

本科生毕业设计

|  |
| --- |
| 基于GPU的SSSP算法研究与优化 |

|  |  |
| --- | --- |
| 院 系 | 计算机科学与技术 |
| 专业班级 | CS1808 |
| 姓 名 | 马忠平 |
| 学 号 | U201814719 |
| 指导教师 | 张宇 |

2022年06月10日

**学位论文原创性声明**

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包括任何其他个人或集体已经发表或撰写的成果作品。本人完全意识到本声明的法律后果由本人承担。

作者签名： 年 月 日

**学位论文版权使用授权书**

本学位论文作者完全了解学校有关保障、使用学位论文的规定，同意学校保留并向有关学位论文管理部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权省级优秀学士论文评选机构将本学位论文的全部或部分内容编入有关数据进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于 1、保密囗，在 年解密后适用本授权书

2、不保密囗 。

（请在以上相应方框内打“√”）

作者签名： 年 月 日

导师签名： 年 月 日

摘 要

Abstract

目 录

[摘 要 I](#_Toc23945441)

[Abstract III](#_Toc23945442)

[1 绪 论 1](#_Toc23945443)

[1.1 课题背景 1](#_Toc23945444)

[1.2 国内外研究现状 2](#_Toc23945445)

[1.3 研究目的和主要内容 3](#_Toc23945446)

[1.4 论文结构 4](#_Toc23945447)

[1.5 课题来源 4](#_Toc23945448)

[2 方案论证（或具体背景技术概述） 5](#_Toc23945449)

[2.1 系统需求分析 5](#_Toc23945450)

[2.2 系统可行性分析 6](#_Toc23945451)

[2.3 开发工具分析及选择 6](#_Toc23945452)

[2.4 关键技术分析 6](#_Toc23945453)

[2.5 基本方案制定 7](#_Toc23945454)

[2.6 本章小结 7](#_Toc23945455)

[3 XXX系统设计 8](#_Toc23945456)

[3.1 功能需求 8](#_Toc23945457)

[3.2 系统总体设计 9](#_Toc23945458)

[3.3 功能模块设计 13](#_Toc23945459)

[3.4 本章小结 16](#_Toc23945460)

[4 XXX系统实现 17](#_Toc23945461)

[4.1 过滤器实现 17](#_Toc23945462)

[4.2 属性管理模块实现 18](#_Toc23945463)

[4.3 数据迁移模块实现 19](#_Toc23945464)

[4.4 本章小结 22](#_Toc23945465)

[5 性能测试与分析 23](#_Toc23945466)

[5.1 测试环境 23](#_Toc23945467)

[5.2 功能测试 23](#_Toc23945468)

[5.3 系统界面 23](#_Toc23945469)

[5.4 性能测试 23](#_Toc23945470)

[5.5 本章小结 25](#_Toc23945471)

[6 总结与展望 26](#_Toc23945472)

[致 谢 27](#_Toc23945473)

[7 毕业设计模板基本框架 28](#_Toc23945474)

[7.1 封面 28](#_Toc23945475)

[7.2 原创性声明页 29](#_Toc23945476)

[7.3 摘要 29](#_Toc23945477)

[7.4 目录 29](#_Toc23945478)

[7.5 参考文献 30](#_Toc23945479)

[7.6 附录 30](#_Toc23945480)

[7.7 毕业设计任务书 30](#_Toc23945481)

[7.8 成绩评定页 31](#_Toc23945482)

[8 毕业设计撰写要求 32](#_Toc23945483)

[8.1 图的格式 32](#_Toc23945484)

[8.2 表的格式要求 36](#_Toc23945485)

[8.3 公式 38](#_Toc23945486)

[8.4 流程图 39](#_Toc23945487)

[8.5 常见格式问题 39](#_Toc23945488)

[参考文献 41](#_Toc23945489)

[附录：大学期间发表或提交的论文 44](#_Toc23945490)

# 绪 论

## 课题背景

### 研究背景和趋势

社交网络正在变得无处不在，其数据量正在急剧增加。流行的在线社交网络网站，如Facebook，Twitter和LinkedIn，如今都拥有数亿活跃用户。谷歌的新社交网络Google+吸引了2500万独立用户，并在推出后的第一个月以每天约100万访问者的速度增长。实现这些海量图形的在线和交互式查询处理，特别是快速捕获和发现实体之间的关系，正在成为从社会科学到广告和营销研究再到国土安全的新兴应用不可或缺的组成部分。

最短路径计算是管理和查询社交网络最基本但最关键的问题之一。社交网络网站LinkedIn开创了著名的最短路径服务"你如何连到A"，它在3个步骤内表示出了你和用户A之间友谊链的精确描述。微软的Renlifang（EntityCube）[1]记录了超过1000万个实体（人员，位置，组织）的十多亿个关系，允许用户在距离小于或等于6时检索两个实体之间的最短路径。新出现的在线应用程序"Six Degrees"[2]提供了一种互动方式来展示您如何与Facebook网络中的其他人建立联系。此外，最短路径计算在确定信任和发现网络游戏中的朋友方面也很有用[3,4]。 另一方面，具有多个附加加速器的节点在高性能计算中逐渐崭露头角，由于图形加速处理器（GPU）的效率、并行性、可扩展缓存机制以及区别于通用计算机的专用高效率指令等在图形计算方面的优势，图形加速器的运用逐渐变得流行起来。由于单个GPU设备运算能力终究有限，为了扩展主机的计算能力，多GPU的节点成为一种有效的算力提升方式。

多GPU节点由一个主机（CPU）和多个图形加速器（GPU）组成，这些设备通过低延迟、高吞吐量的总线进行连接（参见图1），这些互联允许并行应用程序有效地交换数据，并利用GPU进行并行计算。

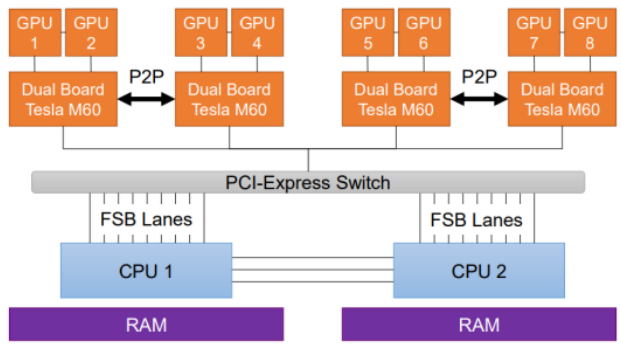


图 1 多GPU节点原理图

### 提高并行图计算的途径

多GPU节点通常使用以下两种方法之一进行编程：

1. 简单方法

简单方法之中，每个GPU都是单独管理的，每个GPU设备运行一个进程[5,6]或者使用批量同步并行的编程模型（BSP）[7]，其中的应用程序按照回合形式执行，每个轮次需要完成本地计算和全局通讯的过程[8,9]。该方式会受到来自各种源（如OS等）的开销影响，并且需要消息传递接口进行通信。另一方面，BSP模型可能会在实现基于轮次执行的全局程序中引入不必要的序列化。

上述两种编程方法都可能导致多GPU平台利用率的低下。特别是对于不规则应用程序（负载不平衡以及常发生不可预测通讯的程序），多GPU平台难以发挥其并发性优势，极端情况下，甚至同一时间只能有一台设备运行，其他设备则等待数据同步。

1. 异步编程模型

原则上，异步编程模型可以减少其中一些问题。与基于回合形式的通讯不同，处理器可以自主计算和进行异步通讯，而无需等待其他处理器达到全局同步。但是，很少有应用程序能够进行异步并发执行。开发异步并发程序时，需要深入了解底层体系结构和通讯网络，并且涉及对代码进行复杂的修整。

### 面临的问题和挑战

本课题从两个方向入手，分别从硬件、软件方面对单源点最短路径算法进行优化。

硬件方面拟采用多GPU加速器架构，采用良好并行性的硬件设备。软件方面拟使用尽量高效的算法，并且在算法涉及过程中更多考虑到充分利用硬件设备的可能性。硬件、软件以及软硬件的交互等方面都有着一些需要注意的难点。

硬件方面首先需要考虑多GPU的异步控制方式，其次还需要解决多GPU之间的高效率通讯的问题。软件方面需要尽量减低算法时间复杂度，尽量提高算法效率。另外，为了充分利用硬件的高并发性，很明显需要考虑对图数据进行拆分，以满足能执行并行处理条件，为了保证算法的正确性，图结构拆分后的合并处理也是需要解决的一大问题。

## 国内外研究现状

2012年，郭绍忠等人进行了基于GPU的并行Moore算法的初步研究，针对图形处理器(GPU)上的动态数据处理问题,在分析已有的并行单源最短路径(SSSP)算法的基础上,对GPU上的Moore SSSP算法进行并行化设计与实现。并行的Moore算法有效降低了空线程开销、访问开销以及同步时间[10]。

2013年Ruoming Jin等人完成了一种基于集线节点（具有大量边的点）为中心的新的最短路径的计算技术——Hub-Accelerator框架[11]，用于计算k度最短路径。该技术使我们能够通过大大限制集线点的邻域扩展范围或完全修剪搜索中的集线器（使用集线器标记方法）来显著减少搜索空间。Hub-Accelerator方法用于最短路径计算比BFS和最先进的近似最短路径方法Sketch快两个数量级以上。Hub-Network方法不会引入额外的索引成本和轻度的预计算成本;Hub2-Labeling的索引大小和索引构建成本也适中，算法效率优于或可与近似索引Sketch方法。

2018年，吴曼等人提出了割点与点割集的概念，将寻找割点与点割集的算法，与经典的Dijkstra算法结合，形成改进的并行算法并予以实现与应用，为寻找无向图的最短路径提供了理论依据，并用其改进了路由协议OSPF中的路由选择算法，降低了算法的时间复杂度[12]。

2019年，向志华等针对大规模网络中所有节点的全源最短路径的计算需求基于广度优先遍历（BFS）思想，在计算过程中设置存储队列，引入阻断路径，限制后续图节点的扩展范围，完成了图的减枝，大幅度降低最短路径计算的时间复杂。经测试，文中所设计的算法相较于传统Dijkstra算法在高、中、低规模的数据集上均可降低50%以上的运算时间；相较于BFS算法，可以降低20%以上的运算时间[13]。

2020年Tal Ben-Nun等人提出了用于不规则计算的异步多GPU的Groute编程模型，还实现了通用的集合操作和分布式工作列表，无需大量编程工作即可开发不规则的应用程序。该方法为8-GPU和异构系统上的一套不规则应用程序提供了强大的扩展能力，使某些算法的执行效率提高了7倍以上[14]。

陈智康等人针对在已知环境地图中的单个陆地移动机器人路径规划求解问题，将蚁群算法的信息素思想加入到经典Dijkstra算法中，实验结果表明，优化后的算法能够在很大程度上减少路径规划过程中产生的冗余点，减少机器人寻路的移动代价[15]。

2021年，曹大有等人通过利用遗传算法的框架模型建立了一个好理解、快速的最短路径问题求解过程[16]．

本文重点研究Groute编程模型和Hub-Accelerator编程框架。

### Groute编程模型

Groute —— 一种异步编程模型和运行时环境，可用于在多GPU系统上开发各种应用广泛的应用程序。基于底层网络的概念，Groute旨在降低多GPU和异构平台上异步应用程序的编程复杂性。Groute的通信结构很简单，但它们可用于有效地应用于从常规应用程序和BSP应用程序到特殊不规则算法的多类型程序。运行时环境的异步特性还促进了负载 平衡，从而更好地利用了异构多GPU节点。

（1）GPU间通信

GPU之间的内存传输可以由主机启动或设备启动。特别是，显式复制命令支持主机启动的内存传输（对等传输），而设备启动的内存传输（直接访问，DA）则使用不同GPU间的内存访问来实现。请注意，并非所有GPU对都可用对等内存的直接访问，具体取决于总线拓扑的结构。但是，可以从所有GPU访问特定的主机内存。

设备启动的内存传输通过虚拟寻址实现，虚拟寻址将所有主机和设备内存映射到单个地址空间。虽然DA比对等传输更灵活，但它对内存对齐、合并、活动线程数和访问顺序有高度敏感性。为了支持设备启动的传输，在无法访问彼此内存的GPU之间，可以执行两阶段间接复制。在间接复制中，源GPU首先将信息"推送"到主机内存，然后由目标GPU"拉取"，使用主机标志和系统范围的内存栅栏进行同步。在图1所示的拓扑中，GPU一次只能传输到一个目标。这阻碍了异步系统的响应能力，尤其是在传输大型缓冲区时。解决此问题的一种方法是当消息在网络中传输时将其切分为多个数据包。经过实验证明，使用分组内存传输而不是单个对等传输时，开销随着数据包大小的增加而线性减少，对于1-10MB数据包，开销范围在1∼10%之间，此参数可根据各个应用程序进行调整来平衡延迟和带宽。另外，直接（推送）和间接（推/拉）传输的传输速率也存在一定差异，分组设备启动的传输和细粒度控制是有效的，即使主机管理的分组对等传输也是如此。由于设备启动的内存访问是用户代码编写的，因此可以在传输过程中同时执行其他的数据处理。

多GPU通信的另一个重要方面是多个源/目标传输，就像在集合操作中一样。由于互连和内存复制引擎的结构，优化较差的应用程序可能会使总线拥堵。NCCL库[17]中使用的一种方法是在总线上创建环形拓扑。在这种方法中（如图3所示），每个GPU都传输到一个目标，通过直接或间接的设备启动传输进行通信。这可确保使用每个GPU的两个内存复制引擎，并充分利用总线。数据包化还用于在从上一个流水线操作接收信息时开始将信息传输到下一个GPU。

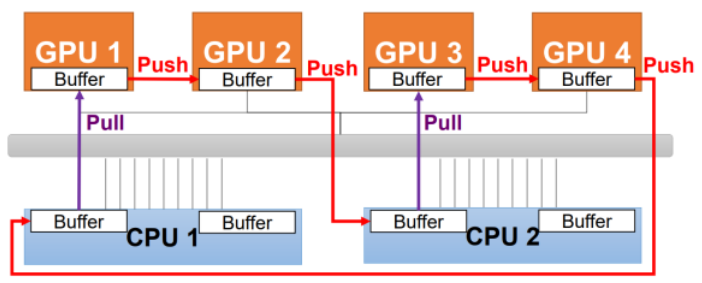


图 2 DA环拓扑技术

（2）Groute编程模型

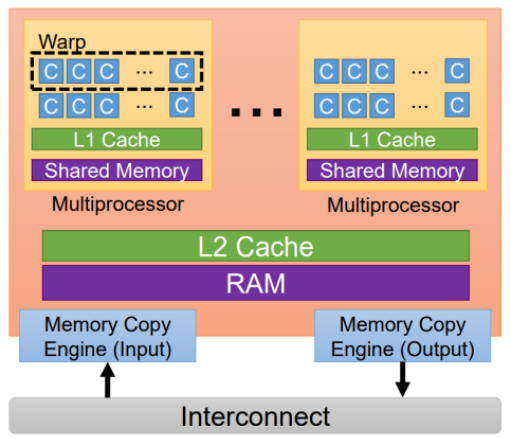


图 3 单个GPU的标准架构

单个GPU设备的结构如图3所示。每个 GPU都包含一组固定的多处理器 （MP）和一个RAM单元（称为全局内存）。GPU过程（内核）通过调度由多个线程组成的网格（分组为线程块）在MP上并行运行。在每个分配给单个 MP 的线程块中，Warp（通常由32个线程组成）在内核上同步执行。此外，同一线程块中的线程可以通过共享内存进行同步和通信，以及使用自动管理的L1和L2高速缓存。为了支持并发内存写入，在共享内存和全局内存上定义了原子操作。

内核调用和主机启动的内存传输通过命令队列（流）执行，这些命令队列可用于表达任务并行性。一个或多个GPU之间的流同步通常使用事件执行，这些事件在一个流上运行，并在另一个流上等待。

GPU调度程序按线程块分派内核，当没有更多线程块可供正在运行的内核调度时，通过调度其他内核的线程块，在同一GPU上实现多流并发。但是，高优先级流允许应用程序开发人员立即调用内核，先调度来自新内核的线程块，然后再调度来自正在运行的内核的线程块。

虽然流/事件构造提供了对内核调度的细粒度控制，但在编写涉及多个GPU的高级功能时会出现困难。我们列出了一个编程表1：Groute 编程接口抽象，它补充了现有模型，以简化多GPU开发，使用来自微基准测试得出的结论来最小化通信延迟。

表 1 Groute编程接口

|  |  |
| --- | --- |
| 构造 | 描述 |
| 基本构造 | |
| **Context** | 表示运行时的单例 |
| **Endpoint** | 可以通信的实体（例如 GPU、CPU、路由器） |
| **Segment** | 封装缓冲区、其大小和元数据的对象。 |
| 通讯设置 | |
| **Link**(  **Endpoint** src,  **Endpoint** dst,  **int** packetsize,  **int** numbuffers) | 将src连接到dst，使用多缓冲区和numbuffers缓冲区和数据包大小的数据包。 |
| **Router**(  **int** numinputs,  **int** numoutputs,  **RoutingPolicy** policy) | 多个端点连接在一起，实现动态通信。 |
| 通讯调度 | |
| **EndpointList RoutingPolicy**(  **Segment** message,  **Endpoint** source,  **EndpointList** routerdst) | 程序员定义的函数，用于根据发送终结点和路由器目标终结点的列表确定可能的消息目标。路由器将按可用性选择目的地。 |
| 异步对象 | |
| **PendingSegment** | 当前正在接收的段。 |
| **DistributedWorklist**(  **Endpoint** src,  **EndpointList** workers) | 管理由路由器和每 GPU 链路组成的全对全工作项分发。 |

Groute编程模型提供了多种构造来促进异步多GPU编程。表1列出了这些构造及其编程接口的简述。Groute应用程序的运行包括两个阶段：数据流图构造和异步计算。Groute程序首先指定计算的数据流图。这个有向图中的节点（我们称之为端点）表示（i）CPU和GPU等物理设备，或（ii）称为路由器的虚拟设备，它们是实现复杂通信模式的抽象节点。数据流图中的边缘表示端点之间的通信链路，只要没有自循环（端点直接连接到自身）或多图中的多个相同边（即路由器只能有一个到同一端点的传出边），就可以创建边。注意，为了支持多任务处理，可以从同一物理设备创建多个虚拟终结点。发送和接收方法允许端点在链路上发送和接收数据;收到数据后，端点可以使用回调对其执行操作。创建路由器时，程序员会指定路由策略，以确定接收到输入时的行为。例如，输入可以发送到单个终结点，也可以根据其可用性发送到终结点的子集。创建链路时，将指定该链路的分组和多缓冲策略。

### Hub-Accelerator编程框架

大规模社交网络中最短路径计算的核心问题来自Hub点（具有大量边的顶点）。与整个图网络规模相比，Hub点的数量可能很少，但是，它们几乎是任何点的邻接点。事实上，中心在局部子图中扮演着非常重要的角色。它们充当连接顶点之间最短路径的常见中介，就像航空公司航班小世界中的枢纽城市一样。事实上，理论分析表明，少量的Hub（由于幂律度分布）显着缩短了顶点之间的距离，并使大型的复杂网络显得更加“小”[18]。然而，枢纽是导致搜索空间爆炸的关键因素。对于Hub点的一步BFS将会极大扩展搜索的邻域空间，因此，Hub点的处理方式非常重要，充分利用Hub点的特性，可以大大简化搜索空间，进而提高搜索效率。

Hub-Accelerator框架是一种以Hub点为中心的新技术，该技术使我们能够通过大大限制Hub点的扩展范围（使用新颖的保持 距离的集线器网络概念）或完全修剪在线搜索中的集线器（使用Hub2标记方法）来显着减少搜索空间。Hub-Accelerator方法平均比BFS和最先进的近似最短路径方法快两个数量级以上，包括Sketch[19]，Tree Sketch[19]和Rigel Paths[20]。集线器网络方法不会引入额外的索引成本和轻度的预计算成本；Hub2-Labeling的索引大小和索引构建成本也适中，优于或可与近似索引Sketch方法相媲美。我们注意到，尽管最短路径计算已经过广泛研究，但大多数研究仅关注道路网络[21，22，23，24，25，26，27，28]或大规模社交网络上的近似最短路径（距离）计算[19，20]。据我所知，这是第一部明确解决这些网络中确切最短路径计算的工作。Hub-Accelerator技术也很新颖，距离保持子图（Hub-network）发现问题本身对于图的挖掘和管理具有理论和实践的重要性。

Hub-Accelerator框架的设计旨在利用枢纽点进行最短路径计算，而无需完全扩展其邻域。该技术解答了限制Hub点邻域扩展限制、高效利用Hub网络进行最短路径搜索等问题，使得利用Hub点加速最短路径搜索过程成为可能。该技术分为两个步骤：Hub网络发现：用于构建包含Hub节点的子图；基于Hub网络的双向BFS：算法的计算核心，与一般的BFS不同的是，它在遇到Hub点时，不会无限制的进行扩展。另外，另一项技术用于对上述方法进行进一步优化：Hub2标记技术，通过完全避免扩展任何集线器来进一步推动最短路径计算的速度边界。为了实现这一目标，使用更昂贵但通常负担得起的预计算和内存成本来加快在线搜索速度。

## 研究目的和主要内容

了解SSSP（单源最短路径）算法和GPU的基础相关知识，学习并了解现有的SSSP算法，基于现有SSSP算法，探索新的SSSP算法。

## 论文结构

随时调整

## 课题来源

# 引用

[1] <http://entitycube.research.microsoft.com>.

[2] <http://www.6-degreeonline.com>.

[3] Xiaohan Zhao, Alessandra Sala, Christo Wilson, Haitao Zheng, and Ben Y. Zhao. Orion: Shortest path estimation for large social graphs. In Proceedings of the 3rd Workshop on Online Social Networks (WOSN 2010), 2010.

[4] Xiaohan Zhao, Alessandra Sala, Haitao Zheng, and Ben Y. Zhao. Efficient shortest paths on massive social graphs. In Proceedings of 7th Internation al Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2011

[5] P.-Y. Hong, L.-M. Huang, L.-S. Lin, and C.-A. Lin. Scalable multi-relaxation-time lattice Boltzmann simulations on multiGPU cluster. Computers & Fluids, 110:1 – 8, 2015.

[6] E. Mej´ıa-Roa, D. Tabas-Madrid, J. Setoain, C. Garc´ıa, F. Tirado, and A. Pascual-Montano. NMFmGPU: nonnegative matrix factorization on multi-GPU systems. BMC Bioinformatics, 16(1):43, 2015.

[7] L. G. Valiant. A bridging model for parallel computation. Commun. ACM,33(8):103–111,1990.

[8] T. Ben-Nun, E. Levy, A. Barak, and E. Rubin. Memory access patterns: The missing piece of the multi-GPU puzzle. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC ’15, pages 19:1– 19:12. ACM, 2015.

[9] Y. Pan, Y. Wang, Y. Wu, C. Yang, and J. D. Owens. MultiGPU graph analytics. CoRR, abs/1504.04804, 2015. URL <http://arxiv.org/abs/1504.04804>.

[10] 郭绍忠,王伟,周刚,胡艳.基于GPU的单源最短路径算法设计与实现[J].计算机工程,2012,38(02):42-44.

[11] Ruoming Jin et al. Hub-Accelerator: Fast and Exact Shortest Path Computation in Large Social Networks[J]. CoRR, 2013, abs/1305.0507.

[12] 吴漫,白明丽,曾咏欣,蒋峰,利叶斌.基于点割集的最短路径算法的改进与应用[J].数学理论与应用,2018,38(Z2):18-32.

[13] 向志华,赖小平.基于BFS算法的有阻断路径的最短路径算法研究[J].信息通信,2019(11):41-42.

[14] Ben-Nun T,Sutton M,Pai S,et al.Groute: Asynchronous Multi-GPU Programming Model with Applications to Large-scale Graph Processing[J]. ACM Transactions on Parallel Computing, 2020, 7(3):1-27.

[15] 陈智康,刘佳,王丹丹,张运喜.改进Dijkstra机器人路径规划算法研究[J].天津职业技术师范大学学报,2020,30(03):30-35.10.19573/j.issn2095-0926.202003005.

[16] 曹大有,马斌.基于遗传算法的单源最短路径研究[J].汉江师范学院学报,2021,41(06):1-5.10.19575/j.cnki.cn42-1892/g4.2021.06.001.

[17] NVIDIA. NVIDIA Collective Communication Library (NCCL), 2016. URL <http://www.github.com/NVIDIA/nccl/>.

[18] Reuven Cohen and Shlomo Havlin. Scale-free networks are ultrasmall. Ph ys. Rev. Lett., 90, Feb 2003.

[19] Andrey Gubichev, Srikanta Bedathur, Stephan Seufert, and Gerhard Weik um. Fast and accurate estimation of shortest paths in large graphs. In Proc eedings of the 19th ACM international conference on Information and kno wledge management, CIKM ’10, pages 499–508, 2010.

[20] Xiaohan Zhao, Alessandra Sala, Haitao Zheng, and Ben Y. Zhao. Efficient shortest paths on massive social graphs. In Proceedings of 7th Internation al Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2011.

[21] N. Jing, Y. Huang, and E. A. Rundensteiner. Hierarchical encoded path vie ws for path query processing: An optimal model and its performance evalu ation. TKDE, 10(3):409–432, 1998.

[22] S. Jung and S. Pramanik. An efficient path computation model for hierarc hically structured topographical road maps. TKDE, 14(5):1029–1046, 200 2.

[23] S. Shekhar, A. Fetterer, and B. Goyal. Materialization trade-offs in hierarc hical shortest path algorithms. In SSD ’97, 1997.

[24] P. Sanders and D. Schultes. Highway hierarchies hasten exact shortest pat h queries. In 17th Eur. Symp. Algorithms (ESA), 2005.

[25] H. Bast, S. Funke, P. Sanders, and D. Schultes. Fast Routing in Road Net works with Transit Nodes. Science, 316:566–, April 2007.

[26] R. J. Gutman. Reach-based routing: A new approach to shortest path algor ithms optimized for road networks. In ALENEX/ANALC, pages 100–111, 2004.

[27] A. V. Goldberg and C. Harrelson. Computing the shortest path: A search m eets graph theory. In SODA ’05, 2005.

[28] H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In SIGMOD’08, 2008.