doi:10.3772/j.issn.1002-0470.2018.01.001

NJ-GPCA:一种面向并行空间计算的高效数据访存策略①

姚 晓②*** 邱 强* 肖茁建*** 方金云*

(*中国科学院计算技术研究所 北京 100190)

(**中国科学院大学 北京 100190)

摘 要 针对并行矢量空间叠加分析中存在的I/O性能差及并行算法调度效率低的缺陷,提出了"去"归并通用并行计算架构(NJ-GPCA)。该架构首先基于内存数据库Redis设计内存矢量空间数据模型;其次通过数据预处理以及任务分发技术,减少进程等待,提高I/O性能;最后重新进行任务分配以及规划进程调度,避免结果数据归并收集,使得并行叠加分析算法归并收集阶段的时间复杂度由 $O(n\log n)$ 降低到 $O(n)$。实验结果表明,该方法对真实地理数据下的并行叠加分析操作,I/O时间至少减少75%,对于提高算法整体性能有明显效果。

关键词 空间叠加分析, I/O, 并行计算, 任务调度

0 引 言

地理信息系统(geographic information system, GIS)是一种处理地理空间数据的信息系统,已经在政府部门、企业、公共信息服务等各个领域得到广泛应用[1]。空间分析作为GIS的核心基本功能,在城市规划与管理、空间选址、水污染检测、灾害分析、地形地貌分析等领域都占据着不可代替的地位。

GIS空间分析的核心算法之一是矢量空间叠加分析算法,该算法是指在统一空间参考系统下,通过将两个空间数据进行的一系列叠加运算,产生新数据的过程。矢量空间叠加分析算法具有计算密集和数据密集的双重特征。近年来,随着移动互联时代的到来,空间数据开始呈现形式多样化、规模海量化的趋势,而且随着并行编程技术、分布式计算的发展和多核处理器性能的不断提升,空间分析算法并行化研究也取得了很大的进展。但是,随着并行空间叠加分析算法的不断优化,算法性能的不断提升,新的性能瓶颈,即算法的I/O开销,也慢慢凸显出来。无论数据源是存储于文件系统还是关系型数据库系统,存储介质都是基于传统的磁盘。因此,受制于磁盘I/O读写性能的限制,随着算法计算时间的不断优化,数据I/O的比重会越来越大,从而导致空间分析算法的性能很难得到进一步提升。本研究借助于内存数据库Redis,设计并实现了一套全内存的用于并行矢量空间叠加分析的"去"归并通用计算架构(no join-general parallel computing architecture, NJ-GPCA),优化了并行矢量空间分析算法的并行架构,通过减少并行空间叠加分析算法的通讯量以及降低并行调度成本,解决了算法中通讯、I/O等多个影响算法效率的关键问题,其主要贡献包括以下三点:(1)设计了一种基于Redis的内存矢量空间数据模型,该模型能够支持内存矢量空间数据存储管理及并行化操作,提供基于Redis的矢量空间数据的快速存取。(2)实现了一种针对NJ-GPCA的矢量空间数据预处理以及任务分发技术,使得各个子进程可以直接获取所需数据,进行叠加运算,减少进程等待,提高算法I/O效率。(3)实现了一种"低"通讯

① 国家重点研发计划(2016YFB0502300,2016YFB0502302)资助项目。

② 男,1990年生,博士生;研究方向:并行空间分析,空间数据存储与管理,搜索引擎技术等;联系人,E-mail: yaoxiao@ict.ac.cn

(收稿日期:2017-08-25)

量进程调度模型,有效避免并行叠加分析算法的数据归并收集。

1　研究现状

1.1　并行矢量空间分析

并行矢量空间分析是 GIS 矢量空间分析方法与并行计算的结合,致力于通过空间分析算法的并行化解决空间数据处理的速度与质量问题[2]。

目前,并行空间分析算法主要有两种研究思路:细粒度的算法并行和粗粒度的任务并行。朱效民[3]基于 OpenMP 实现了线段求交以及点面叠加并行算法,通过数据排序、利用动态调度,以及改进的并发内存分配技术,取得了较为理想的线性加速比。但该方法由于内存分配在计算密集型算法流程中占得比重较小,因此并发内存对于算法的优化不是很明显,此外该研究路线只能依据具体算法设计相应的并行化策略,通用性不高。Zhao[4]等将反距离加权插值算法映射到统一计算设备架构(CUDA),使得空间插值算法达到 40 倍以上的加速比,但随着数据量的增长,总线带宽成为限制算法性能提升的主要瓶颈。Puri[5]等基于消息传递接口(message passing interface,MPI)实现了多核集群环境下的多边形裁剪算法,并将其应用于 MPI-GIS 系统,算法的运行效率可以达到原算法的 44 倍。Qiu[6]等采用 MPI,基于 fork-join 的并行计算架构,采用动态树形归并算法以及负载均衡技术在多核的集群环境下实现了并行扫描线算法,使得算法数据收集阶段的时间复杂度由 $O(n^2)$ 降低到 $O(n\log n)$,算法性能得到极大提高。该模型如图 1所示。

该并行架构存在两个问题:一方面任务数越多,消息传递次数越多,通讯指令 I/O 开销越高;另一方面数据归并收集时,子进程需要将结果数据传送给另一个子进程,使得通讯数据 I/O 开销偏高(注:本文对进程间数据通讯做了如下细分:进程之间指令的传输称作通讯指令 I/O,进程之间计算结果的传输称作通讯数据 I/O)。此外,主进程任务划分不均衡也会造成数据 I/O 成本增加以及算法总体性能的下降。

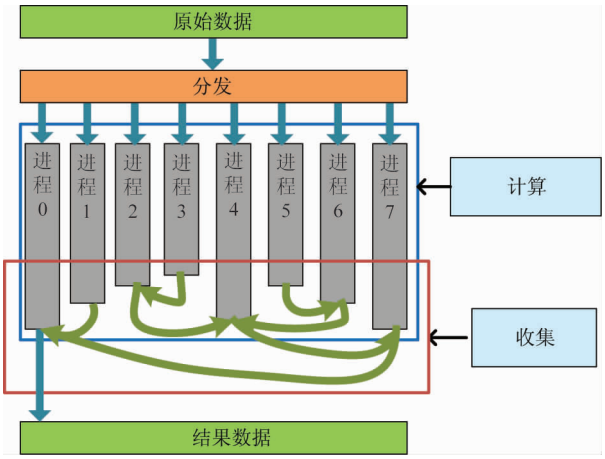

图 1　fork-join 架构

1.2　内存数据库的空间扩展

计算机硬件性能的不断提升,以及 64 位操作系统的出现极大地推进了内存数据库的发展。内存数据库如 Redis,存取数据都是基于内存,因此不存在外存 I/O,具有很高的读写性能[7]。

目前国内外对于空间数据库的研究大都集中于传统的关系型数据库,如 Oracle Spatial、PostGIS 等,受限于磁盘读写的性能瓶颈,很难在实时性方面有重大突破。在内存数据库的空间扩展方面,相关研究主要围绕空间索引、查询等进行展开。Chen 等[8]实现了一种 pre-execution prefetching 的缓存策略,打破了计算和 I/O 并发的壁垒,极大地减少了并行算法的 I/O 延迟。朱进[9]设计了基于 Redis 的矢量地理数据库的分层组织模型以及实现了基于 Redis 的矢量数据引擎原型系统,实验结果显示该引擎响应速度高,并发性能好。但该研究仅仅对于空间查询表现出了超高的性能,对于并行空间分析并没有提供很好的支持。雷德龙[10]基于 MongoDB,在三层式云存储架构基础上,设计并实现了一套矢量空间数据云存储与处理系统,该系统取得了良好的读取性能和海量数据处理性能,但并发写入性能仍然是该系统的主要性能瓶颈。

2　“去”归并通用并行计算架构

如图 1 所示,传统的基于 fork-join 架构的并行矢量叠加算法,主进程接到输入参数,首先进行数据

划分,然后向子进程发送计算指令,各个子进程获得指令后开始读取数据,进行相应计算,完成后向主进程发送计算完成的消息,主进程将发送归并指令给最先完成计算的两个子任务,子任务进行相应的结果归并,随后各个完成计算的进程依次两两归并,最后结果汇总到主进程,由主进程输出。

"去"归并通用并行计算架构(NJ-GPCA)如图 2 所示,基于该架构的并行叠加分析算法可以利用内存数据库 Redis 进行矢量空间数据的快速按需读写,减少并行计算过程中的进程通讯成本,避免数据归并收集。

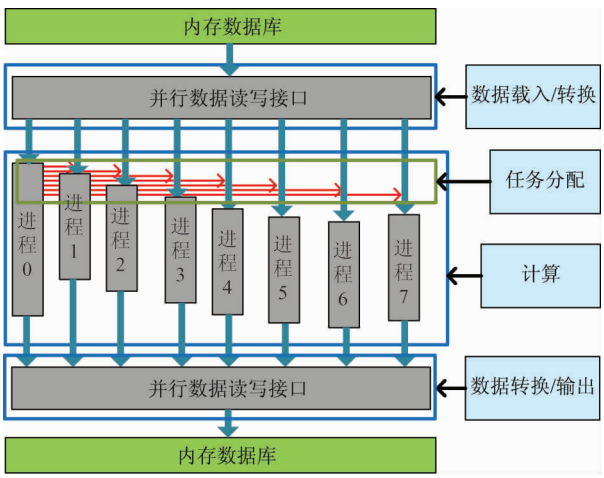

图 2　NJ-GPCA

2.1　内存矢量空间数据模型

矢量空间叠加分析算法属于数据密集型计算,特点是进行地理计算时需要分析全部数据。空间分析典型的过程是一次性读入原始数据,计算过程不修改输入数据,分析得到的结果数据一次性输出。简化的矢量图层包含元数据、属性表定义、几何要素数据以及属性记录数据 4 个部分,其中元数据、属性表定义和属性记录数据这三者在矢量空间叠加算法中不参与运算,结果数据直接通过继承输入数据得到,而几何要素数据要通过分析计算得来。本文设计了矢量空间数据的分块存储模式。

输入数据

对于输入数据,由于各个子进程在主进程进行数据划分之前无法估计各自的计算任务,对于要计算的数据块也是未知,将输入数据一次性全部取出不仅增大 I/O 消耗,也会造成内存浪费。因此,需要将矢量空间数据按照空间填充曲线划分之后分块存储。图 3 所示为内存矢量空间数据结构图,Key 中 M 表示元数据,存储参考系统,图形几何类型等信息,采用 Redis 中的 Hash 存储;ATD 表示属性表定义,采用 Redis 中 Set 数据结构存储;R 表示属性信息表,采用 Redis 中的 Hash 存储;F 表示要素集合信息,HID 为 Hilbert 网格编码,其值为 OGC[11] 规定的用于描述空间要素集合信息的标准格式 WKB(Well-Known Binary),采用 String 存储。

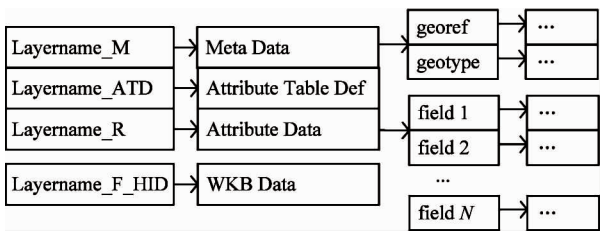

图 3　内存矢量空间数据结构

输出数据

对于输出数据而言,元数据、属性表定义以及属性记录数据这三者继承自输入数据,因此存储模型不变。对于几何要素数据,由于各个子进程计算得到的要素几何信息需要并行输出,而单一的 key 会造成数据写冲突以及进程等待,为了更好地满足矢量叠加分析算法的并发写需求,本文设计了如图 4 所示的数据结构,O 表示叠加分析结果,P 表示进程号,采用 Redis 中的 Hash 数据结构存储,size 表示该 WKB 的大小。每个子进程将各自的运算结果直接写入相应的 key 中。当需要进行可视化或者持久化存储的时候,数据导出工具可以通过匹配 key 中的统一前缀将数据合并并导出到外存。

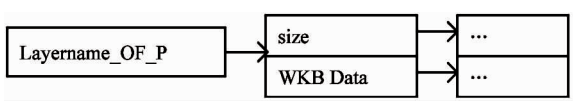

图 4　几何要素输出结构

2.2　数据预处理和任务分发

为了保证算法各个任务的负载均衡性,本文的数据划分采用基于 Hilbert 空间填充曲线的数据划分策略[12],低阶划分块(大区域)可以通过高阶划

分块(小区域)编码的前缀模糊匹配获得。由于 Redis 大量的模糊匹配会带来大量 CPU 占用,进而降低其性能,因此,矢量空间数据在入库存储时,不能划分过多的数据块。本文方法在矢量空间数据载入 Redis 前,首先按照 3 阶 Hilbert 曲线进行粗划分,并将各个数据块的 Hilbert 网格二进制编码作为要素集合 Key 的后缀。

空间分析算法需要将存储在硬盘的空间数据转换为计算任务所需的内存对象格式。因此各进程在数据载入阶段存在外存矢量空间数据到具体算法所需的内存空间对象的载入与转换过程,该过程是非常耗时的,是主要的 I/O 性能瓶颈之一。传统的 fork-join 模式是基于外存中的 Shapefile 文件作为输入源,主进程首先将文件全部读入内存,然后在进行数据划分时,各个子进程是处于等待的状态,待接到计算任务之后,各个子进程将文件全部读入内存,完成数据载入及转换,并从中选取相应的数据进行计算。NJ-GPCA 架构中,基于内存数据库 Redis 的并行读取能够实现按需读取,加快矢量空间数据的访问速度。

如图 2 所示,首先主进程(进程 0)进行分块数据载入、数据划分(此过程耗时很短)以及任务分发调度;然后子进程(进程 $i, i>0$)接收到的任务集之后,通过空间要素集合 Key 前缀匹配过滤所需的数据进行计算。

2.3 低通讯并行调度模型

在基于 fork-join 架构算法的数据归并收集阶段,结果数据由主进程进行输出,因此通讯数据 I/O 在整个算法过程中所占比重比较高。尤其是在跨网络的集群环境中,随着进程数的增加,中间结果数据可能需要经过多次网络传输,最终到达主进程所在的内存空间,由主进程进行最终结果的输出。因此,为了消除通讯数据 I/O 的影响,本文对并行矢量叠加分析算法流程进行重新设计,如图 5 所示。

该架构重新进行任务分配及调度,主进程接到任务后通过载入数据进行任务划分,任务调度和非几何属性的继承;子进程接到任务后,读取相应的数据进行计算,各子进程计算任务完成后不进行结果归并,而是直接将计算结果输出至内存数据库,并向

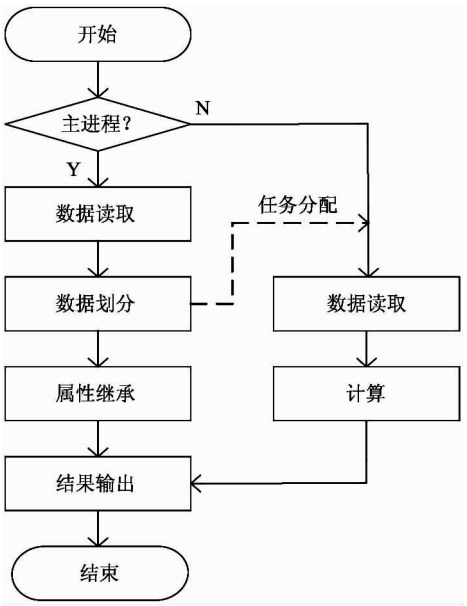

图 5 并行调度流程图

主进程发送计算完成指令。最终,主进程在完成所有属性继承,并且收到所有子进程的结果输出完成指令后,整个计算任务结束。

此外,该调度模型可以有效避免通讯数据 I/O 带来的性能损耗。在 fork-join 架构下的数据收集阶段,各进程需要将计算结果通过两两归并的方式进行汇总输出,该阶段算法的时间复杂度为 $O(n\log n)$ [6]。假设并行算法进程数为 k, 则 fork-join 架构下数据收集阶段的时间复杂度 $T(n)$ 满足:

$$T(n) = kT(\frac{n}{k}) + O(n) \tag{1}$$

此时满足分治归并算法主定理[13]情况 2,所以

$$T(n) = O(n^{\log_k^k}\log n) = O(n\log n) \tag{2}$$

在 NG-JPCA 架构下,算法不需要将结果数据两两归并收集再输出,各个子进程可以直接将结果数据并行输出,即式(1)中 $kT(\frac{n}{k})$ 项不存在,因此其归并收集阶段时间复杂度为

$$T(n) = O(n) \tag{3}$$

3 实验与分析

3.1 实验环境

本文采用的硬件环境为:CPU 为 Intel Xeon,主频为3.4GHz,共 8 核;内存为 4GB DDR3 1333MHz

ECC;外存为 500GB 7200RPM SATA 硬盘;采用千兆以太网连接各个节点。

软件环境:操作系统为 64 位 CentOS 专业版;MPI 采用 OpenMPI 版本;Redis 版本为 2.6.10;集成开发环境为 Eclipse + CDT;采用 GCC 作为编译器。

3.2　数据 I/O

数据 I/O 优化效果的验证实验中,本文选取面面叠加求交算法为例,设计了两组对比实验,分别用于验证不同进程数以及不同数据规模下本文方法与 fork-join 架构下算法的 I/O 性能对比,并以:

$$\frac{T_{I/O:fork-join} - T_{I/O:NG-GPCA}}{T_{I/O:fork-join}} \cdot 100\% \tag{4}$$

作为 I/O 性能优化程度的评价标准,其中 $T_{I/O} = T_{read} + T_{write}$。

实验 1 选取全国土地利用和县界行政图,在 8 核环境中通过设置不同进程数,NJ-GPCA 架构下算法与 fork-join 架构下算法的 I/O 与通讯时间对比如图 6 所示。图 6(a)和图 6(b)分别为读写性能对比,相较于 fork-join 架构,本文方法的 I/O 效率有着明显的提升,以 2 进程的叠加求交分析算法为例,I/O时间减少 75%,并且随着进程数的增加,读写性能有进一步提升的趋势。fork-join 架构下算法存在数据归并收集阶段,各个进程的计算结果需要经过多次进程通讯进行数据汇总,因此如图 6(c)所示,随着进程数的不断增多,通讯时间会越来越长。由于本文方法不存在数据归并收集阶段,只存在耗时很少的通讯指令 I/O,因此使得通讯时间在 NJ-GPCA 架构下所用时间几乎为零。

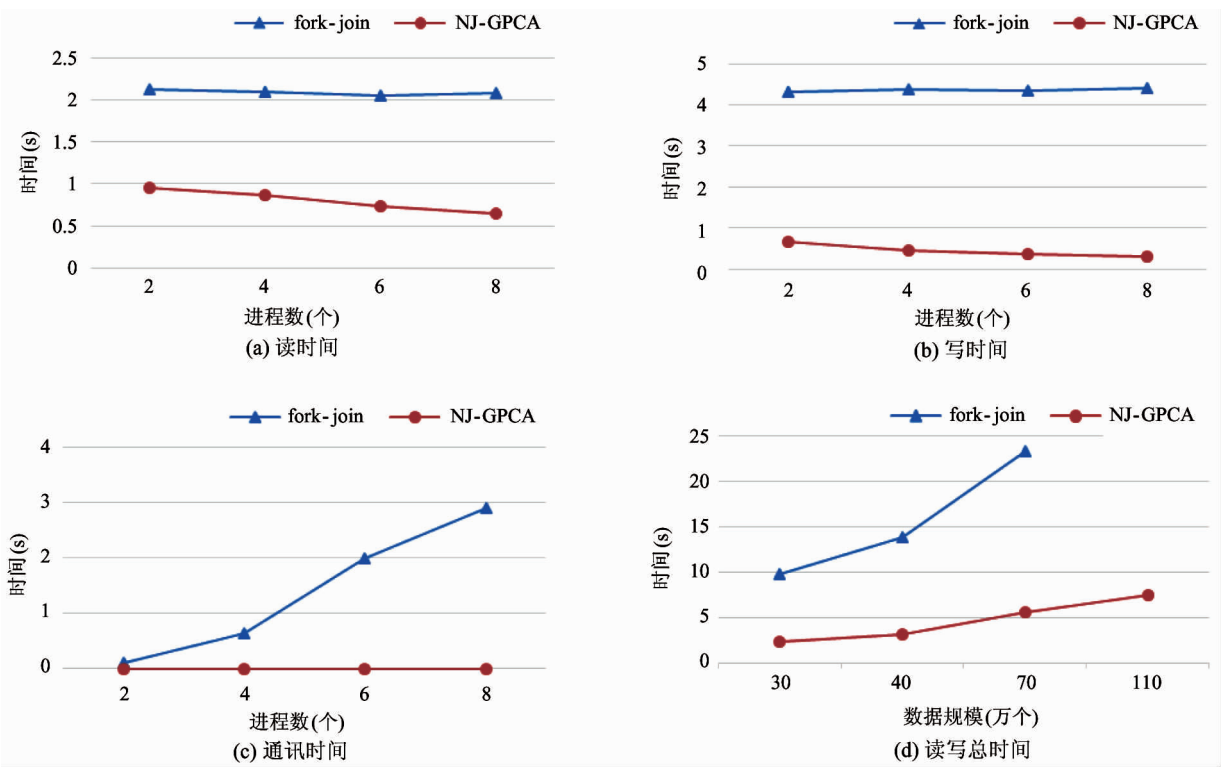

图 6　算法 I/O 性能对比

通过实验 1 得知,随着进程数的增加,NJ-GPCA 架构下算法 I/O 效率优化效果越来越明显,因此,实验 2 在 2 进程的条件下,通过选取不同数据规模的河流数据和县界行政图,进行叠加求交分析测得算法读写总时间。如图 6(d)所示,在 30 万、40 万、70 万数据规模下,NJ-GPCA 架构下的 I/O 效率比 fork-join 架构下分别提升75.87%、76% 和75.7%,并且在 110 万数据规模时,fork-join 架构下的算法出现运行失败的情况,分析可知,是由于各进程接到计算命令后读入全部数据,导致内存耗尽,程序退出。实验数据见表 1。

表 1　实验数据

数据名称	数据类型	要素个数	数据说明	操作说明
ict _ landuse. shp	面	122552	全国土地利用数据	实验 1、3 输入数据
ict _ counties. shp	面	2449	中国县界行政图	实验 1、2、3 叠加数据
River _ 300000. shp	线	303711	河流 30 万抽吸数据	实验 2 输入数据
River _ 400000. shp	线	405130	河流 40 万抽吸数据	实验 2 输入数据
River _ 700000. shp	线	700742	河流 70 万抽吸数据	实验 2 输入数据
River. shp	线	1102257	河流完整数据	实验 2 输入数据

3.3　整体效率

本文通过实验 1 和实验 2 验证了 NJ-GPCA 架构下算法 I/O 性能有着明显的优化效果，因此实验 3 选取全国土地利用和县界行政图进行叠加分析算法总时间的实验对比。对算法的总体性能优化实验结果如图 7 所示，本文方法也表现出了很好的效果，在 2、4、6、8 进程的实验对比中，NJ-GPCA 架构下的算法总时间比 fork-join 架构下的算法总时间分别减少18.53%、25.36%、26.73%和33.23%。可以看出，随着进程数的增多，NJ-GPCA 架构的优势会越来越明显。fork-join 架构下的算法从 6 进程到 8 进程出现了算法总时间增加的趋势，结合图 6(c)以及算法的数据划分分析可知，一方面通讯 I/O 的增加导致算法总时间的增加，另一方面，基于 Hilbert 空间填充曲线的数据划分策略仍然难以实现完全的负载均衡，数据划分不均导致某些进程任务耗时比较长，因而也会加大算法的总时间。而本文方法由于 I/O 时间的优化以及避免了数据归并收集，抵消了部分由数据划分不均带来的部分性能损耗，使得算法总时间仍有进一步的优化。

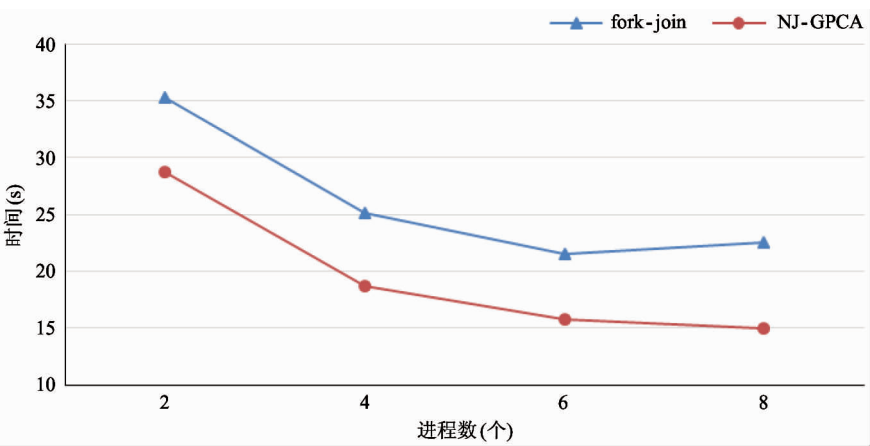

图 7　算法总体性能对比

4　结　论

算法并行化和 I/O 优化是提高传统空间分析算法性能的重要手段。本文提出了“去”归并通用并行计算架构(NJ-GPCA)，减少了并行叠加分析算法进程等待及通讯成本，避免了结果数据归并收集。实验结果表明，本文方法对于并行矢量空间分析算法的 I/O 以及总体性能提升有着显著的效果。

本文提出的 NJ-GPCA 架构通用性较强，对于空间数据挖掘，空间大数据分析等其他领域算法的并行化研究同样具有一定的启发意义。尤其在应对空间大数据、数字地球和智慧城市的建设过程中，本文的方法具有一定的实践意义。本文在探索并行矢量空间分析的过程中发现数据划分均衡性也是影响算法性能的重要瓶颈。此外，栅格数据也同属 GIS 中

比较重要的数据格式,与人们的生活息息相关。因此,数据负载均衡性和栅格数据内存模型的构建及其并行化分析将是下一步工作的研究重点。

参考文献

[1] Chang K T, 陈健飞. 地理信息系统导论[M]. 北京:科学出版社, 2003. 4-5

[2] 王结臣, 王豹, 胡玮, 等. 并行空间分析算法研究进展及评述[J]. 地理与地理信息科学, 2011, 27(6): 1-5

[3] 朱效民, 潘景山, 孙占全,等. 基于 OpenMP 的两个地学基础空间分析算法的并行实现及优化[J]. 计算机科学, 2013, 40(2): 8-11

[4] Zhao Y, Qiu Q, Fang J, et al. Fast parallel interpolation algorithm using CUDA[C]. In: Proceedings of Geoscience and Remote Sensing Symposium, Me Lbourne, Australia, 2014. 3662-3665

[5] Puri S, Prasad S K. A parallel algorithm for clipping polygons with improved bounds and a distributed overlay processing system using MPI[C]. In: Proceedings of IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Shenzhen, China, 2015. 576-585

[6] Qiu Q, Zhu X, Yao X, et al. A parallel strategy for plane sweep algorithm in multi-core system[C]. In: Proceedings of 2013 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), Melbourne, Australia, 2013. 3642-3645

[7] 王珊, 肖艳芹, 刘大为,等. 内存数据库关键技术研究[J]. 计算机应用, 2007, 27(10): 2353-7

[8] Chen Y, Byna S, Sun X H, et al. Hiding I/O latency with pre-execution prefetching for parallel applications[C]. In: Proceedings of IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, 2008. 1-10

[9] 朱进, 胡斌, 邵华,等. 基于内存数据库 Redis 的轻量级矢量地理数据组织[J]. 地球信息科学学报, 2014, 16(2): 165-72

[10] 雷德龙, 郭殿升, 陈崇成,等. 基于 MongoDB 的矢量空间数据云存储与处理系统[J]. 地球信息科学学报, 2014, 16(4): 507-16

[11] Open Geospatial Consortium. OGC standards and supporting documents[EB/OL]. http://www.opengeospatial.org/standards: OGC, 2013

[12] 邱强, 方雷, 姚晓,等. 基于空间聚类的矢量空间数据并行计算划分方法[J]. 高技术通讯, 2015, 25(4): 327-33

[13] Cormen T. 算法导论[M]. 北京:机械工业出版社, 2013. 93-94

NJ-GPCA: a fast data I/O strategy for parallel spatial computing

Yao Xiao[*,**], Qiu Qiang[*], Xiao Zhuojian[*,**], Fang Jinyun[*]

([*]Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([**]University of Chinese Academy of Sciences, Beijing 100190)

Abstract

A parallel computing strategy named no join-general parallel computing architecture (NJ-GPCA) is proposed to solve the spatial overlay analyzing technique's drawbacks of poor I/O performance and low parallel task scheduling efficiency. Firstly, the strategy puts the vector spatial data into the database of Redis with the new structure, and then cuts down the processes wait time and improves the efficiency of I/O by data pre-processing and task distributing. Finally, the strategy uses a new task allocation and task scheduling to avoid result collection, which reduces the time complexity of the algorithm's collection stage to $O(n)$ from $O(n\log n)$. The experimental result shows that the strategy can reduce the I/O time by at least 75% and significantly improve the efficiency of the algorithm.

Key words: spatial overlay analysis, I/O, parallel computing, task scheduling