CPU-MIC异构并行架构下基于大规模频繁子图挖掘的药物发现算法

彭绍亮[1]，牛琦[1]，李肯立[1]，邹权[2]
1. 湖南大学信息科学与工程学院，湖南 长沙 410082；
2. 电子科技大学基础与前沿研究院，四川 成都 610054

摘要
频繁子图挖掘是许多实际应用领域中需要解决的重要问题，由于计算密集性、挖掘的图集及其结果容量大，现有的频繁子图挖掘方案无法满足时间需求，其处理效率是目前面临的主要挑战。原创性地提出了并行加速的频繁子图挖掘工具cmFSM。cmFSM主要在3个层次上进行并行优化：单节点上的细粒度OpenMP并行化、多节点多进程并行化和CPU–MIC协作并行化。在单节点上cmFSM的处理速度比基于CPU的最佳算法快一倍，在多节点方案中cmFSM提供可扩展性。结果表明，即使只使用一些并行计算资源，cmFSM也明显优于现有的最先进的算法。这充分表明提出的工具在生物信息学领域的有效性。

关键词
频繁子图挖掘；生物信息学；并行算法；内存约束；同构；集成众核

中图分类号：TP31　　文献标识码：A　　doi: 10.11959/j.issn.2096-0271.2019016

A scalable CPU-MIC coordinated drug-finding tool by frequent subgraph mining

PENG Shaoliang[1], NIU Qi[1], LI Kenli[1], ZOU Quan[2]
1. College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China
2. Institute of Fundamental and Frontier Sciences, University of Electronic Science and Technology of China, Chengdu 610054, China

Abstract
Frequent subgraph mining is an important issue to be solved in many practical fields. Due to the computational intensiveness, the mining of the atlas and the large capacity of the results, the existing solutions can not meet the time requirements, and its efficiency is currently the main challenge. The frequent subgraph mining tool cmFSM for parallel acceleration was originally proposed. cmFSM performs parallel optimization on three levels: fine-grained OpenMP parallelization on a single node, multi-node multi-process parallelization and CPU-MIC collaborative parallelization. cmFSM is twice as fast as the best CPU-based algorithm on a single node and provides scalability in a multi-node approach. In the future, we will continue to improve the scalability of multiple solutions. The results show that even with only a few parallel computing resources, cmFSM is significantly better than the most advanced algorithms available. This fully demonstrates the effectiveness of the proposed tool in the field of bioinformatics.

Key words
frequent subgraph mining, bioinformatics, parallel algorithm, memory constraints, isomorphism, many integrated core

1 引言

在生物信息学研究中，为帮助在药理学化合物数据库或生物网络的核心功能结构中寻找新药，频繁子图挖掘的解决方案尤为重要。但现有的频繁子图挖掘方案无法有效解决内存消耗与时间需求问题。Lin W等人[1]认为频繁子图挖掘问题可分为两个方面：一方面是在一个图的不同区域挖掘子图，适用于社交网络分析领域；另一方面是在大规模图集中挖掘子图，适用于计算药理学和生物信息学领域。两方面都面临共同的问题：大规模挖掘任务产生的大量挖掘结果超过了单台机器的存储器空间，且无法满足时间需求。并行技术是解决这类问题的有效方案。针对第一方面，专家已经提出基于串行CPU技术[2-4]、并行计算框架（MapReduce、MPI、Spark）[5-8]及图形处理器（graphics processing unit，GPU）[9]的解决方案。本文意在解决生物信息学领域中频繁子图挖掘的问题。

2 相关工作

现有频繁子图挖掘方案以递归计数为基础，可以挖掘出所有频繁子图，且大部分频繁子图挖掘算法基于广度优先搜索（breadth-first searc，BFS）改进生成，例如基于先验的挖掘算法（apriori-based algorithm for mining，AGM）[11]和频繁子图挖掘（frequent subgraph discovery，FSG）算法工具[12]。但是深度优先搜索（depth-first-search，DFS）内存需求更低、性能更优。Meinl T等人[13]对一些典型的DFS算法（如MoFa[14]、FFSM[15]、gSan[16]和Gaston[17]）进行了定量比较和详细比较，发现遇到大规模挖掘任务时，它们很难满足时间需求。值得注意的是，它们都是单核串行版本。此外，Buehrer G等人[18]提出了多核系统中的并行挖掘策略，并在多个内存处理器间划分挖掘任务。这些解决方案充分利用了单节点上的机器资源实现加速挖掘。然而，这些方案都是基于内存的，并假设内存空间适配于图集和挖掘规模。但是，随着数据量的增加以及支持度的降低，挖掘规模呈指数递增，内存空间不再适配。为解决这个问题，Wang C等人[19]和Nguyen S N等人[20]基于磁盘分别提出ADI-Mine算法和数据分区方法。但是这类方案又面临着访问数据的巨大开销问题。

Hill S[11]等人基于MapReduce[21]框架提出了IFSM算法。该算法首先计算图表集合中的所有频繁子图的支持度。其次，排除未达到设定支持度的频繁子图。与IFSM[10]类似，FSM-H[22]和mrFSM[23]也是通过迭代方法在MapReduce框架上开发的，且更加关注每次迭代中的负载平衡。然而MapReduce框架不适合迭代计算，可能会产生大量I/O和序列化开销，因此基于MapReduce的这些方案仍然会产生严重的性能问题。MapReduce框架上性能更为出色的是MRFSM[24]。MRFSM不采用迭代方法，而是根据支持度智能化地过滤频繁子图，再将所有频繁子图分配给所有机器进行挖掘，并整合最终结果。因为没有迭代，所以它能提供比IFSM、FSMH和mrFSM更好的性能。但是MRFSM使用标准I/O和数据间的交互，其性能受到严格限制。此外对于大规模挖掘任务，每台机器上会产生严重的存储压力，MRFSM无法满足时间需求。

针对大规模挖掘问题，笔者提出了cmFSM算法。该算法分别在3个方面实现了并行化：单节点上的细粒度OpenMP（共享存储并行编程）并行化、多节点多进程并行化和CPU-MIC协作并行化。

3 算法介绍

cmFSM算法实现了多级别和多粒度的并行性，并以集成众核（many integrated core，MIC）为加速器，使用OpenMP实现多线程，目的是解决药物发现过程中大规模频繁子图挖掘的时间需求和内存限制等问题。信息传递接口（message passing interface，MPI）基于4种静态任务划分策略和一种基于监管的动态任务划分策略，实现最佳负载平衡。此外，在传输模式下使用MIC仅传输双边频繁子图，并备份复杂数据结构，以避免过度传输造成的瓶颈。通过充分利用MIC的多核计算能力，可以在CPU和MIC协作的情况下达到预期的执行速度。

3.1 单节点上的OpenMP并行化

3.1.1 并行化策略

基于DFS的频繁子图算法难以控制并行粒度，且无法控制递归过程，造成程序一直调用函数的后果，从而产生大量挖掘结果，这必然会导致不同挖掘任务间的负载不平衡。

为了解决这个问题，笔者采用基于OpenMP的细粒度并行策略。具体来说，挖掘频繁子图的过程将公共递归挖掘过程转化为BFS循环挖掘过程，从而实现单边增长粒度的并行性。此外，笔者创立了4个新缓冲区：原子区、t子区、l子区和c子区。原子区记录每个级别（从单边到多边的频繁级别）挖掘的子图集，并按照规定级别进行挖掘，其中同一级别的子图具有相同的边数。当原子区没有空间时，将进行下一级挖掘任务。t子区是单线程任务中的局部变量，记录每个子图边挖掘到的子图。l子区也是单线程任务中的局部变量，它记录从t子区并集中获得的每个线程的所有子图挖掘情况。c子区记录从每个级别挖掘得到的子图集。在单线程结束时，l子区被视作关键区域中的c子区。同时c子区和原子区将释放空间，以便进行下一级迭代挖掘。值得注意的是，在并行计算中无法确保每个线程中的所有挖掘任务同时结束，尚未完成任务的线程将继续使用原子区数据。c子区存在的意义在于不能直接将l子区应用于原子区，否则可能导致挖掘失败。新并行算法伪代码如下。

算法：新并行算法。

```
Feequent_Subgraph_Mining(GS,S)
1-6：same with old algorithml 1-6;
7：for edge e in S[l]do
8：    initialize l-edge graph g with e;
9：One_edge_growth(GS,S,g,children);
10：len←size(children);
11：while len ≠ 0 do
12：#pragma omp parallel
13：#pragma omp parallel for
14：for graph c in children
15：One_edge_growth(GS,S,c,tchildren);
16：lchildren←lchildren∪tchildren;
17:     clear(tchildren);
18：end for
19：  #pragma omp critical
20：cchildren←cchildren∪lchildren;
21：#end parallel
22：    swap(children,cchildren);
23：    clear(cchildren);
24：len←size(children);
25：end while
26-29：same with old algorithm 10-13;
30：end for
```

总体来说，此操作计算规模大，但因为并行粒度小，在线程调度中不使用大多数CPU资源，而且可以通过OpenMP的动态调度策略轻松实现良好的负载均衡。此外，递

归挖掘过程被循环挖掘过程替换，因此不需要系统协助管理堆栈，这会产生额外的加速。

3.1.2 内存管理深度优化

频繁子图挖掘的主要挑战是内存约束。为了实现内存重用和内存空间的有效利用，笔者采用“动态应用和存储指针”的策略。具体来说，程序动态地挖掘子图，并在图代码结构中存储边指针，而不是存储实际边对象，以便挖掘出的频繁子图与其原子图共享大多数边，这将显著节省内存空间。内存重用示意如**图1**所示。由**图1**可看出，只有边指针存储在图形代码中，而且每个子图只存储一个实例在存储器中，从而达到内存重用的目的。

3.2 多节点多进程并行加速

3.2.1 任务划分策略

多节点程序的最大挑战是通信开销。使用粗粒度并行策略可有效地避免大量通信开销。其过程是先通过MPI划分频繁子图，然后每个进程在其节点中保存其挖掘结果，以避免大量输出文件造成单节点内存压力。最后，将所有输出文件整合为最终挖掘结果。此外，结合单节点的多进程可为每个MPI进程创建多个线程完成并行化，以实现良好的性能。

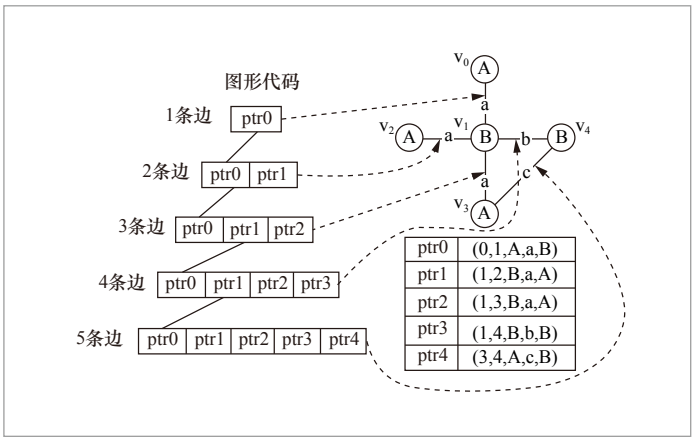

图1 内存重用示意

但是这种粗粒度策略不利于负载平衡，很容易导致数据倾斜，无法充分利用系统资源实现最佳性能。基于数据集的不同特征，笔者设计并提出了4种静态任务划分（即对等、递增、单任务和循环）策略和一种基于监管的动态任务划分策略，以尽可能地实现负载均衡。具体来说，4种静态任务划分和基于监管的动态划分如下所示。

- 对等策略（静态划分）：该策略平均分配挖掘任务，但不绝对，只需保证分配给各节点的任务相等或差值不大于1即可。然而，实验发现负载极不平衡，很容易遇到内存瓶颈。由于频繁子图按降序进行处理，大量挖掘任务集中在前端节点，同时产生大量挖掘结果，且排序越靠前的子图就越频繁，这些子图将被优先挖掘。而排在后面的子图因为瓶颈的出现，就可能不再被挖掘，而是选择一些结果并提前停止挖掘。此外，结果的规模随着顺序呈指数下降。因此，在大多数情况下，这种策略很难实现负载平衡。

- 递增策略（静态划分）：该策略给第一个节点分配一个子图，给第二个节点分配两个子图，给第三个节点分配3个子图，依此类推，最后一个节点被分配剩余的所有子图。实施此策略可以提高性能，并实现更好的负载平衡。但是，在大规模挖掘背景下，前端节点被分配太多子图会导致负载不平衡，此策略不再适用。

- 单任务策略（静态划分）：该策略给各节点分配一个子图，剩余子图分配给最后一个节点。这种策略有时可以实现很好的性能。

- 循环策略（静态划分）：该策略先按照正序给各节点分配一个子图，再按照反序

给各节点分配一个子图。按照这种方式一直循环，直到所有频繁子图都被分配为止。当数据集足够大且并行度不是特别高时，循环策略可实现更好的负载平衡。

● 监管策略（动态划分）：基于任务队列按照顺序将子图依次分配给各节点，然后将剩余子图分配给最早完成任务的节点。整个过程持续到挖掘结束。理论上，动态划分策略比静态划分策略更能实现负载平衡。为了实现这种动态策略，将process( )视为一个“监工”，监管所有任务。当节点完成任务时，它首先向process( )询问剩余子图的信息。process( )将搜索子图任务队列并返回信息。当任务队列不为空时，process( )将为此节点分配一个新的子图。否则，它将回复“1”并计数。当计数等于节点数时，process( )将结束工作。另外，当节点收到“1”时，也将结束工作。动态策略可以实现比静态策略更好的负载平衡，但由于节点和process( )需要通信，因此整体性能不一定更优。但是，在大多数情况下建议采用动态策略。5种划分策略的示例如**图2**所示。

3.2.2 删除多节点冗余结果

为了避免挖掘原子图产生多余的结果，挖掘后的频繁子图需删除。而在多节点模式中，各节点在挖掘过程中只能处理各自的频

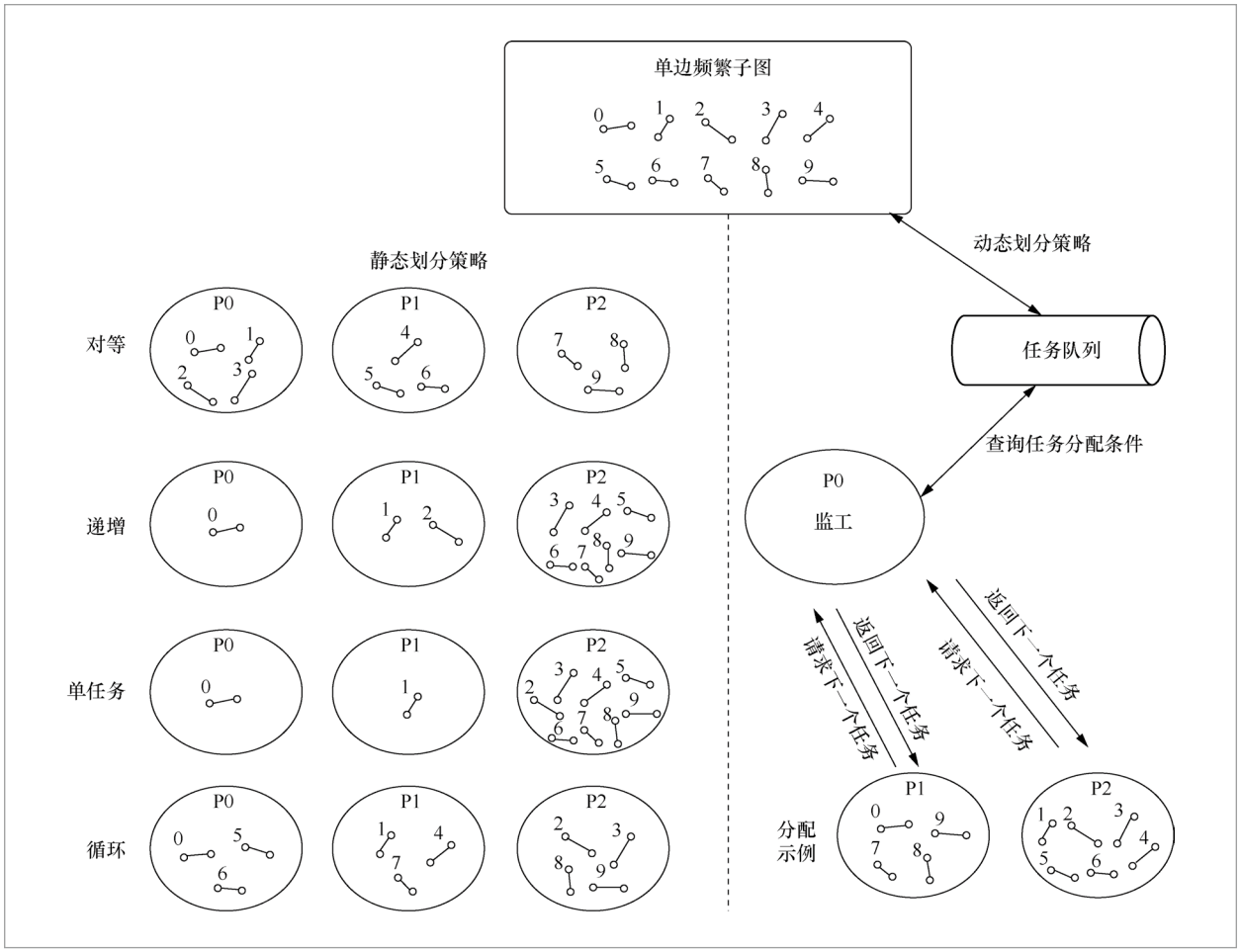

图 2　5 种划分策略示例

繁子图且不会删除其他节点中的子图，从而造成冗余结果。为了解决此问题，笔者将并行算法扩展到多节点模式，且分别处理分配给各节点的频繁子图。这个特性有可能在挖掘过程之前去除先于当前对象的多余频繁子图。

3.3 CPU–MIC协作并行优化

3.3.1 cmFSM的协同并行化

CPU-MIC协作采用中粒度的并行策略。总体来说，先在CPU和MIC之间划分频繁子图，并采用传输模式将频繁子图转移到MIC。通过承受一定的通信开销，并充分利用MIC的多核计算能力，可以实现负载平衡和预期的计算速度效果。CPU和MIC之间的协同过程如图3所示。值得注意的是，笔者没有使用粗粒度策略，因为在该策略下难以有效地实现负载平衡。另外，频繁子图不一定能合理地分配给每个进程。例如，有可能遇到一些进程只能负责一个子图。而且CPU和MIC之间存在计算能力差异，负载平衡将是一个巨大的挑战。此外，细粒度策略也不加以考虑，因为它不适配CPU和MIC之间的共享内存，且需要大规模的通信

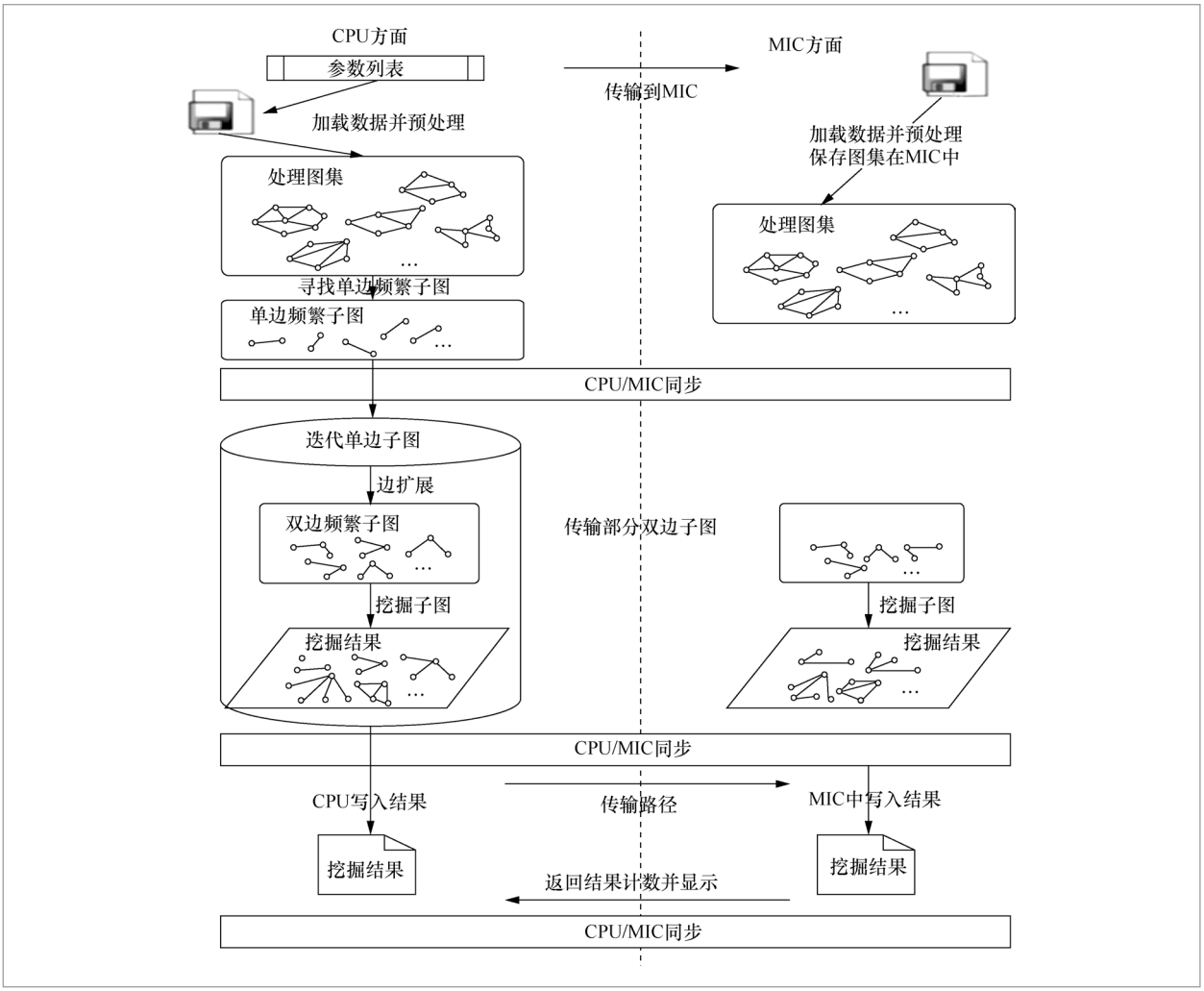

图 3 CPU–MIC 协同并行框架

开销才能传输和划分子图。因此，这种策略也不利于提高整体效率。

3.3.2 内存重用

一张MIC卡的内存大约为5 000 MB，无法与节点内存适配，而且内存分配和释放的速度比CPU慢。实验测试表明，在MIC上分配1 000 MB的内存大约需要5 s。因此，必须降低MIC上的内存分配和释放效率，并最大限度地提高内存重用。除了“动态应用和存储指针”策略外，cmFSM可通过内存重用减少MIC上的内存分配时间，具体方法是创建一个计数器JobCount记录工作号。如果count=1，则使用“alloc_if(1)free_if(0)”为列出的数组和对象分配内存；当count>1时，使用“alloc_if(0)free_if(0)”重用内存；最后使用“alloc_if(0)free_if(1)”在操作完成后释放内存。这种方式可最小化MIC上的内存分配和释放的频率。

此外，当数据集相对较大且挖掘程度足够深时，仅一个频繁子图的挖掘过程也会耗尽MIC上的内存。在这种情况下，不能先传输所有要挖掘的子图，再挖掘其结果。因此笔者不是一次上传所有的频繁子图，而是采用迭代的方法，一次只上传同一频繁子图挖掘到的部分图，以方便数据压缩节省MIC的存储空间。

3.3.3 数据传输优化

尽管MIC支持C++STL (standard template library)，但ICC (Intel C++ compiler) 不支持使用offload模式传输频繁子图，它只支持没有指针的基本数据类型和数组。因此，笔者采用“拆除和恢复”的策略传输子图。在迭代中传输的n个频繁子图原理如**图4**所示。

首先，拆除频繁子图，并集成它们的元素，以使相同结构的频繁子图存储在同一个连续缓冲区中。然后通过offload模式将这些缓冲区传输到MIC中，并在MIC上分配内存。最后通过填充这些缓冲区中的相应元素，在MIC中恢复子图。从**图4**可以发现，在传输过程中使用了7个缓冲区。因为边的代码格式是五元组 (ix、iy、x、a、y)，所以前5个缓冲区用于传输边。下标为0的元素代表第一个公共边。以下n个元素表示每个子图的第二条边。next缓冲区表示n个子图的节点数。gs缓冲区是数据集中频繁子图的编号列表。因为数字从0开始，笔者使用-1分隔这些列表。总体而言，笔者可以有效地组织和传输CPU和MIC之间的数据。

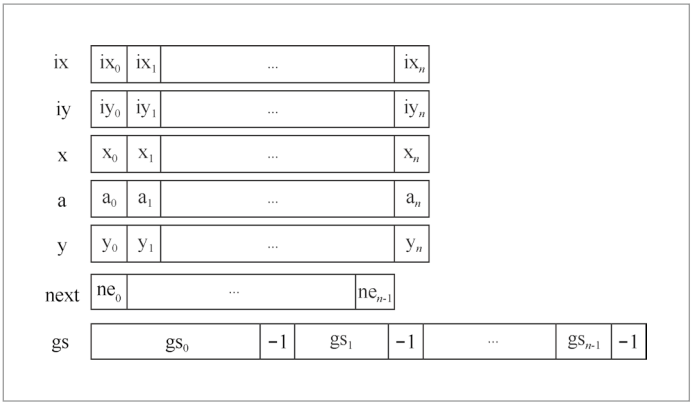

图 4 双边子图迭代传输格式

另外，一些复杂的数据结构将用于整个挖掘过程，例如处理子图集可能产生巨大的传输开销和内存分配开销。因此，复杂的数据结构需要备份，这些数据结构是可重用的，并且难以最大性能化传输。具体来说，将分析参数传递给MIC，协处理器可以根据这些参数读取数据并自行构建子图集，此操作简单，可以在MIC上快速完成。在许多情况下，挖掘结果太大，无法通过offload模式进行传输。因此返回结果计数以显示CPU的总体结果，而不是将所有挖掘结果返回给CPU。具体挖掘结果将直接写在MIC上。这样可以轻松合并这些文件数据获得整个结果。

3.3.4 CPU、MIC之间的负载平衡和数据分配

根据先前的策略，只有从相同边挖掘的频繁子图才将在迭代中被转移到MIC。如果继续挖掘这些频繁子图，挖掘结果将更贴合实际。因为CPU和MIC的计算能力在测试后依然紧密贴合，所以笔者简单地采用一种使用区间划分的静态策略，使具有稍强计算能力的主机设备首先启动，这是因为任务队列前端的频繁子图理论上仍然具有很多充满可能性的挖掘，以此实现CPU和MIC的负载平衡。

基于3个MIC的数据划分和CPU-MIC协作挖掘过程如**图5**所示。虽然每个设备的挖掘深度和计算规模都不确定，但是所有过程都将持续到频繁子图不再挖掘为止。在多节点模式中，为形成不同的任务队列，只为每个节点分配不同的频繁子图。

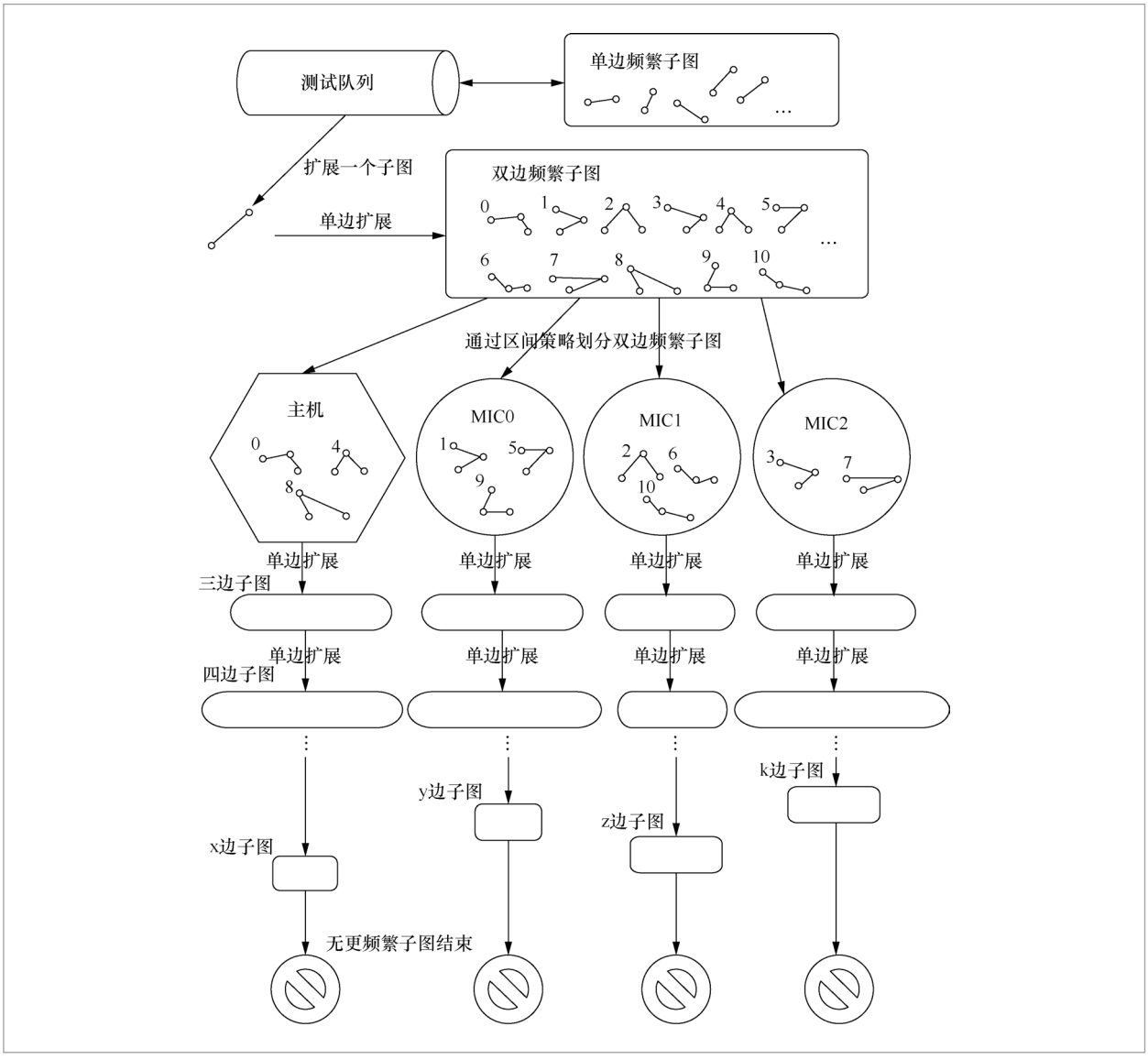

图 5 数据划分和 CPU–MIC 协作挖掘过程

4 实验结果

笔者在5个方面评估了cmFSM的性能：单节点并行化、多节点划分策略、多节点多线程加速效率、多节点协作和多节点CPU/MIC协作。

4.1 设置和数据集

cmFSM使用STL在C++中实现，并使用-O2编译标准进行编译。第一个实验是在高性能服务器上进行的，该服务器由8个18核Xeon E7-8800 v3 CPU处理器、2个57核Xeno Phi 3120协处理器和2个K40 M GPU组成，内存为2 TB，操作系统是Red Hat Enterprise Linux Server 7.1版。接下来的4个实验是在天河-2超级计算机上进行的。配置见**表1**。

笔者对合成数据集的实验进行了全面的性能研究。测试的第一个数据集是Predictive Toxicology数据集 (PTE) 。此稀疏数据集含有340种化合物，24个不同原子，66种原子类型和4种键。第二个数据集是美国国家癌症研究所 (National Cancer Institute, NCI) /美国国立卫生研究院 (National Institutes of Health, NIH) 的发展治疗计划 (developmental therapeutics program, DTP) 的艾滋病 (AIDS) 抗病毒筛选化合物数据集。此数据集含有43 405种化合物。筛选实验的结果可分为3类：CA (高活性) 、CM (中等活性) 、CI (无效) 。笔者仅在包含422个分子 (数据集DTP) 的测试中使用CA类。

合成图数据集使用类似于Inokuchi A等人[11]描述的合成数据生成器。用户可以设置参数，以确定图的数量和平均大小。设定测试的3个数据集 (S1、S2、S3) 分别包含10 000个图形、20 000个图形和100 000个图形。**表2**显示了这5个数据集更多的信息。

4.2 单节点并行化

笔者将cmFSM与各种功能相当的频繁子图挖掘工具进行比较，如FSM、FFSM、gSpan和Gaston。这里笔者使用3个数据集进行分析，结果表明cmFSM可以比拥有相对较低并行化水平中的任何其他挖掘工具呈现出更好的性能。**表3**比较了cmFSM和其他挖掘工具在PTE数据集上的结果和运行时间。

从**表3**中不难发现，cmFSM具有显著的性能优势。**表3**中的最后3列表示cmFSM分别启动2个线程、8个线程和32个线程。可以看出，即使是串行版本，cmFSM的运行时间也小于gSpan。此外，只要启动超过8个线程，cmFSM的运行时间就比其他4种工具都小。这证明即使只应用一些并行计算资源，cmFSM依然

表1 天河-2超级计算机配置

硬件参数	详情
节点数/个	16 000
计算节点	2个Xeon E5 CPU 和 3个 Xeon Phi协处理器
CPU内存	64 GB
协处理器内存	8 GB
通信系统	互联网网络
文件系统	Lustre文件系统
操作系统	Kylin 2.6.32-431.TH.x86_64

表2 数据集信息

数据集	图数量/个	边平均	边最长	点平均	点最大
PTE	340	28	214	27	214
DTP	422	42	196	40	188
S1	10 000	29	276	26	225
S2	20 000	32	214	30	197
S3	100 000	45	321	38	278

表 3 PTE 数据集结果

最小支持度	挖掘结果	运行时间/s							
		FSM	FFSM	gSpan	Gaston	cmFSM	cmFSM_2t	cmFSM_8t	cmFSM_32t
2%	136 949	312.21	78.12	101.12	36.53	97.32	68.23	21.51	6.32
4%	5 935	11.22	5.21	7.21	2.13	3.28	2.98	1.03	0.44
6%	2 326	4.12	2.01	2.31	1.01	1.25	1.01	0.42	0.21
8%	1 323	2.51	1.03	1.21	0.66	0.71	0.63	0.27	0.16
10%	844	1.69	0.75	0.83	0.38	0.49	047	0.21	0.14
20%	190	0.66	0.58	0.65	0.12	0.15	0.17	0.08	0.06
30%	68	0.31	0.29	0.33	0.06	0.08	0.09	0.07	0.06

可以表现出比其他4种挖掘工具更好的性能。此外，挖掘结果的一致性也表明笔者提出的并行优化不会影响挖掘的正确性。

图6反映了DTP数据集上各挖掘工具的运行时间比较。从**图6**中可以看出，在DTP数据集上与其他挖掘工具相比，拥有线程较少的cmFSM也可以获得更好的性能。

图7反映了cmFSM的优异并行加速效果。笔者分别设定1%、2%、3%、4%作为支持度，以形成不同的挖掘规模。可以看出，随着线程数量加倍，加速效果呈线性递增。此外，支持度越小，挖掘规模越大，cmFSM的并行效率越高。这表明cmFSM可以很好地应用于大规模挖掘过程。

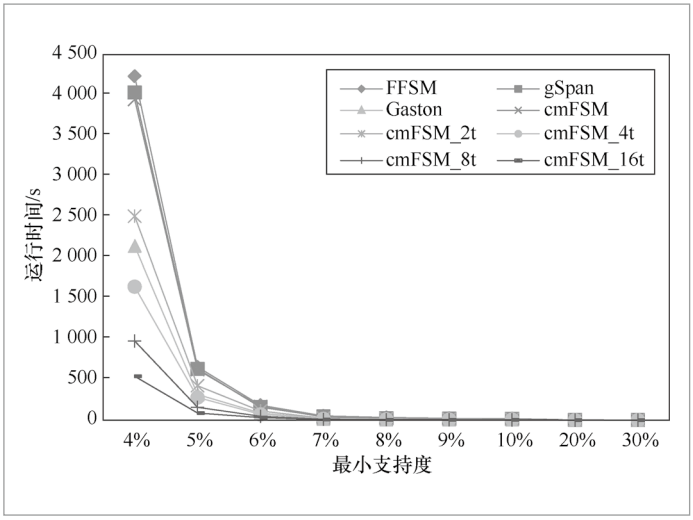

图 6 DTP 数据集上各挖掘工具的运行时间比较

4.3 多节点划分策略

为了比较不同模式下5种划分策略的优缺点，分别设置4%、2%和1%作为支持度，对DTP、PTE和S2数据集进行了实验。另外，为了消除多线程并行化的影响，只启动单进程的单个线程。在DTP数据集上的运行结果见**表4**。

从**表4**可以看出，随着进程数的增加，这5种策略的运行时间都没有显著降低。这是因为第一个频繁子图将在DTP数据集中产生80%的结果。因此，第一个频繁子图挖掘进程被视为实验瓶颈。而且对等战略是最糟糕的策略，单任务和递增策略相对更快，动态策略由于通信开销，性能略低于这两策略。然而无论挖掘深度如何，第一个频繁子图的挖掘时间始终是这种特殊数据集的瓶颈。

PTE和S2数据集上的多节点运行如**图8**和**图9**所示。可以看出，对等策略是最差的策

略，单任务和递增策略性能相当，循环策略的性能优势随着进程的增加而增加。而起初性能最好的动态策略随着进程的增加逐渐弱于循环策略，这是因为单边频繁子图的挖掘过程会有频繁的任务请求和通信开销。但是这不是一个明显的弱点，当挖掘规模足够大时，动态策略总是优于所有静态策略的。因此，笔者更推荐使用动态策略。

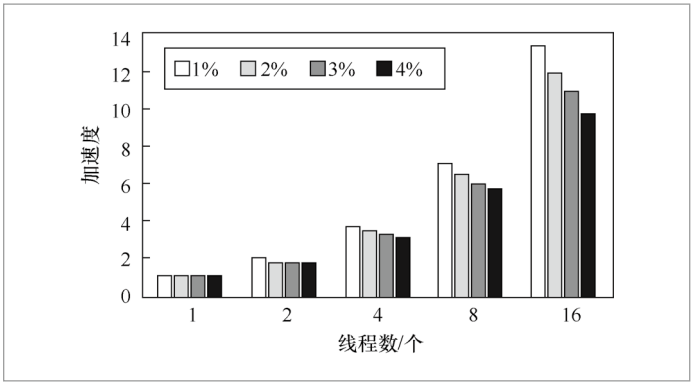

图 7 并行优化实验

4.4 多节点多线程加速效率

为了评估多节点加速效率，笔者分别对S1、S2和S3数据集设置1%、1%和2%的支持度，并进行实验。**表5**为多节点可扩展性的效果。实验中采用动态策略，这是因为动态策略可以在所有的策略中实现平均最佳性能，通过这种方式简化测试。

从**表5**可以看出，随着数据的增加，cmFSM并行效率依然很高，这意味着cmFSM可以很好地适用于大规模挖掘任务。

表 4 5 种划分策略在 DTP 数据集上的运行结果（支持度为 4%）

进程数	对等/s	单任务/s	递增/s	循环/s	动态/s
2	3 646	3 183	3 192	3 532	3 203
3	3 584	3 174	3 185	3 493	3 213
4	3 567	3 178	3 191	3 468	3 208
5	3 552	3 185	3 188	3 452	3 216

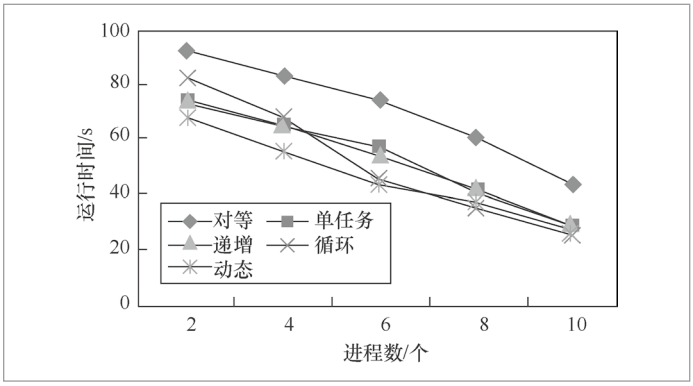

图 8 数据集 PTE 分区策略结果（支持度为 2%）

4.5 单节点CPU/MIC协作

使用分别以1%、1%和2%作为支持度的3个数据集来评估单节点CPU/MIC协作效果。结果见**表6**。由**表6**可知，通过2个CPU和3个MIC的并行加速，速度比单核运行提升明显。这充分表明了工具的运行加速效果。

图10为不同数据集上不同CPU/MIC协作模式的比较，可以发现，挖掘任务规模越大，cmFSM加速效果就越好。此外，笔者在S3数据集上获得了50倍的加速，比预期好很多。这是因为笔者采用内存重用、数据传输等进行优化。S1数据集实验很快就达到了加速瓶颈，2MIC模式和3MIC模式之间差距很小，这是由数据集本身特征引起的。在S1数据集中大量挖掘任务集中在主机和第一个MIC协处理器中。不过在大多数挖掘

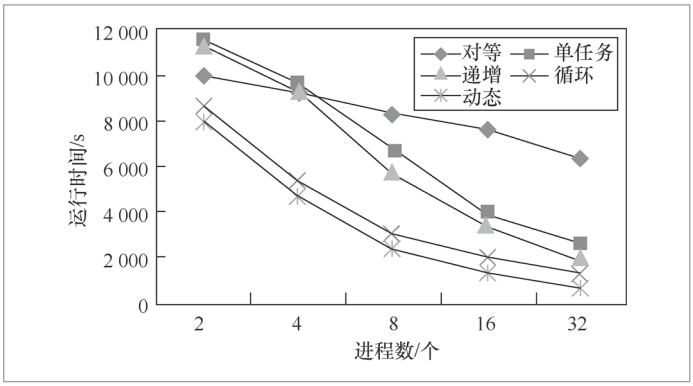

图 9 数据集 S2 分区策略结果（支持度为 1%）

表 5 多加点加速效率

数据集	运行时间/s					
	12核2进程 6线程	24核2进程 12线程	48核4进程 12线程	96核4进程 24线程	192核8进程 24线程	384核16进程 24线程
S1	742	408	223	131	83	74
S2	1 732	921	487	261	161	104
S3	22 821	11 763	6 069	3 135	1 640	871

表 6 单节点 CPU/MIC 协作运行时间

数据集	运行时间/s				
	1核	2CPU	2CPU&1MIC	2CPU&2MIC	2CPU&3MIC
S1	7 590	382	274	234	208
S2	18 944	875	607	486	410
S3	266 794	11 625	7 749	6 457	5 320

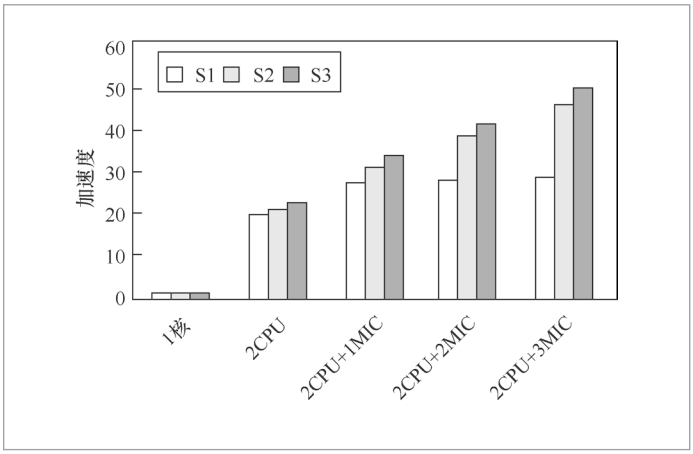

图 10 单节点上的 CPU/MIC 协作

任务中, CPU/MIC的加速效果依然显著。

4.6 多节点CPU/MIC协作

笔者使用最大数据集S3, 分别以2%、1%和0.8%作为支持度来评估多节点CPU/MIC的协作效果。**表7**表示节点充分利用2个CPU和3个MIC的结果。

图11为S3数据集上不同支持度多节点加速的比较, 从**表7**可看到具体情况, 笔者发现在大挖掘规模下可以实现更好的多节点加速效果。总体来说, 这些实验表明多节点比线性加速更弱, 这是由多节点模式通信开销、进程竞争和同步引起的。但是, 这种情况不影响cmFSM在大规模挖掘任务下的可扩展性。

5 结束语

cmFSM是一个可扩展的并行频繁子图挖掘工具, 它基于CPU/MIC协作实现多级别和多粒度的并行性。首先基于细粒度并行策略, 将常见的递归挖掘过程转换为单节点上的BFS循环挖掘过程。除了一些特殊的数据集, cmFSM呈现出接近线性加速的效果。其次, 基于粗粒度并行策略实现4种静态划分 (即对等、递增、单任务和循环) 策略和一种基于监管的动态划分策略, 以尽可能地实现负载平衡。实验表明, 特别是面对大规模挖掘任务时, 动态划分策略表现出比所有静态划分策略更好的性能。此外, 结合单节点上的多线程工作, cmFSM允许通过为每个MPI进程创建多个线程来生成二级并行化, 这显示cmFSM的可靠扩展性。第三, 基于中粒度并行策略备份数据结构, 以避免过度传输出现瓶颈。通过内存重用和充分利用MIC的多核计算能力, 可以在单个节点上为某些数据集获得超过50倍的加

速。此外，多节点CPU/MIC协作在大规模挖掘任务下提供了出色的可扩展性。

此外，几个数据集的实验评估表明，即使只使用一些并行计算资源，cmFSM明显优于现有的算法。这充分表明笔者提出的工具的有效性。但是在一些特殊的数据集中，cmFSM会显示出很大的局限性。在这种情况下，它将很快达到加速瓶颈，这需要笔者进一步改进cmFSM工具。

表 7 多节点 CPU/MIC 协作运行时间

最小支持度	运行时间/s			
	1节点	2节点	4节点	8节点
2%	5 320	2 816	1 736	993
1%	18 329	9 802	5 508	2 918
0.8%	43 252	23 026	12 847	6 295

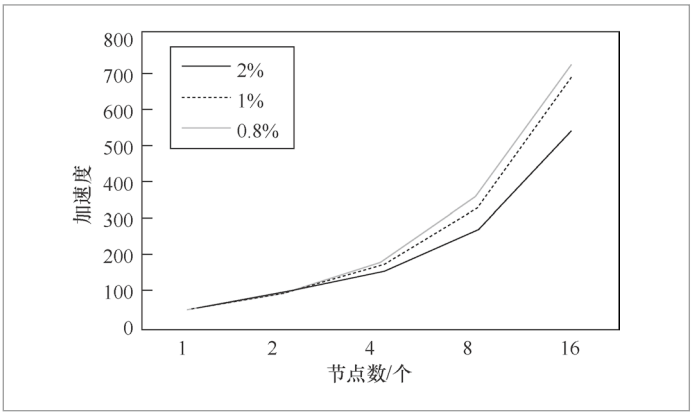

图 11 多节点上的 CPU/MIC 协作

参考文献：

[1] LIN W. Efficient techniques for subgraph mining and query processing[D]. Singapore: Nanyang Technological University, 2015.

[2] HUAN J, WANG W, PRINS J, et al. SPIN: mining maximal frequent subgraphs from graph databases[C]//The 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August 22-25, 2004, Seattle, USA. New York: ACM Press, 2004: 581-586.

[3] JIANG X, XIONG H, WANG C, et al. Mining globally distributed frequent subgraphs in a single labeled graph[J]. Data and Knowledge Engineering, 2009, 68(10): 1034－1058.

[4] KURAMOCHI M, KARYPIS G. Finding frequent patterns in a large sparse graph[J]. Data Mining & Knowledge Discovery, 2005, 11(3): 243－271.

[5] KANG U, TSOURAKAKIS CE, FALOUTSOS C. PEGASUS: mining peta-scale graphs[J]. Knowledge & Information Systems, 2011, 27(2): 303－325.

[6] REINHARDT S, KARYPIS G. A multi-level parallel implementation of a program for finding frequent patterns in a large sparse graph[C]//2007 IEEE International Parallel and Distributed Processing Symposium, March 26-30, 2007, Rome, Italy.Piscataway: IEEE Press, 2007: 1－8.

[7] WU B, BAI Y L. An efficient distributed subgraph mining algorithm in extreme large graphs[C]//The 2010 International Conference on Artificial Intelligence and Computational Intelligence,October 23－24, 2010, Sanya, China. Heidelberg: Springer, 2010: 107－115.

[8] YAN Y, DONG Y, HE X, et al. FSMBUS: a frequent subgraph mining algorithm in single large-scale graph using spark[J]. Journal of Computer Research and Development, 2015, 52(8): 1768－1783.

[9] LIN W, XIAO X, XIE X, et al. Network motif discovery: a GPU approach[C]//IEEE 31st International Conference on Data Engineering, April 13-17, 2015, Seoul,Korea. Piscataway: IEEE Press, 2015: 831－842.

[10] HILL S, SRICHANDAN B, SUNDERRAMAN R. An iterative MapReduce approach to frequent subgraph mining in biological datasets[C]//The ACM Conference on Bioinformatics, Computational Biology and Biomedicine, October 7-10, 2012, Orlando, USA. New York: ACM Press, 2012: 661－666.

[11] INOKUCHI A, WASHIO T, MOTODA H. An apriori-based algorithm for mining frequent

substructures from graph data[C]//The 4th European Conference on Principles of Data Mining and Knowledge Discovery, September 13-16, 2000, London, UK. London: Springer-Verlag, 2000: 13-23.

[12] KURAMOCHI M, KARYPIS G. Frequent subgraph discovery[C]//IEEE International Conference on Data Mining, November 29-December 2, 2001, San Jose, USA. Piscataway: IEEE Press, 2001: 313-320.

[13] MEINL T, FISCHER I, PHILIPPSEN M. A quantitative comparison of the subgraph miners mofa, gspan, FFSM, and gaston[C]//European Conference on Principles and Practice of Knowledge Discovery in Databases, October 3-7, 2005,Porto, Portugal. Heidelberg: Springer, 2005: 392-403.

[14] BORGELT C, BERTHOLD M R. Mining molecular fragments: finding relevant substructures of molecules[C]//2002 IEEE International Conference on Data Mining, December 9-12, 2002, Maebashi City, Japan. Piscataway: IEEE Press, 2002: 51-58.

[15] HUAN J, WANG W, PRINS J. Efficient mining of frequent subgraphs in the presence of isomorphism[C]//The 3rd IEEE International Conference on Data Mining, November 19-22, 2003, Melbourne, USA. Washington, DC: IEEE Computer Society, 2003: 549-552.

[16] YAN X, HAN J. gSpan: graph-based substructure pattern mining[C]//2002 IEEE International Conference on Data Mining,December 9-12, 2002, Maebashi City, Japan. Piscataway: IEEE Press, 2002: 721-724.

[17] NIJSSEN S. A quickstart in frequent structure mining can make a difference[C]//The 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August 22-25, 2004, Seattle, USA. New York: ACM Press, 2004: 647-652.

[18] BUEHRER G, PARTHASARATHY S, CHEN Y K. Adaptive parallel graph mining for CMP architectures[C]//The 6th International Conference on Data Mining, December 18-22, 2006, Hong Kong, China. Piscataway: IEEE Press, 2006: 97-106.

[19] WANG C, WANG W, PEI J, et al. Scalable mining of large disk-based graph databases[C]//The 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August 22-25, 2004, Seattle, USA. New York: ACM Press, 2004: 316-325.

[20] NGUYEN S N, ORLOWSKA M E, LI X. Graph mining based on a data partitioning approach[C]//The 19th Conference on Australasian Database,December 3-4, 2007, Gold Coast, Australia. Darlinghurst: Australian Computer Society, Inc., 2008: 31-37.

[21] DEAN J, GHEMAWAT S. MapReduce: simplified data processing on large clusters[C]//The 6th Conference on Symposium on Opearting Systems Design & Implementation, December 6-8, 2004, San Francisco, USA. Berkeley: USENIX Association, 2004: 107-113.

[22] BHUIYAN M A, AL H M. An iterative MapReduce based frequent subgraph mining algorithm[J]. IEEE Transactions on Knowledge & Data Engineering, 2013, 27(3): 608-620.

[23] LU W, CHEN G, TUNG A K H, et al. Efficiently extracting frequent subgraphs using MapReduce[C]//2013 IEEE International Conference on Big Data, October 6-9, 2013, Silicon Valley, USA. Piscataway: IEEE Press, 2013: 639-647.

[24] LIN W, XIAO X, GHINITA G. Large-scale frequent subgraph mining in MapReduce[C]//2014 IEEE 30th International Conference on Data Engineering, March 31-April l4, 2014, Chicago, USA. Piscataway: IEEE Press, 2014: 844-855.

作者简介

彭绍亮 (1979-) , 男, 博士, 湖南大学信息科学与工程学院教授, 国家超级计算长沙中心 (湖南大学) 副主任。长期从事高性能计算、大数据、生物信息、人工智能、区块链等技术研究工作, 担任国防科技大学“天河”生命科学方向负责人, 深圳华大基因研究院“特聘教授”, 湖南大学“岳麓学者”三级教授。发表学术论文上百篇。参与“天河”系列超级计算机应用软件研发工作, 参与国家“973”项目、“863”项目、军队重大型号项目等13项, 获军队科技进步奖一等奖1项, 中国计算机学会 (CCF) 科学技术奖自然科学二等奖, 2016年荣立三等功。

牛琦 (1995-) , 男, 湖南大学信息科学与工程学院硕士生, 主要研究方向为计算生物学。

李肯立 (1971-) , 男, 湖南大学信息科学与工程学院院长, 国家超级计算长沙中心主任, 教育部“长江学者”特聘教授、国家杰出青年科学基金获得者、国家“万人计划”科技创新领军人才。学术兼职有教育部“高效能计算学科创新引智基地”负责人、数据分析湖南省工程技术研究中心主任。担任国家超级计算创新联盟副理事长、新一代人工智能产业技术创新战略联盟专家委员会委员、IEEE高级会员、CCF杰出会员、CCF高性能计算专业委员会常务委员、湖南省计算机学会秘书长等。主要研究方向为并行分布式处理、超级计算与云计算、面向大数据和人工智能的高效能计算等。

邹权 (1982-) 男, 电子科技大学基础与前沿研究院教授, IEEE会员, ACM会员, CCF高级会员, 中国人工智能学会会员 (粗糙集与软计算专业委员会委员、生物信息学与人工生命专业委员会委员、机器学习专业委员会通讯委员) , 中国运筹学会会员, 中国自动化学会会员。主要研究方向为生物信息学、机器学习和并行计算, 现主要研究基于并行计算方法的下一代测序数据的蛋白质分类、基因组装配、注释和功能分析。

收稿日期: 2018-12-04

基金项目: 国家重点研发计划基金资助项目 (No.2017YFB0202602, No.2018YFC0910405, No.2017YFC1311003, No.2016YFC1302500, No.2016YFB0200400, No.2017YFB0202104) ; 国家自然科学基金资助项目 (No.61772543, No.U1435222, No.61625202, No.61272056)

Foundation Items: National Key Research and Development Program of China(No.2017YFB0202602, No.2018YFC0910405, No.2017YFC1311003, No.2016YFC1302500, No.2016YFB0200400, No.2017YFB0202104), The National Natural Science Foundation of China(No.61772543, No.U1435222, No.61625202, No.61272056)