 到顶点 t 的最短路径长度。Brandes 定义 $P[s,v]$ 表示从顶点 s 出发经过顶点 v 的最短路径中顶点 v 的前驱节点集合。当算法每次扫描到边 (u,v) 时，如果该条边满足条件 $d(s,v)=d(s,u)+d(u,v)$，那么顶点 u 加入到顶点 v 的前驱集合 $P[s,v]$。此时，关系 $\sigma_{sv}=\sum_{u\in P[s,v]}\sigma_{su}$ 成立。通过 Dijkstra 的单源点最短路径算法（有权图）或 BFS 算法（无权图）可以很容易地计算顶点 v 的前驱集合 $P[s,v]$。

Brandes 改进后的 BC 算法关键在于发现了节点依赖性的计算遵循式(3)所示的迭代规律，从而避免了其他 BC 算法实现的求和过程，在算法的回溯阶段，前驱节点的部分 *BC* 值可以根据其后继节点累加而成。

$$\delta_s(v)=\sum_{w:v\in P[s,w]}\frac{\sigma_{sv}}{\sigma_{sw}}(1+\delta_s(w)) \tag{3}$$

式(3)表示顶点 $s\ni V$ 对图中任意顶点 $v\ni V$ 的依赖所满足的迭代规律。Brandes 改进后的 BC 算法时间复杂度由 $O(n^3)$降低到 $O(nm+n^2\log n)$。

3.2　BC 算法访存特征分析

与传统科学计算相比，BC 算法具有 3 个典型特征。1) 非规则内存访问模式。这使得传统基于时空局部性优化技术（如硬件预取技术、推测技术等）失效，因此需要研究新的预取与推测方法。2) 计算工作量小。该应用属于数据密集型应用，与访存时间相比，计算量相对较小，这使得程序性能对内存带宽和访存延迟比较敏感。3）非结构化并行。程序的非结构化并行特征是程序设计和程序分析所面临的重要难题，BC 算法的并行程序往往需要综合权衡并行粒度以及软硬件协同等问题。

4　帮助线程预取方法

帮助线程预取技术本质是一种 leader/follow 结构[5]的软件实现，在片上多核处理器上，利用一个空闲核运行一个帮助线程（精简版的主线程），主要用来实现主线程所需数据的预取。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。由于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，因而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，以达到隐藏访存时间目的。

4.1　帮助线程预取模型

首先，建立基于非规则访存应用的帮助线程预取模型如下。

设 C_m 为多核平台上主线程所需的执行核，C_h 为帮助线程所需的执行核，BC 算法程序 $\Sigma=\{s_1, s_2,\cdots,s_n\}$ 由多个指令序列 s_i 组成，$i=1,2,\cdots,n$。$Rv(s_i)$ 是指令序列 s_i 中只读变量的集合，$Wv(s_i)$ 是指令集合 s_i 中写变量的集合。

定义 3　并行执行：$s_i \parallel s_j$

对于 s_i，$s_j\in\Sigma$，如果 2 个指令序列所操作的变量集之间不存在读写相关和写写相关，即满足条件 $Rv(s_i)\cap Wv(s_j)=\Phi$、$Rv(s_j)\cap Wv(s_i)=\Phi$ 和 $Wv(s_i)\cap Wv(s_j)=\Phi$，称 s_i 可并行于 s_j，记作 $s_i\parallel s_j$。也就是 s_i 和 s_j 能够并行，说明 s_i 和 s_j 的执行顺序无关。

如果一个热点程序块 $\Gamma=(\gamma_1,\gamma_2,\cdots,\gamma_n)$ 是由多个指令序列 γ_i 组成，并且指令序列满足条件 $Rv(\gamma_i)\neq Rv(\gamma_{i+1})$ 和 $Wv(\gamma_i)\neq Wv(\gamma_{i+1})$，主线程执行核 C_m 按序执行 $\gamma_1,\gamma_2,\cdots\gamma_n$，那么，帮助线程执行核 C_h 执行 Γ 的精简版的程序块 $\Gamma'=(h_1h_2\cdots h_n)$，并且让 Γ' 和 Γ 满足定义 3 的条件，则有 $\Gamma'\parallel\Gamma$。其中，帮助线程预取控制的难点在于如何保证主线程执行核 C_m 正在执行 γ_j 的情况下，帮助线程执行核 C_h 已经执行完成指令序列 h_j，即完成了程序序列 γ_i 部分的数据预取工作。

这样，为了保证帮助线程预取的准确性，帮助线程触发点、预取工作量大小等参数的选择就成为关键因素。例如传统帮助线程预取[6~9]在触发执行精简版本的程序块 Γ' 后，预取工作量即为热点函数中的所有长延迟指令，而当主线程计算工作量不足的情况下，帮助线程会落后于主线程的执行步调，预取数据会污染共享缓存，造成预取性能下降。为此，提出了一个 KPB 多参数帮助线程预取框架，用参数 *sKip* 来调节帮助线程预取的触发点，用参数 *Prefetch size* 来调节预取工作量大小，同时也可用参数 *Block* 调节帮助线程与主线程的同步频次。例如，一个 KPB 帮助线程预取的工作场景如图 1 所示。

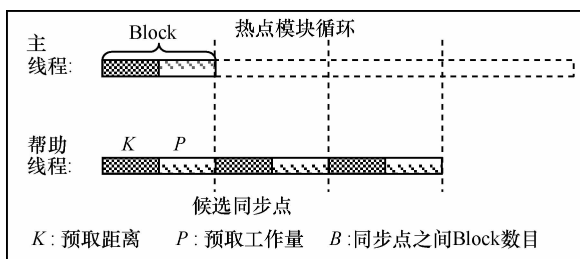


图 1 KPB 多参数帮助线程预取工作场景

程序热点模块通常是由一个频繁执行长延迟指令的循环结构组成，在图 1 中将热点模块循环按照循环数被分成等长的 Block,称之为循环块。帮助线程触发后，首先在每个循环块中，帮助线程首先跳跃 *K* 个热点循环，然后触发预取指令，预取的工作量大小为 *P* 个热点循环。通常情况下，帮助线程与主线程的同步是每个 Block 同步一次，当同步开销大时，可设计为多个 Block 同步一次。KPB 帮助线程预取控制框架的核心思想就是将长延迟指令的访存操作并行化，即由主线程和帮助线程并行分别承担一定的长延迟指令的缓存缺失操作，而不是由帮助线程预取全部数据。KPB 帮助线程预取框架的好处在于能够确保帮助线程不落后于主线程，并且通过灵活地调整参数 *K*，*P* 和 *B* 能够适应主线程具有不同计算工作量大小的应用。

4.2 BC 算法热点分析

对 BC 算法进行热点分析，确定其中的热点模块。

BC 算法的主体主要包括 2 个阶段，即 BFS 搜索阶段和回溯（backtrack）阶段，以 SSCA2 中实现的 BC 算法为分析对象。利用 Intel Vtune Amplifier 作为 Profiling 工具通过采样处理器硬件性能监视器发现程序的热点。Amplifier 能够记录程序运行过程中发生的各种硬件性能事件（如 CPU CLOCKS, LLC 缺失等信息）在源代码中发生的位置和数量，那些占用大量处理器时钟的模块或函数即为程序的热点。

表 1 所示为 SSCA2 测试程序中 BC 算法程序的 Profiling 结果。从表 1 可以看出，BC 算法程序的处理时间占整个 SSCA2 测试程序总时间的 91.68%，其中 BC 算法的 BFS 模块和 Backtrack 模块运行时间分别占整个 SSCA2 总时间的 64.15%和 24.71%。而 2 个模块相应的 LLC 缺失数分别占整个 SSCA2 测试程序的 61.31%和 37.33%。LLC 缺失表示程序所需数据没有在最后一级共享缓存中，此时需要大量的访存操作完成数据的缓存加载，从而造成处理器流水线的停顿。Profiling 结果表明，SSCA2 程序中约 99%以上的 LLC 缺失由 BFS 和 Backtrack 模块造成，相应的运行时间占整个程序的 90%以上，因此 BFS 和 Backtrack 模块就成为帮助线程预取的候选热点模块。

定义 4 热点模块 $f(A)$：对于应用程序 A 中的任意模块 $f(A)$，存在阈值 ε_1 和 ε_2 分别满足 $Cycle(f(A))/Cycle(A) \geqslant \varepsilon_1$ 和 $LLC_Miss(f(A))/LLC_MISS(A) \geqslant \varepsilon_2$，则称 $f(A)$ 为热点模块。其中 $0<\varepsilon_1,\varepsilon_2<1$，$Cycle(\ )$ 和 $LLC_Miss()$ 分别表示 Profiling 工具获取的时钟数和 LLC 缺失数。

热点程序模块的特征可以用 CPI（clocks per instruction）度量指标来描述。在乱序发射的处理器核上所能达到的指令级并行级别并不能完全隐藏主存储器的访问延迟，仍会导致过多的指令流水线停顿。因此，程序中 CPI 指标越高的模块，往往是缓存缺失最严重的模块。观察表 1 中的局部模块，可发现 BFS 循环模块和 Backtrack 模块的 CPI 较高，而其最后一级缓存缺失分别占整个程序 LLC 缺失的 61.31%和 37.33%。若 $\varepsilon_1 = 0.2$，$\varepsilon_2 = 0.3$，依据定义 4，这 2 个模块就是热点模块，提高它们的性能就必须通过合理的主存访问延迟隐藏技术来达到目的。

在 SSCA2 测试程序的 BC 算法实现中，为了节省算法的空间开销，稀疏图的数据结构采用了数组数据结构。稀疏图的存储主要由索引数组和后继节点数组完成，索引数组自身的索引用来表示图中的节点，后继节点数组用来存储图中某个节点的后

表 1 SSCA2 中 BC 算法程序热点分析（输入 scale 为 16，Q6600 平台）

热点模块	时钟数(10^3)	指令数(10^3)	LLC 缺失数(10^3)	每指令时钟数（CPI）	时钟百分比/%	LLC 缺失百分比/%
SSCA2 程序	75 474 000	25 504 000	221 000	2.959 301	100	100
BC 算法程序	69 198 000	21 040 000	219 000	3.288 878	91.68	99.10
BFS 模块	48 420 000	16 624 000	135 500	2.912 656	64.15	61.31
Backtrack 模块	18 646 000	3 352 000	82 500	5.562 649	24.71	37.33

继，同时 BC 算法实现过程中的数据变量均采用线性数组表示。

4.3　BC 算法预取代码构造

根据对 BC 算法的 Profiling 分析结果，利用 KPB 帮助线程预取框架来构造 BC 算法的帮助线程。由于 BC 算法中 BFS 模块和 BackTrack 模块分别属于 2 个独立的循环中，因此在帮助线程中通过构造 2 个预取模块来实现对主线程所需数据的推送。以 BFS 模块为例，帮助线程构造的算法伪代码如图 2 所示。

从图 2 中可以看出，基于 KPB 推送框架构造的帮助线程预取模块，在保证和原始程序数据访问流一致的情况下，当主程序开始进入热点循环之后，帮助线程并不是从主线程当前访问的节点开始广度优先搜索，而是在队列中跳跃 *SKIP__K* 个节点，从主线程当前访问节点的第 *K*+1 个节点开始广度优先搜索。为了防止帮助线程任务过重，在推送完 *PUSH_P* 个节点的邻接节点数据之后，与主线程同步一次，之后再向前跳跃 *SKIP_K* 个节点，从而保证帮助线程预取的及时性。图 2 中，通过软件预取指令来实现长延迟访存指令的数据访存预取操作，主要是因为软件预取指令在总线带宽紧张的情况下可被处理器丢弃，从而减少帮助线程对总线带宽带来的压力。活跃变量同步通过共享变量来实现，为了避免假共享引起的缓存行抖动，共享变量定义为与缓存行对齐并且大小等于缓存行大小。

4.4　帮助线程预取优化原则和方法

帮助线程的预取控制模型的参数选择是解决帮助线程预取性能优化的关键。模型中通过 *K*、*P* 和 *B* 3 个参数控制帮助线程数据预取的效果。对于不同的测试程序，其控制参数的最优取值不同，即使对于同一测试程序，运行环境不同，参数的最优值也不相同，因此给出帮助线程性能优化的原则和方法。

定义 5　热点循环 $H(L)$：对于热点模块 $f(A)$ 中的任意循环 *L*，包括嵌套循环，若最大循环次数 $Iterations(L) \geqslant \varepsilon_3$，则称 *L* 为一个热点循环。

原则 1　程序热点模块和循环的预取性能获益要能弥补帮助线程的启动和同步开销。热点模块 $f(A)$ 的选择可依据定义 4 中合理的 $\varepsilon_1, \varepsilon_2$ 阈值来确定；热点模块中热点循环 $H(L)$的选择可依据定义 5 中合理的 ε_3 阈值来确定，ε_3 阈值过小不能够折衷帮助线程的启动和同步开销。在程序热点模块中存在循环多级嵌套的情况下，由 ε_3 阈值决定在哪一级循环构造帮助线程预取代码。Kim[7]指出热点循环应该至少运行 1 000 次才能弥补帮助线程的开销，但是实际应用中应当根据不同应用程序的 Profiling 结果做出具体的选择。

原则 2　帮助线程构造代码是有效的，当且仅当主线程中的 LLC 缺失部分转移至帮助线程。在此情况下，主线程性能提升取决于帮助线程控制参数的选取。

原则 3　帮助线程的 KPB 参数优化依赖于热点循环的类型，针对存储密集型循环，优先考虑 *K* 值选取，针对计算密集循环，优先考虑 *P* 值的选取。一般情况下参数 *B* 等于参数 *K* 与 *P* 值之和。

依据以上原则，帮助线程参数优化的步骤如下。

step1　确定参数 *K* 的范围。

step2　根据参数 *K* 和帮助线程预取率 $R_p = P/(K+P)$ 生成参数 *P*，确定参数 *B*=*K*+*P*。

step3　将参数 *K* 和 *P* 应用于程序热点模块，获取程序运行过程中的实时性能信息 CPI（每指令时钟数）。

step4　选取最小 CPI 值对应的参数组合即为最优参数。

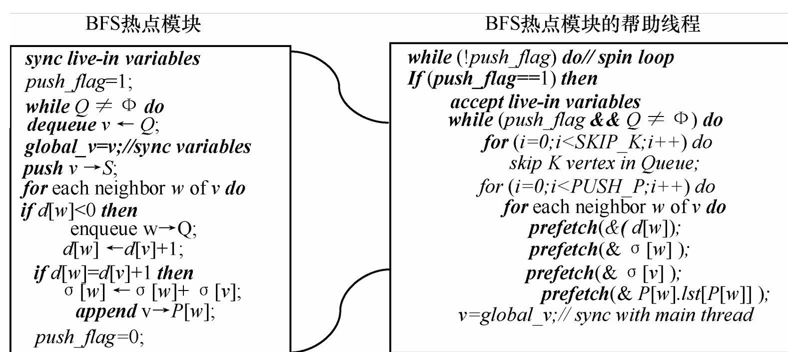


图 2　BFS 模块帮助线程架构

5 性能评测

5.1 实验环境

表 2 是实验所用 2 个 CMP 平台 Intel Q6600 和 I7 架构的参数。Core2 Quad Q6600 处理器芯片共有 4 个 CPU 核、8 个 L1 缓存和 2 个 L2 缓存。每个 CPU 核独享一级指令 Cache(I-Cache)和一级数据 Cache(D-Cache)，4 个处理核分 2 组分别共享两个 4 MB 大小的 L2 缓存（LLC），2 个 L2 缓存分别通过总线和主存储器相联，可将处理核分为{(0,1),(2,3)}。Intel Core I7 处理器属于 Intel Nehalem 微结构系列，该处理器有 4 个物理核，每个物理核支持超线程运行，因此整个处理器有 8 个逻辑处理核。将处理核分为{(0, 4), (1, 5), (2, 6), (3, 7)}4 组，每组共享一个物理核，具有单独的 L1 和 L2 缓存，所有的处理核共享一个 8 MB L3 缓存。在实验中，2 个平台所有的硬件预取器都保持缺省的开启状态，In Core I7 平台的 Turbo Boosting 选项为打开状态。

5.2 基准测试程序

实验测试程序采用美国乔治亚理工学院 David A. Bader 教授研究组开发的 SSCA2 测试程序。SSCA2 是一个模拟现实世界图信息计算应用的基准测试程序，其主要特征是所有运算均为整数运算，程序运行占用内存大，访存行为呈现出不规则模式。SSCA2 程序包括 4 个模块，第 1 个模块利用无尺度（scale-free）生成图算法完成图的构造，能够根据输入参数不同实现不同规模的图构造。后续 3 个模块均是在已构造的图上进行分析运算。介度中心算法属于第 4 个模块，主要用来计算标识节点重要性的介度中心指标，同时 SSCA2 测试程序还提供了基于 OpenMP 实现的并行 BC 算法。

5.3 实验结果与分析

为避免真实机器上程序运行时间的波动，所有实验结果均在 Q6600 平台和 I7 平台分别运行 3 次（O3 优化级别），取 3 次运行结果的中位数作为评测结果（如图 3 和图 4 所示）。图 3 和图 4 中_ORI 代表原始程序，_OMP 代表 SSCA2 测试程序的 OpenMP 并行优化（采用 2 个线程并行情况），_HT 代表采用帮助线程进行优化之后的程序。HT 表示帮助线程。Scale 为所生成无尺度图的大小，例如，当输入 scale 为 16 时，生成的图节点数 $2^{16}-1$ 个。本实验中，Q6600 平台帮助线程的 K-P-B 参数为（BFS 模块：50-21-71；Backtrack 模块：70-105-175),I7 平台帮助线程的 K-P-B 参数为（BFS 模块：440-188-628；Backtrack 模块：10-1-11），KPB 参数通过 Profiling 实验确定，使得 BC 算法性能获得最佳提升的参数组合作为候选 KPB 参数组合。

5.3.1 帮助线程预取对 L2 缓存缺失的影响对比

图 3 和图 4 中 MPKI（miss per kilo instructions）表示每千条指令 LLC 缺失数。CPI 表示每指令时钟数。从图中可以看出，和原始串行程序相比，实施帮助线程预取优化后， BC 算法热点函数发生的缓存缺失情况明显减少，而大部分 LLC 缓存缺失被转移到了帮助线程中。例如，图 3 中与 BC 算法原始程序相比，改进后的 BC 算法 LLC 缓存缺失减少了 58.68%(（原始 BC 算法 LLC 缺失-改进后 BC 算法 LLC 缺失）/原始 BC 算法 LLC 缺失)，而性能提高了 29.04%。应用程序的执行时间可以分成 2 部分，一部分是不需要访存的计算工作时间，另外一部分是由于缓存缺失的停顿等待时间。那么实施帮助线程预取优化之后，程序所获得的性能加速比主要来源于主线程 LLC 缓存缺失的等待时间的减少，通过实现存储器级的并行操作加速原单线程程序。

表 2 实验平台参数

处理器	Intel Core 2 Quad Processor Q6600	Intel Core I7-930 with Hyperthreading
内存大小	2 GB(DDR 667, non-ECC)	4 GB(DDR3, non-ECC)
L1 数据缓存	32 KB×4 8 路组相联 缓存行大小 64 byte	32 KB×4 8 路组相联 缓存行大小 64 byte
L1 指令缓存	32 KB×4 8 路组相联 缓存行大小 64 byte	32 KB×4 4 路组相联 缓存行大小 64 byte
L2 缓存	4 096 KB×2 16 set-association cache line 64 byte	256 KB×4 8 路组相联 缓存行大小 64 byte
L3 缓存	—	8 192 KB×1 16 路组相联 缓存行大小 64 byte
前端总线速度	1 066 MHz	4.80 GT/s Intel®QPI
编译器	gcc version 4.3.0	gcc version 4.3.0
操作系统	Fedora 9 with kernel 2.6.25	Redhat Enterprise with kernel 2.6.37.1

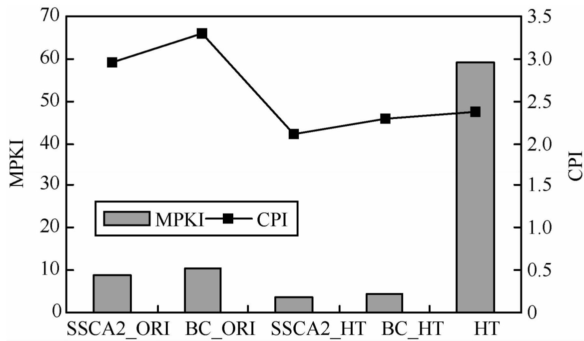


图 3　Q6600 平台 LLC 缓存缺失和时钟改善情况（scale 为 16）

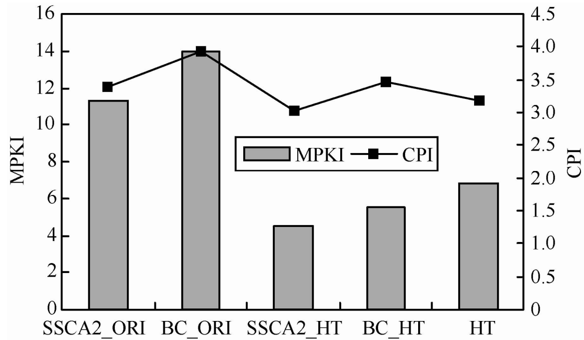


图 4　I7 平台 LLC 缓存缺失和时钟改善情况（scale 为 18）

5.3.2　帮助线程预取有效性分析

一般使用预取准确率、预取覆盖率和预取时效性 3 个指标来分析预取的效果。然而与模拟实验不同的是，在真实机器上很难精确区分帮助线程有用预取和无用预取的数量，幸运的是可以通过测量应用程序在应用帮助线程预取前后的访存行为变化得到近似结果。通过使用当前多核平台的 PMU 事件，可以测量应用程序的内存访问行为。有用预取可以通过应用帮助线程预取前后 LLC 缺失的变化来近似逼近，因此预取准确率可以由有用预取与帮助线程发出预取请求数量之比来表示，预取覆盖率可以由有用预取与主线程对 LLC 的请求数量之比来表示。以 Q6600 平台为例，图 5 为 SSCA2 测试程序在实施帮助线程预取后的预取准确率和覆盖率情况，帮助线程控制参数如前所述,可看出预取覆盖率达 50%以上，预取准确率可达 30%以上，并且预取覆盖广的情况下其准确率也会变高。

为了有助于对帮助线程预取的时效性进行分析,现将有用预取划分为 LLC 缓存完全命中预取和 LLC 缓存部分命中预取两大类。完全命中情况表示主线程所需的数据已经在 LLC 缓存中,部分命中情况是指主线程所需数据已经由帮助线程预取发出，但是其访存请求还处于 LLC 缓存的 MSHR（miss state hold register）中，所请求数据还未被响应。图 6 显示了 SSCA2 在不同问题规模下预取的时效性。图中正值表示其所代表的事件是增加的，相反，负值表示减少。从图 6 中看出，当程序输入 scale 为 13，14 时，主线程的 LLC 缓存完全命中事件减少，尽管其缓存部分命中事件是增加的，但二者综合作用会使帮助线程的预取性能降低。而当在问题输入 scale 为 15、16、17 时，帮助线程的性能呈现好转，说明当前帮助线程的控制参数对问题输入 15、16、17 的情况下适应性良好。通过分析发现，帮助线程的参数控制调节时,应尽可能地将 LLC 缓存部分命中的情况转化为 LLC 缓存完全命中事件,从而提高帮助线程的预取效果。同时图 6 的实验结果也验证了之前关于帮助线程在不同问题输入下对其控制参数要求也不同的结论[19]。

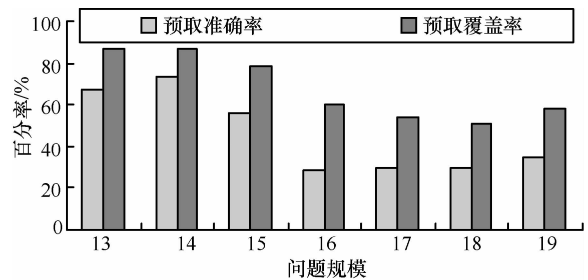


图 5　Q6600 平台帮助线程预取的准确率和覆盖率

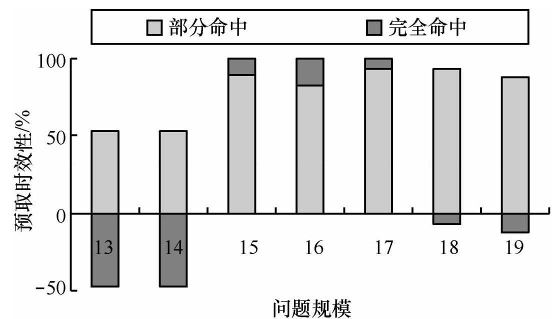


图 6　Q6600 平台帮助线程预取时效性分析

5.3.3　帮助线程参数敏感性分析

帮助线程构造完成之后，就需要确定帮助线程的控制参数。在 KPB 预取框架中，K 表示预取距离，用来控制预取的及时性，P 表示帮助线程的预取数据工作量，B 表示同步距离。K、P、B 的单位为 iterations，即循环数。帮助线程预取率 $R_p = P/(K+P)$，通过调节 R_p 即可动态调整帮助线程的预取及时性和预取覆盖率。以 Q6600 平台上，BFS 热点模块为例，说明帮助线程参数和性能之间的关系。图 7 中参数的横坐标表示不同的参数集合。在即定 K 值下，通过调节 R_p 由 0.1 到 0.9，计算 P 值大小。

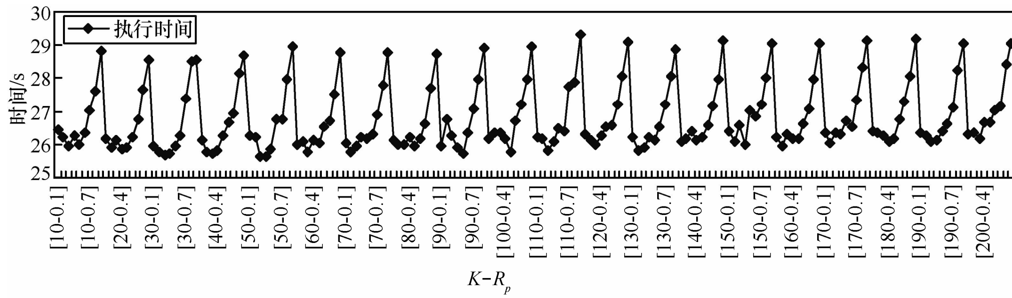


图 7 帮助线程 KPB 参数敏感性分析（scale 为 16）

由图 7 可以看出，当预取率 R_p 周期性变化时，程序的性能也呈现出周期性的变化，并且在每个周期内均有一个最好的参数组合。当 K 不断增大时，程序的性能曲线基本变化不大，表明针对 BFS 热点模块帮助线程的预取性能对于参数 K 不敏感。相反帮助线程的性能对于参数 K、P 的组合非常敏感，即对数据预取率 R_p 敏感。针对 BFS 模块，观察到 R_p 约在 0.3 时，帮助线程预取带来的性能提升最明显。

5.3.4 与 OpenMP 并行优化比较

图 8 和图 9 中纵坐标表示不同配置情况下，程序运行的加速比，横坐标为 SSCA2 测试程序的输入规模 scale。SSCA2 基准测试程序的 OpenMP 并行优化考虑采用 2 个线程并行的情况。

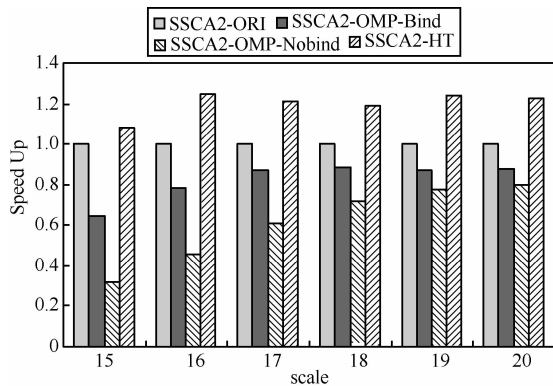


图 8 Q6600 平台与 OpenMP 两线程性能比较

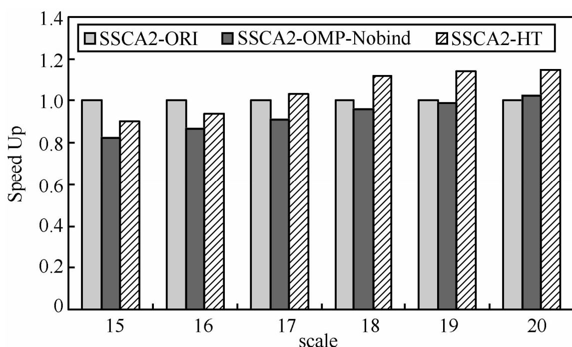


图 9 Intel I7 平台与 OpenMP 两线程性能比较

图 8 中采用 OpenMP 实现的 BC 算法并行程序运行分为 2 种情况：一种情况为将并行执行的 2 个线程绑定在一组共享 LLC 的处理器核上(SSCA2-OMP-Bind)；另外一种情况是不绑定处理器核(SSCA2-OMP-Nobind)，由操作系统随机调度。从图 8 可以看出，在不同 SCALE 输入下，在绑定处理器核心情况下，程序的性能均比不绑定性能好，原因在于 SSCA2 中实现的 OpenMP 并行执行的线程需要大量的数据交换和锁机制，而共享缓存为其提供了性能的提升。由于 I7 平台的 8 个逻辑处理核共享一个 L3 缓存，所以，OpenMP 实现的并行线程没有做绑定限制，即图 9 中只包括 SSCA2-OMP-Nobind 情况。从图 8 和图 9 中可以看出，当问题规模较小时，帮助线程的预取优势不是很明显，主要原因在于帮助线程的预取获利不足以弥补其启动和同步开销。相反，随着问题规模的增大，帮助线程优化后的程序性能加速比基本保持稳定。在 Q6600 平台， BC 算法在不同规模输入下平均性能加速比为 1.20，最高性能加速比为 1.25。在 I7 平台，BC 算法在不同规模输入下平均性能加速比为 1.11，最高性能加速比为 1.15。

图 10 和图 11 所示为 SSCA2 程序采用 OpenMP 优化增加并行线程的数量时的实验结果。图中 OMP-n 中 n 表示 OpenMP 并行线程的数量。

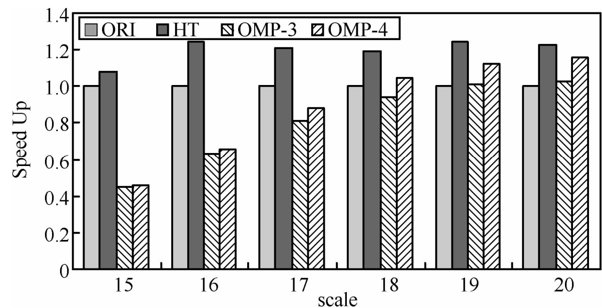


图 10 帮助线程与 OpenMP 多线程性能比较（Q6600）

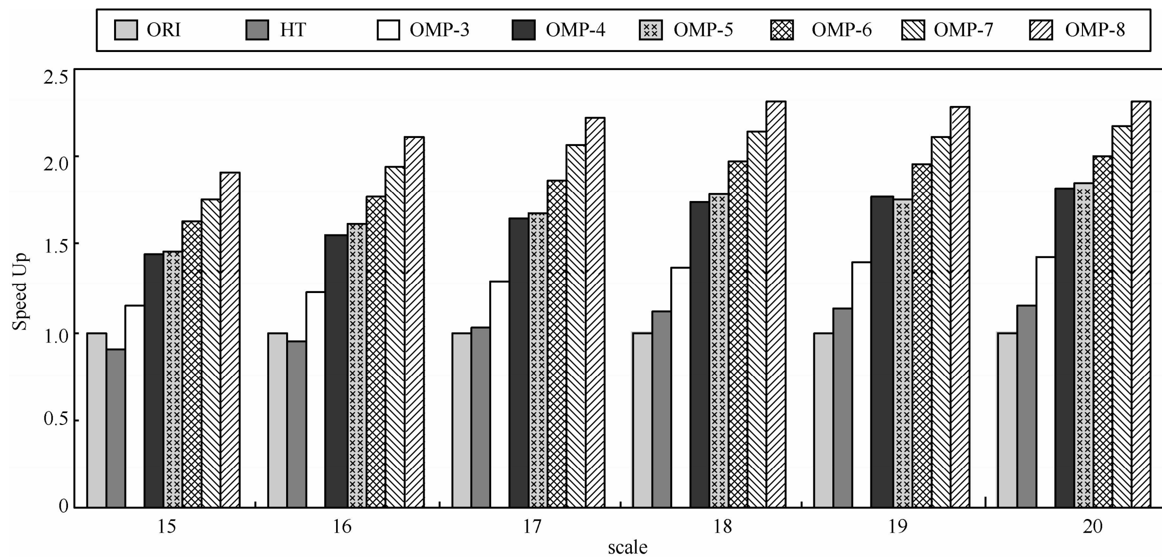


图 11　帮助线程与 OpenMP 多线程性能比较（Intel I7）

图 10 表示 Q6600 平台增加 OpenMP 并行线程的数量到处理器最大并行执行线程数 4 时，随着问题规模的增大，并行程序的执行性能与原程序相比有所提升，但是仍然不及帮助线程预取效果好。深入分析原因发现，由于在 SSCA2 中实现的并行 BC 算法在遍历和回溯过程中，当多个线程并行访问同一个节点时，需要进行加锁互斥操作，由于 Q6600 平台 4 个处理核分 2 组分别共享 2 个 L2 缓存，从而导致不共享 L2 缓存的多个线程互斥访问时的性能开销加剧。而在图 11 中，I7 平台上由于 8 个逻辑处理核共享一个 L3 缓存，当逐步增加 OpenMP 并行执行的线程数量时，并行 BC 算法性能提升优于帮助线程预取的性能提升，主要原因是这里所用的仅是原串行程序的帮助线程，而不是配套这里并行程序的多帮助线程技术。

6　结束语

介度中心算法的核心是图理论中具有代表性的广度优先搜索算法（BFS），其广泛应用于基于稀疏图遍历的大数据分析系统。针对非规则大数据分析的特点，利用片上多核最后一级共享缓存，通过提出的帮助线程预取方法，实现了这类非规则应用程序的性能改进。本文对 SSCA2 测试程序中的 BC 算法进行了深入分析，提出一种基于帮助线程的预取控制模型和参数优化方法，并在商用多核平台上予以验证。虽然帮助线程的应用减少了 CMP 平台可用处理核的数量，但是对于具有非结构化并行特征或很难并行的应用来说，帮助线程预取能够有效提升该类应用程序的性能，为提高当前大数据分析系统的性能提供了可行的解决方案。下一步工作是继续深入研究自适应的帮助线程预取机制，实现帮助线程控制参数的实时动态调节。

参考文献：

[1] SOFFER A, HEILD M. Big data meets social analytics[A]. Proc of 2012 Conference of IBM Lotusphere[C]. Orlando, Florida, USA, 2012.

[2] BRANDES U. A faster algorithm for betweenness centrality[J]. Journal of Mathematical Sociology, 2001, 25(2):163-177.

[3] BADER D, MADDURI K, GILBERT J, *et al*. Designing scalable synthetic compact applications for benchmarking high productivity computing systems[J]. CTWatch Quarterly, 2006, 4B(2):1-10.

[4] LUK C. Tolerating memory latency through software controlled pre-execution in simultaneous multithreading processors[A]. Proc of the 28th Annual International Symposium on Computer Architecture (ISCA)[C]. Göteborg, Sweden, 2001. 40-51.

[5] SMITH JE. Decoupled access/execute computer architectures[A]. Proc of the 9th International Symposium on Computer Architecture (ISCA)[C]. Austin, TX, USA, 1982. 112-119.

[6] SONG Y, KALOGEROPULOS S, TIRUMALAI P. Design and implementation of a compiler framework for helper threading on multi-core processors[A]. Proc of the 14th International Conference on Parallel Architectures and Compilation Techniques (PACT)[C]. Saint Louis, MO, USA, 2005. 99-109.

[7] KIM D, LIAO S, WANG P, *et al*. Physical experimentation with prefetching helper threads on Intel's hyper-threaded processors[A]. Proc of the International Symposium on Code generation and Optimization (CGO)[C]. Palo Alto, Calif, 2004.27-38.

[8] LEE J, JUNG C, KIM D, *et al*. Prefetching with helper threads for lossely coupled multiprocessor systems[J]. IEEE Transactions on Parallel and Distributed System, 2009, 20(9):1309-1324.

[9] LU J, DAS A, HSU W, *et al*. Dynamic helper threaded prefetching on the Sun UltraSparc CMP processor[A]. Proc of 38th Annual IEEE/ACM International Symposium Micro Architecture (MICRO)[C]. Barcelona, Spain, 2005. 93-104.

[10] ZHOU J, CIESLEWICZ J, ROSS K, *et al*. Improving database performance on simultaneous multithreading processors[A]. Proc of the 31th International Conference on Very Large Data Bases[C]. Trondheim, Norway, 2005. 49-60.

[11] HUANG Y, TANG J, GU Z, *et al*. The performance optimization of threaded prefetching for linked data structures[J]. International Journal of Parallel Programming, 2012, 40(2):141-163.

[12] DUDAS A, JUHASZ S. Reconfigurable pre-execution in data parallel applications on multicore systems[A]. Electrical Engineering and Intelligent Systems Lecture Notes in Electrical Engineering[C]. 2013. 29-38.

[13] MADDURI K, EDIGER D, JIANG K, *et al*. A faster parallel algorithm and efficient multithreaded implementations for evaluating betweenness centrality on massive datasets[A]. Proc of IEEE International Symposium on Parallel & Distributed Processing(IPDPS)[C]. Rome, Italy, 2009. 1-8.

[14] TU D, TAN G, SUN N. Fine-grained parallel betweenness centrality algorithm without lock synchronization[J]. Journal of Software, 2011, 22(5):986-995.

[15] EDMONDS N, HOEFLER T, LUMSDAINE A. A space-efficient parallel algorithm for computing betweenness centrality in distributed memory[A]. Proc of International Conference on High Performance Computing (HiPC)[C]. Goa, India, 2010.1-10.

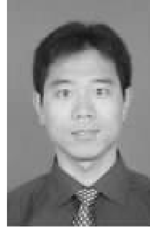
[16] TU D, TAN G. Characterizing betweenness centrality algorithm on multi-core architectures[A]. Proc of IEEE International Symposium on Parallel and Distributed Processing with Application(ISPA)[C]. Chengdu, China, 2009.182-189.

[17] TAN G, VUGRANAM C, SREEDHAR, *et al*. Analysis and performance results of computing betweeness centrality on IBM Cyclops64[J]. Journal of Supercomputing, 2011, 1(56):1-24.

[18] TAN G, VUGRANAM C, SREEDHAR, *et al*. Just-in-time locality and percolation for optimizing irregular applications on a manycore architecture[A]. Proc of Conference on languages and compilers for parallel Computing Lecture Notes in Computer Science[C]. Edmonton, Canada, 2008.331-342.

[19] ZHANG J, GU Z. Exposing the shared cache behavior of helper thread on CMP platforms[A]. Proc of the 14th IEEE International Conference on Computational Science and Engineering(CSE)[C]. Dalinan, China, 2011. 379-386.

作者简介：



张建勋（1978-），男，河北保定人，北京理工大学博士生，主要研究方向为高性能计算、多核缓存优化技术。

古志民（1964-），男，山西运城人，博士，北京理工大学教授、博士生导师，主要研究方向为并行计算与分布式计算、多核缓存优化等。

胡潇涵（1988-），女，山西陵川人，北京理工大学硕士生，主要研究方向为多核计算、缓存优化研究等。

蔡旻（1982-），男，湖南长沙人，北京理工大学博士生，主要研究方向为多核体系结构、存储子系统优化和体系结构模拟。