高效 KD 树并行算法优化①

李天驹[1,2,3], 张 铮[4], 张为华[3]

[1](复旦大学 软件学院, 上海 201203)
[2](上海市数据科学重点实验室(复旦大学), 上海 201203)
[3](复旦大学 并行处理研究所, 上海 201203)
[4](解放军信息工程大学 数学工程与先进计算国家重点实验室, 郑州 450001)

摘 要: KD 树作为一种用于查询高维键值的流行算法, 由于其准确性高、可扩展性强与较快的查询速度而应用于多媒体检索领域, 但缓慢的建树效率已不能很好的满足当前的应用场景. 针对 KD 树的低效建树过程, 作者探寻并分析了 KD 树建树现存的并行潜能并提出了一种面向 KD 树建树过程的多核并行算法—ParK(Parallel KD-Tree). ParK 探求了不同的并行模式来充分利用现代硬件中的计算资源, 并在此基础上提出了一种新的内存分配策略来解决并行处理中的数据争用状况. 实验结果表明 Park 相比于原始串行版本最高能够在 16 核的服务器上达到 21.75 倍的加速.

关键词: 多媒体检索; KD 树; 多核; 并行

Parallelization Optimization of KD-Tree Building Algorithm

LI Tian-Ju[1,2,3], ZHANG Zheng[4], ZHANG Wei-Hua[3]

[1](Software School, Fudan University, Shanghai 201203, China)
[2](Shanghai Key Laboratory of Data Science, Fudan University, Shanghai 201203, China)
[3](Parallel Processing Institute, Fudan University, Shanghai 201203, China)
[4](State Key Laboratory of Mathematical Engineering and Advanced Computing, PLA Information Engineering University, Zhengzhou 450001, China)

Abstract: In recent years, multimedia data has become one of the major data types transferred and processed on the Internet. K dimensional tree is one of the most popular tree structures for searches involving a multidimensional search key, which is similar to feature points extracted from multimedia data, due to its good accuracy, scalability and fast retrieval speed. However, its slow building speed limits its application area, especially with large dataset. Fortunately, Modern processors provide tremendous computing power by integrating multiple or many cores. In this paper, we explore and analyze the existing potential parallel in KD-Tree building process. Then we present ParK, a customized parallel solution that exploits multi-core CPUs to accelerate KD-Tree building process. ParK exploits different parallel models to fully utilize computation resource in modern hardware and solves data race by presenting a new memory allocation strategy. The final experimental results show ParK achieves about 21.75X speedup compared to original serial version on 16-core server.

Key words: multimedia retrieval; KD-Tree; multi-core; parallelism

大数据时代推动了互联网的发展, 也对互联网上的各种应用提出了更高的数据处理能力要求. 多媒体数据作为互联网上最主要的数据类型之一, 相关的应用已经成为了人们生活密不可分的一部分. 知名网站 YouTube 每分钟处理的视频时长超过 100 小时[1]; 每天有超过合计大小超过 300MB 的图片(近 7PB 每月)被上传至著名社交网站 Facebook. 为了解决当下互联网面对的处理多媒体数据的巨大压力, 相关研究的重心开

① 基金项目:上海市科委科技攻关项目(13DZ1108800);国家高技术研究发展计划(863)(2012AA010901);国家自然科学基金(61370081)
收稿时间:2014-12-03;收到修改稿时间:2015-01-12

始转移至如何持续高效的收集、处理、查询并分析不断增长的多媒体数据.

图像的匹配与检索算法作为多媒体数据处理的基础算法, 被广泛应用于诸如物体检索[2]和动作识别[3]等领域. 而 KD 树[4](k-dimensional 树)作为一种分割 k 维数据空间的数据结构, 主要应用于多维空间关键数据的搜索(如: 范围搜索和最近邻搜索). 为了方便图像数据的处理, 通常用大量能代表图像特征的特征向量来指代整个图像数据, 这种特征向量符合了高维键值的特点, 适合利用 KD 树来建立其索引从而加速图像的检索过程. 然而, 虽然 KD 树本身的数据结构特点非常适合用来做图像的匹配与检索算法, 但随着数据量的增大, 它本身较慢的建树速度成为其应用的主要瓶颈.

尽管 KD 树本身的特点使得它在用于图像检索时表现出较好的适应性, 但其效率, 尤其是进行建树的训练阶段效率, 显得较为低下. KD 树在建树过程中面临的最大挑战主要来自于两方面: 1)计算密集带来的高耗时: 由于在建立索引树的训练阶段, KD 树的建树算法需要通过多次迭代进行子空间划分, 划分的结果还要反复进行更细致的划分, 较为耗时, 在数据规模大的情况下尤甚; 2)不合理的内存分配策略阻碍进一步优化: 早期实现中的内存分配方式并未考虑到进一步的优化, 在固有的内存分配模式下进行性能优化较为困难, 需要提出新的内存分配方案.

KD 树建树算法在应用到实际应用时, 必须充分认识到当前的挑战, 并且有针对性的进行更多的优化, 才能更好地处理庞大的数据量, 充分满足当前的用户需求.

近年来, 现代处理器保持高速发展, 多核处理器已经成为主流, 其丰富的并行资源为提高应用性能提供了广阔的前景, 应用的并行化也已经成为现在主流. 本文中, 我们提出并分析了 KD 树建树过程中可以利用多核处理器来进行并行优化的潜在可能性. 我们提出了 ParK(Parallel KD-Tree), 这是一种利用多核处理器并行优化来加速 KD 树建树过程的解决方案. 在 ParK 中, 我们基于上述的讨论提出了一种高效的并行模式: KD 树建树过程中的计算分割超平面及排序部分的并行化; 为了解决数据争用问题, 提出了一种优化了的内存分配方式来替代传统的方式.

实验结果表明了 ParK 性能的优越性: 相比较于串行版本可以在 16 核的服务器上达到最高 21.75 倍的加速.

本文的主要贡献如下:

① 对于 KD 树建树过程并行所面临的主要挑战进行了全方位的分析;

② 基于多核处理器架构提出了一种并行的 KD 树建树的优化算法

本文余下的组织结构如下: 第一章简单介绍当今多媒体检索所面临的主要挑战, 针对 KD 树算法的建树过程进行探讨并分析其主要瓶颈; 第二章提出 ParK 的设计思路与具体实现; 第三章说明具体的实验环境配置, 分析实验数据; 第四章介绍其它针对图像检索算法优化加速的研究工作; 第五章对全文进行总结, 对未来的工作进行展望.

1 背景介绍

在本章节中, 首先对图像检索算法进行简单的介绍. 接下来, 概述 KD 树算法并介绍当前主流的并行硬件架构. 最后对 KD 树优化中的主要挑战进行介绍.

1.1 图像检索系统

近年来, 多媒体数据已经成为互联网上传播与处理的主要数据类型之一. 在 2008 年的一项统计中, 图像和视频已经占据了 80%的互联网数据[5]. 因此如何能高效的从持续增长的多媒体数据中检索出对于用户有价值的信息成为了当今的一个重要课题. 而图像检索作为多媒体检索的基础更是相关研究的热点之一.

一个较典型的图像检索系统可以分为如图 1 顺序所示三个阶段: 特征提取阶段、特征匹配阶段、几何验证阶段.

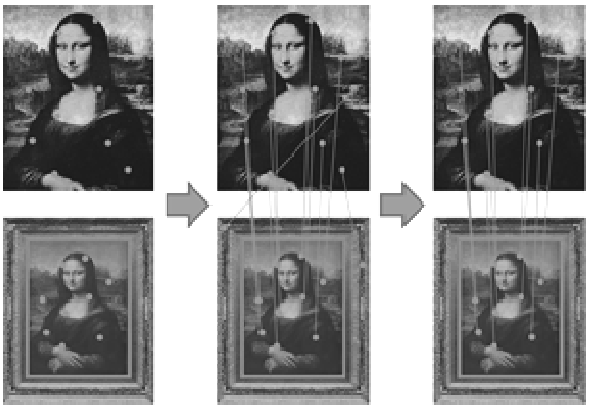

图 1 图像检索系统的组成

1) 特征提取: 这个阶段主要是将具体的多媒体数据(图像)转化成为便于处理的、能够指代整个图片特征的信息(特征点). 为了检索结果的有效性, 特征点通常是利用诸如 SIFT[6], SURF[7]算法提取的高维向量.

2) 特征匹配: 为了确保匹配过程的高效率, 诸如 KD 树[8], Vocabulary tree[9]和 LSH[10,11]等对图像检索有良好支持的索引结构被应用到实际中. 而 KD 树是这类算法中最为流行的算法之一.

3) 几何验证: 在特征匹配阶段中, 为了兼顾效率, 通常使用的并不是严格的比配. 为了进一步提高精度, 引入了几何验证算法(如 RANSAC[12])来过滤匹配较差的结果.

1.2 KD 树算法

KD 树作为当前最流行的检索结构之一, 已经被广泛的应用于各个领域.

KD 树是每个节点为一个 K 维向量的二叉树. 每个非叶子节点可以看作一个超平面, 而这个超平面将多维空间分割为两个子平面. 在这个超平面左侧的点被分为左子树, 右侧的点则被分为右子树. 决定这个超平面方向的方式如下: 每个 KD 树中的点都与 K 维向量中的一个特定维度相关联, 而这个维度正是垂直于分割空间的超平面的轴的维度. 举个例子, 如果这个轴是 X 轴, 那么 x 的值小于决定这个超平面的点的剩余点被分到左子树, 所有 x 的值大于这个点的被分到右子树.

决定这个分割子空间的超平面的方法有很多种, 所以构建 KD 树的方法相应的也有多种. 其中计算子平面中所有点每一维的方差并选择方差最大的维度作为垂直于分割子空间的超平面的方向是当今最为流行的构建 KD 树的方法之一[13,14]. 本文中, 我们也是使用了这种应用最为广泛的方式, 而这种方式带来的好处之一就是产生的 KD 树基本上是一个平衡的二叉树. 图 2 是一个 KD 树建树过程的示例, 用来建立 KD 树的空间中所包含的点集为(2,3), (5,4), (9,6), (4,7), (8,1), (7,2). 而图 2 中的上半部分可以看到在 KD 数建树过程中产生的垂直于分割各个子空间的轴, 而下半部分则是建树过程后最终生成的 KD 树.

1.3 并行硬件架构

而今, 并行硬件平台已经成为各个领域的主流, 而多核及众核(如 GPGPU)架构所具备的众多并行计算资源也为 KD 树建树过程的加速提供了巨大前景. 下面简单介绍一下多核及众核架构.

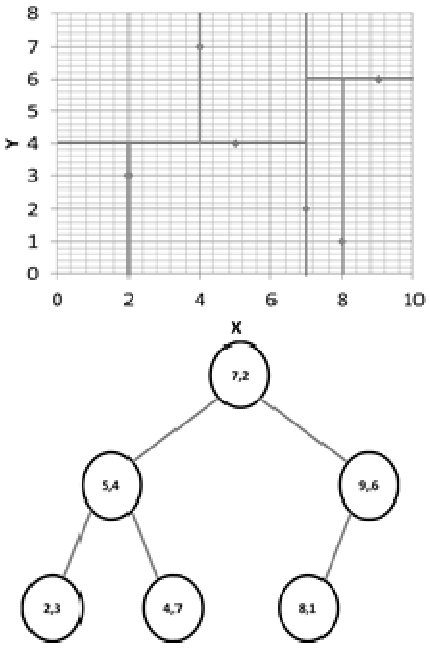

图 2 KD 树建树过程示例

1) 多核架构: 一个多核架构通常是一个包含两个或数个独立的作为基本执行单元的 CPU 核的计算芯片. 这些 CPU 有各自的独立缓存和(或)更大的共享缓存. 这些核通常执行不同的线程并通过共享缓存或者芯片上的通信网络来交换数据. 本文中, 我们主要关注的正是发掘多核架构的并行性能来加速 KD 树的建树过程.

2) 众核架构: 相对于多核结构而言, 众核架构中有更多、更密集的组件被集成到了同一个芯片上. 而在众多的众核架构中, GPGPU 是最流行的架构之一. 一个 GPGPU 通常有数百个 streaming-processor(SP)单元, 而这些 SP 就是 GPGPU 中最基本的单元. 数个 SP 组成 streaming -multiprocessor(MP). 每 32 或 64 个线程被分配到一个 warp 中, 而数个 warp 被分配至一个线程块. 每个 SM 支持数个 warp 来隐藏内存时延. 考虑到整个硬件存储体系, 每个 SM 都有一个独立的包含上千个寄存器和数十 KB 的共享内存. 这些片上的内存是 SM 中的数个 SP 共享的并且对外不可见. 不同 SM 间的数据传输是通过片下的全局内存. 通过以上的设计, GPGPU 在整体开销基本与全局内存一致的情况下支持对寄存器和共享内存的寄存器级别访问速度.

1.4 KD 树建树过程所面临的挑战

随着多媒体数据的迅猛增长, KD 树的建树速度面临着巨大的挑战. 这些挑战主要来自于以下两点:

1) 计算压力: 在 KD 树的建树过程中, 每个迭代中的主要耗时部分为决定分割子空间的超平面和依照超平面分配余下的点至子空间两部分. 每天都有大量的图像数据在互联网上产生、传播与处理. 如之前所说, 每天有 300M 的图片上传至 Facebook, 即 7PB 每月. 对于每一张图片而言, 需要提取成百上千个特征点来保证图片检索时的精度. KD 树的建树算法则把每一个特征点作为基本单元来处理. 因此, 每一张图片都有上千个基本单元等待 KD 树处理. 此外, 每一个特征点都包含了一个高维向量(通常为 64 或 128 维). 以上原因都导致了在数据集较大时, KD 树的建树时间过长.

2) 存储压力: 在传统的 KD 树实现中, 内存分配是连续的, 如图 3 所示.

图 3 传统 KD 树的内存分配方式

其中的 Level 指 KD 树中节点所对应的层数, 同一层中相邻的节点在内存中也是相邻的, 这使得这个树型结构更加直观. 这种直接的策略在串行版本中没有影响, 但在实现并行时, 却面临着较大的挑战. 由于各个核之间共享同一块内存而出现数据争用现象. 在不改变当前内存分配策略的情况下, 可以利用在并行程序中使用的内存锁来避免这一现象. 锁操作是极其费时的, 从而无法达到较高的性能. 为了得到明显的性能提升, 需要针对 KD 树建树过程提出新的无锁内存分配方案. 同时, 在传统的 KD 树实现中, 对于方差的计算是按维度进行的, 在每次迭代中每个特征点都要被访问一次, 这就导致了极差的内存局部性, 这使得计算各维方差的阶段非常低效.

为了进一步说明 KD 树建树所面临的挑战, 我们测试了不同大小数据集下 KD 树的建树时间. 具体测试环境会在第三章中说明.

如图 4 所示, 横坐标表示的是图片集大小, 可以看到, 对于一个 10 万张图片的数据集, KD 树的建树时间大约为 1.5 小时, 如此巨大的耗时显然不能令人满意.

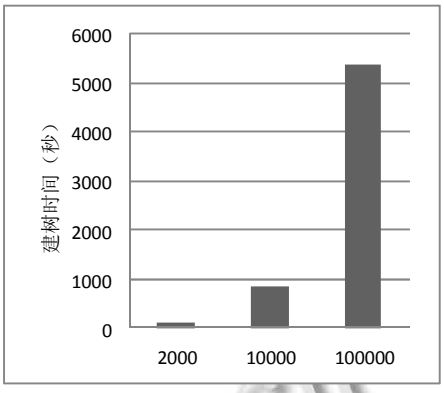

图 4 不同大小图片集建树时间

2 ParK设计

近年来, 诸如多核架构的并行硬件平台已经成为各个领域的主流. 充分发掘这些平台的并行硬件资源为我们加速 KD 树建树过程提供了新的思路与解决方案.

为了达到良好的并行效率, 需要考虑以下几点:

① 如何合理的分配并行运算资源? 何种并行模式能够使得 KD 树的建树过程更加高效?

② 如何在特定的并行模式下减少各个核之间的数据争用现象?

③ 如何进一步加速计算分割子空间的超平面?

④ 如何针对 KD 树建树过程中的排序操作与数据读取进行并行优化?

基于上述问题, 我们首先针对 KD 树建树过程提出一种新的并行设计, 然后针对内存性能的高效利用重新设计了一种新的内存分配方案并且优化内存局部性, 最终提出了一种高效的 KD 树并行实现: ParK(parallel KD-Tree).

2.1 并行模式分析

在 KD 树的建树过程中, 每一次迭代都需要计算子空间中所有的点所有维度的方差并以此确定分割空间的超平面. 因此, 如何能在多核架构上合理的分配特征点的计算任务是提升并行效率的关键所在. 为了通过并行方式加速 KD 树的建树过程, 我们对 KD 树建树过程中现存的并行潜力进行分析. 直观上看 KD 树建树存在两种基础的并行方案: 层级并行与子树级并行.

1) 层级并行: 在这种并行模式下, 每个节点分配一个进程进行计算, 而同一层级中节点间并行运算, 即在不超过硬件并行度上限的前提下, 第 n 层的并行

度为 2n. 由于每层的所有节点是并行关系, 因此称为层级并行. 每个节点的任务分配如图 5 所示, 其中曲线代表线程, 而箭头方向代表建树过程中层级的推进.

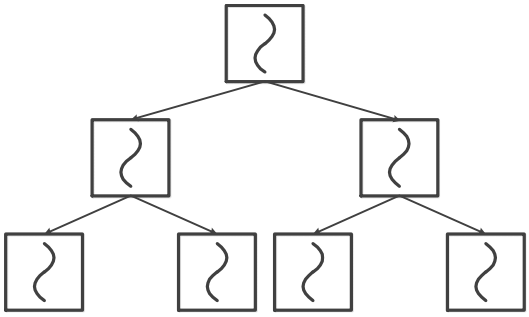

图 5 层级并行中各节点的任务分配

这种分配方式似乎既简单明了又充分利用了各级的并行性. 但在实际情况中, 由于 KD 树种低层级节点的运算是依赖于高层级节点的(即存在数据依赖关系), 并且高层级中子空间更大, 这就意味着高层级有着更大的运算量(两倍于其子节点), 高层级节点的运算耗时更多且更为关键. 为了解决这个问题, 应该考虑分配更多的运算资源给高层级节点.

2) 子树级并行: 在这种并行模式下, 每个节点基于运行平台的 CPU 核数分配数个线程来进行相关运算. 任务分配方式如图 6 所示(以 4 核平台为例), 每个节点分配 4 个线程. 由于最高并行度为 4, 每层同时只有一个子树进行运算, 虚线部分代表需要等待其它子树运算完毕才能进行, 因而成为子树级并行.

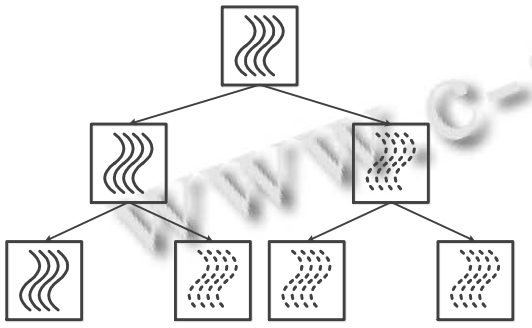

图 6 子树级并行中各节点的任务分配

在这种模式下, 由于数个线程负责同一个节点的相关计算, 存在着线程间同步与通信的开销, 也就存在着计算时间减少与通行时间增加的权衡问题. 在较高层级中, 由于单个任务较大, 计算时间占据整体时间的大部分, 应用子树级并行能较好的加速建树过程; 但在较低层中, 由于单个任务较小, 同步、通信时间占大部分, 无法得到理想的加速效果, 甚至相对于串行性能反而有所下降.

由于这两种基础的并行模式都各自存在着明显的利弊, 如何实现一种扬长避短的并行模式是加速 KD 树建树过程的关键. 以上分析都是基于理论层面, 我们测试了各层在两种基础并行模式下的耗时, 以验证我们的分析, 具体结果和分析在第三章中给出.

2.2 ParK 设计

在本小节中, 我们综合考量了之前提出的两种基础并行模式的优缺点, 提出了一种新的并行模式. 为了提高 KD 树建树过程中特征点排序与读取的效率, 实现了这两部分的并行化. 为了进一步加速计算分割子空间的超平面的速度, 重新设计了这一部分以达到更好的内存局部性. 为了应对在并行程序中的数据争用现象, 我们还提出一种新的无锁内存分配方式来取得更高的加速比.

2.2.1 并行模式设计

层级并行与子树级并行这两种模式都有其明显的优势与劣势, 我们基于这两种基础并行模式, 提出了一种新的扬长避短的并行模式. 在 KD 树的高层中, 由于子空间中包含的点更多、工作量更大而且低层的建立依赖于高层的建立, 需要分配更多的资源. 这种情况下, 显然子树级并行有着较高的效率. 而在 KD 树的低层建立过程中, 由于每个子树任务量较小, 利用子树级并行高昂的通信开销会使得性能下降, 利用层级并行可以达到高加速比且通信开销更小. 我们基于实验结果, 设定一个阈值来决定利用哪种模式进行并行加速: 当该层的节点数总数大于阈值时, 使用层级并行; 而当该层的节点数小于阈值时, 使用子树级并行.

之前关于并行模式的探讨主要针对确定分割空间超平面的运算过程, 而在确定超平面后对子空间中所有的划分(主要为排序操作)也是 KD 树建树过程中较为耗时的一环, 我们期望能利用并行优化在这一阶段取得良好的加速比. 在传统的串行版本中, 这一环节主要利用的快速排序. 快速排序的一个简单递归过程如图 7 所示.

在下一次的递归中, 整个数列会分为两个大小不均的部分, 随着中心点选取的不同, 这种不均匀现象甚至会更严重(其中中心点选为 5 是最均衡的状况, 中

心点选为 1 最不均衡). 快速排序作为时间复杂度最低的排序算法之一, 其应用主要受限于它的不稳定性. 这种不稳定性会使得递归过程中分裂出长度不均等的数列, 而在并行程序中整个程序的运行时间是由运行时间最长的任务决定的, 因此快速排序并不适合并行优化.

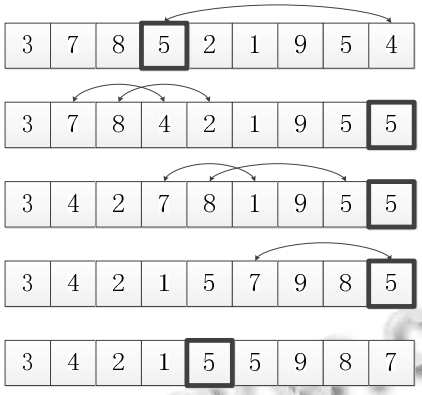

图 7　快速排序递归过程示例

拥有与快速排序同样时间复杂度的归并排序却不存在这个问题, 其在每次递归中都分成两个大小均衡的序列. 归并排序的上述特点, 加上其优越的时间复杂度, 使得归并排序能较为简单与高效的实现并行优化. 由于每次递归产生两个均等的序列, 在并行版本中只需要把每个序列的排序分配给对应线程就能得到线性的加速比. 因此, 我们实现了并行归并排序并将其利用到 KD 树建树过程优化中.

由于在大数据集下特征点的读取也会花费大量的时间, 对于这部分的优化也是不容忽视的. 测试结果显示, 在数据集为 1 万张图片时, 特征点读取时间几乎和建树时间花费相同. 为了加速读取阶段, 对于数据集中每张图片的特征点数建立一个对应的索引表, 在并行优化版本中基于这个索引表来进行具体任务的分配.

2.2.2 基于内存的优化

在传统的 KD 树实现中, 整个树形结构的内存分配是基于层连续分配的. 此时, 同一层中的相邻节点在内存中也是相邻分配的. 这种直观的内存分配方式在串行版本中看似较为合理, 但实际上由于每次访问左子树还是右子树是个随机事件, 导致这种分配方式下无法取得原本期望的良好内存局部性. 这种内存分配方式在在子树级并行模式下, 由于多个核同时处理一个子空间, 会出现内存争用现象甚至是程序出错. 利用内存锁是一个在不改变原有内存分配方式的前提下比较直接的解决方法, 但由于其高昂的同步通信开销使得其严重影响程序效率. 为了达到理想的 KD 树建树加速, 需要实现新的无锁内存分配方式.

在 ParK 中, 我们为每个物理核单独分配了独立的内存来解决数据争用现象, 如图 8 所示.

图 8　独立内存分配方式

在这种分配策略下, 由于每个核有独立的内存空间, 避免对共享内存的同步访问, 实现了无锁的同步访存. 这样又带来了新的问题, 原本连续的树形结构被破坏了. 为了保证并行度, 我们将每个节点存储在处理该节点的对应物理核的私有内存中. 为了保证树形结构的完整性, 每个节点需要更多位来标识其与其它节点的关系. 在 ParK 的设计中, 给原先的节点增加了三位来标明其相对于父节点、左孩子及右孩子在内存中的偏移量, 在保证树形结构完整性的前提下实现高度并行.

提高内存局部性是一个高效但不容易被注意到的优化方式. 计算子空间中所有点的各个维度的方差是计算分割该空间超平面的主要工作, 也是整个 KD 树建树过程中的主要环节之一. 而常用的方差计算公式和它的等价公式如下所示:

$$s^2 = \frac{\sum_{i=1}^{n}(a_i - \bar{a})^2}{n} = \frac{\sum_{i=1}^{n} a_i^2}{n} - \bar{a}^2 \quad (1)$$

图 9 是一个简化的运算过程示意, 其中字母 P 表示的是特征点(feature point), 而下标分别对应的是特征点编号和维度. 在早期实现中, 每个计算周期计算每一个维度的方差. 在原始版本中的运算顺序如(a)中箭头方向所示, 即每次计算周期计算一个维度的方差. 每个计算周期中需要对子平面中的所有特征点都访问一次, 这也带来了非常差的内存局部性.

为了更好的利用内存局部性, 我们对这部分的计算方式进行了重新设计, 如(b)部分所示. 我们利用两个数组(sum 与 square)来存储每个维度的值之和与每个维度值的平方和. 在每个运算周期中, 对该特征点每个维度的值和值的平方在之前累加和的基础上再进行累加, 并覆盖之前存贮在数组 sum 和 array 中的对应值. 最后我们通过已知的 sum 和 square 数组按照如下公式来计

算对应维度的方差(其中 d 表示的维度的编号):

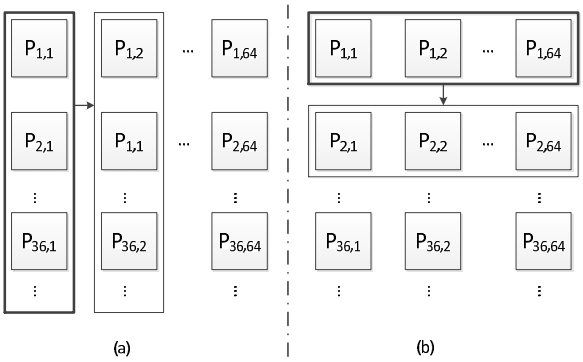

图 9 按维度与按点运算的对比

$$方差[d]=\frac{square[d]}{n}-(\frac{sum[d]}{n})^2 \quad (2)$$

这样最终得到各个维度的方差，决定分割空间的超平面. 由于我们按照特征点来计算方差，整个数组在整个运算过程中，只需要被顺序的访问一次，充分利用了内存局部性. 利用内存局部性的相关优化，进一步提升 KD 树建树过程的速度.

3 实验结果

为了验证关于并行模式优化的结果以及测试最终 ParK 带来的 KD 树建树过程的性能提升，本章节依次给出了相关的测试数据并分析了其具体原因.

3.1 测试环境与测试对象

本次测试中所使用机器的 CPU 为 AMD 6180 SE(2.5Ghz)，内存 64GB. 使用的操作系统为 fedora11，编译器为 GCC 4.4，编译优化选项为-O3. 测试对象集是利用 SURF[7]提取的合计包含 452 万个特征点的 1 万张图片，其中每个特征点为一个 64 维的向量.

3.2 各层耗时

为了验证对两种基础并行模式利弊的分析，我们对层级并行与子树级并行模式下 KD 树建树过程过各层的耗时进行了测试，结果如图 10 所示.

由于测试目的主要是分析各层耗时的规律，并不需要大数据集的支持，我们使用一个含有 48 张图片的小数据集作为测试对象，最大并行度设为 4. 如图 10 中所示，在层级并行中，在未达到并行度上限时可以看作有着线性的加速比(这里上限为 4)，但高层的耗时更多，占了整个建树时间的较大部分. 对于子树级并行，如图 10 所示，虽然在高层中能得到理想的加速比，但由于低层中数据争用现象的加剧，同步时间占据了低层耗时的绝大部分，无法取得理想的加速，甚至还带来了性能上的下降. 实验数据也验证了对于这两种并行模式利弊的分析，为 ParK 最终使用的并行模式提供了理论依据.

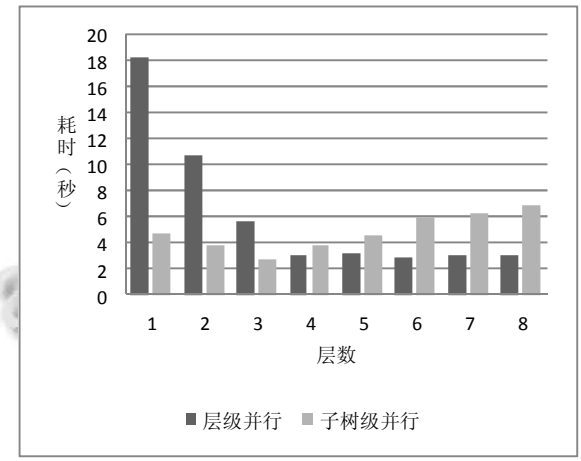

图 10 两种并行模式各层级总耗时(单位: 秒)

3.3 性能对比

在本节中，我们针对 ParK(parallel KD-Tree)的性能优势进行讨论与分析. 整个 KD 树建树过程中各阶段的耗时如图 11 所示.

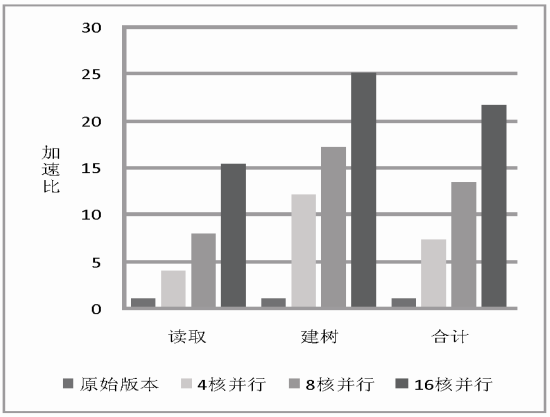

图 11 各版本各阶段加速比(以原始版本为基准)

如图所示，利用了上述所有优化后，ParK 相对于原始版本在 16 核的系统上取得 21.75 倍加速. 建树过程中的主要耗时部分(不考虑数据的读取)为计算分割子空间的超平面和分割后平面中点的归类. 根据测试，这两个部分占据了建树时间约 95%的部分，因此建树阶段最理想条件下并行加速公式如下:

$$speedup = \frac{1}{0.05 + \frac{0.95}{x}} \quad (3)$$

其中 x 代表并行度. 可以看出单纯利用并行优势所取得加速比并不是线性与并行度的, 当并行度为 16 的时候, 理论状态下单纯的并行带来的加速比为 9.14x, 而其余的加速都是内存优化带来的.

4 相关研究

本章节会介绍 ParK 相关研究工作的现状. 相关研究主要可以分为两大类: 多媒体检索相关优化及 KD 树算法优化.

4.1 多媒体检索相关并行

如何能高效的从海量多媒体数据中提取关键信息是当下相关研究面临的挑战之一. 因此多媒体检索相关的优化也主要集中在特征提取领域.

Zhang N.[15]最早提出了发掘 SURF 算法中的图像金字塔并行潜能 P-SURF 算法. 这种算法很好的提升了 SURF 算法中描述符号计算的效率. 测试结果表明 P-SURF 相比于原始版本能带来 2~6 倍的加速比.

陈鹏与杨冬蕾等人[16]设计并实现了一种自适应的流水线系统来处理 SIFT 及 SURF 算法在多核架构上任务分配不平衡的情况, 最终取得了 SIFT 算法 16.88 倍和 SURF 算法 20.33 倍的加速.

GPGPU 也被应用到了特征提取算法中来获得更高的性能. Warn S.[17]针对 SIFT 所做的优化最终取得 13 倍的加速比. Heymann S.[18]在 QuadroFX3400 上实现了 SIFT 算法 20Hz 的处理速度. 而 Cornelis N.[19]和 Terriberry T. B.[20]用 GPGPU 实现了 SURF 算法 100Hz 的处理速度.

4.2 KD 树算法优化

作为一种多维键值检索的基础算法, KD 树被广泛的应用于各个领域. 针对这些各异的应用领域, 有许多研究人员在寻求更高效的利用 KD 树的方法.

Havran[21]提出了一种 KD 树建树过程的自动终止标准并且利用一种新的子空间分割方式来得到精度更高的 KD 树(提高的精度为 10%~30%). 但更为复杂的建树逻辑使得建树速度有所下降.

Di Fatta G.[22]解决分布式存储系统的任务平衡性问题, 提出了一种基于 K-Means 的并行 KD 树算法.

为了提高基于 Surface Area Heuristic(SAH)的 KD 树建树效率, Zhou P.[23]用 GPGPU 来加速了该过程并最终在 CPU 与 GPGPU 的混合架构上取得了 4~5 倍的加速.

5 结论

本文提出了一种基于多核架构的并行 KD 树建树加速算法 ParK(Parallel KD-Tree). 首先提出并分析了 KD 树建树过程中潜在的两种基础并行模式, 并结合它们各自的优点提出了一种新的并行模式并将其应用到 ParK 算法中. 为了应对并行程序中的数据争用现象, 针对 KD 树提出了一种新的无锁内存分配方案. 还针对 KD 树建树过程中的数据读取及排序阶段进行了并行优化. 整个 ParK 算法具有良好的可扩展性, 实验结果表明它能够在 16 核服务器上最高能达到 21.75X 的加速比.

在未来的研究中, 我们计划发掘 GPGPU 平台的并行潜能来加速 KD 树的建树过程. 我们还会花费更多的精力来研究更多图像检索相关领域的算法高度并行化及 GPGPU 支持.

参考文献

1 Youtube statistics. http://www.youtube.com/t/press statistics.

2 Philbin J, Chum O, Isard M, et al. Object retrieval with large vocabularies and fast spatial matching. IEEE Conference on Computer Vision and Pattern Recognition(CVPR'07). IEEE. 2007. 1–8.

3 Mikolajczyk K, Uemura H. Action recognition with motion-appearance vocabulary forest. IEEE Conference on Computer Vision and Pattern Recognition(CVPR 2008). IEEE. 2008. 1–8.

4 Bentley JL. Multidimensional binary search trees used for associative searching. Communications of the ACM, 1975, 18(9): 509–517.

5 Mikolajczyk K, Matas J. Improving descriptors for fast tree matching by optimal linear projection. IEEE 11th International Conference on Computer Vision(ICCV 2007). IEEE. 2007. 1–8.

6 Lowe DG. Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, 2004, 60(2): 91–110.

7 Bay H, Tuytelaars T, Van Gool L. Surf: Speeded up robust features. Computer Vision–ECCV 2006. Springer Berlin

Heidelberg, 2006: 404–417.

8 Vedaldi A, Fulkerson B. VLFeat: An open and portable library of computer vision algorithms. Proc. of the International Conference on Multimedia. ACM. 2010. 1469–1472.

9 Nister D, Stewenius H. Scalable recognition with a vocabulary tree. IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE. 2006, 2. 2161–2168.

10 Gionis A, Indyk P, Motwani R. Similarity search in high dimensions via hashing. VLDB, 1999, 99: 518–529.

11 Datar M, Immorlica N, Indyk P, et al. Locality-sensitive hashing scheme based on p-stable distributions. Proc. of the Twentieth Annual Symposium on Computational Geometry. ACM. 2004. 253–262.

12 Fischler MA, Bolles RC. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM, 1981, 24(6): 381–395.

13 McNames J. A fast nearest-neighbor algorithm based on a principal axis search tree. IEEE Trans. on, Pattern Analysis and Machine Intelligence, 2001, 23(9): 964–976.

14 Sproull RF. Refinements to nearest-neighbor searching ink-dimensional trees. Algorithmica, 1991, 6(1-6): 579–589.

15 Zhang N. Computing parallel speeded-up robust features (P-SURF) via POSIX threads. Emerging Intelligent Computing Technology and Applications. Springer Berlin Heidelberg, 2009: 287–296.

16 Chen P, Yang D, Zhang W, et al. Adaptive pipeline parallelism for image feature extraction algorithms. 41st International Conference on Parallel Processing(ICPP). 2012. IEEE. 299–308.

17 Warn S, Emeneker W, Cothren J, et al. Accelerating SIFT on parallel architectures. CLUSTER. 2009. 1–4.

18 Heymann S, Müller K, Smolic A, et al. SIFT implementation and optimization for general-purpose GPU. 2007.

19 Cornelis N, Van Gool L. Fast scale invariant feature detection and matching on programmable graphics hardware. IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops(CVPRW'08). IEEE. 2008. 1–8.

20 Terriberry TB, French LM, Helmsen J. GPU accelerating speeded-up robust features. Proc. Int. Symp. on 3D Data Processing, Visualization and Transmission (3DPVT). 2008. 355–362.

21 Havran V, Bittner J. On improving KD-trees for ray shooting. Proc. of Wscg Conference. 2002. 209–217.

22 Di Fatta G, Pettinger D. Dynamic load balancing in parallel KD-Tree K-Means. 2010 IEEE 10th International Conference on Computer and Information Technology(CIT). IEEE. 2010. 2478–2485.

23 Zhou P, Meng X. SAH based KD tree construction on hybrid architecture. 2011 Workshop on Digital Media and Digital Content Management (DMDCM). IEEE. 2011. 185–189.