软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(Suppl.(2)):140−149

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

使用分布式 De Bruijn 图遍历基因拼接并行构建和化简*

曾 理, 成杰峰, 孟金涛, 涂志兵, 冯圣中

(中国科学院 深圳先进技术研究院 先进计算与数字工程研究所,广东 深圳 518055)

通讯作者: 成杰峰, E-mail: jiefengcheng@gmail.com

摘 要: 目前基因拼接软件中应用最广泛的技术是基于 De Bruijn 图的基因拼接算法,需要对长达数十亿 BP 长度的基因组测序数据进行处理.针对海量的基因测序数据,快速、高效和可扩展的基因拼接算法非常重要.虽然已出现一些并行拼接算法(如 YAGA)开始研究这些问题,但是拼接过程中时间、空间消耗较大的构图和单链化简这两大步骤在海量数据的挑战下仍然是最主要的计算瓶颈.这是因为现有工作在处理这几个步骤时通常使用了并行的表排序(list ranking),而该方法需要多次对 De Bruijn 图的海量顶点信息进行分布式的排序,产生了大量的计算节点间的通信.单链化简可由 1 次 De Bruijn 图深度优先遍历完成而不再需要表排序,于是提出一种基于分布式海量图遍历方法对单链化简进行优化,极大地减少了处理器间的通信和计算节点之间的数据移动,因而取得较好的扩展性,其算法复杂度为 $O(g/p)$,通信复杂度为 $O(g)$,这里 g 为参考序列的长度,p 为处理器的核数.当对 E.coli 和 Yeast 数据集进行测试,处理器的核数从 8 个增加到 512 个时,算法可以得到 13 倍和 10 倍的加速比;当对 C.elegans 和人类 1 号染色体(chr1)数据集进行测试,处理器的核数从 32 个增加到 512 个时,算法可以得到 7 倍和 10 倍的加速比.

关键词: 并行算法;De Bruijn 图;基因拼接;图遍历;分布式处理

中文引用格式: 曾理,成杰峰,孟金涛,涂志兵,冯圣中.使用分布式 De Bruijn 图遍历基因拼接并行构建和化简.软件学报,2013,24(Suppl.(2)):140−149. http://www.jos.org.cn/1000-9825/13032.htm

英文引用格式: Zeng L, Cheng JF, Meng JT, Tu ZB, Feng SZ. Parallelized De Bruijn graph construction and simplification for genome assembly. Ruan Jian Xue Bao/Journal of Software, 2013,24(Suppl.(2)):140−149 (in Chinese). http://www.jos.org.cn/1000-9825/13032.htm

Parallelized De Bruijn Graph Construction and Simplification for Genome Assembly

ZENG Li, CHENG Jie-Feng, MENG Jin-Tao, TU Zhi-Bing, FENG Sheng-Zhong

(Institute of Advanced Computing and Digital Engineering, Shenzhen Institutes of Advanced Technology, The Chinese Academy of Sciences, Shenzhen 518055, China)

Corresponding author: CHENG Jie-Feng, E-mail: jiefengcheng@gmail.com

Abstract: De Bruijn graph is a vastly used technique for developing genome assembly software nowadays. The scale of this kind of graph can reach billions of vertices and edges, posing great challenges to the genome assembly task. It is of great importance to study scalable genome assembly algorithms in order to cope with this situation. Despite some recent works and therefore begin to address the scalability problem with parallel assembly algorithms, massive De Bruijn graph processing is still very time consuming which needs optimized operations. This paper aims to significantly improve the efficiency of massive De Bruijn graph processing. Specifically, focus is placed on the time consuming and memory intensive processing in the construction phase and the simplification phase of De Bruijn graph. The study observes observe that the existing list ranking approach repeatedly performs parallel global sorting over all De Bruijin graph vertices, which results in a huge amount of communications between computing nodes. It therefore proposes to use depth-first traversal over the underlying De Bruijn graph once to achieve the same objective as the existing list ranking approach. The new method is fast, effective and can be executed in parallel. It has a computing complexity of $O(g/p)$ and communication complexity of $O(g)$, where g is the length of genome reference, p is the number of processors, which is smaller than the existing list ranking approach, Experimental results

* 基金项目: 国家自然科学基金(61103049)
收稿时间: 2012-08-05; 定稿时间: 2013-07-22

using error-free data show that, when the number of CPUs scales from 8 to 512, our algorithm has a speedup of 13 and 10 times on processing the data sets of E.coli and Yeast respectively; and when the number of CPUs scales from 32 to 512, the algorithm has a speedup of 7 and 10 times on processing the data sets of C.elegans and chr1 respectively.

Key words: parallelized algorithm; De Brujin graph; assembler; graph traversal; distributed processing

基因组测序是现代生物学中基础性的问题之一.从 1977 年 Sanger 测序法出现到现在,人类已经获得数以万计的基因组序列.但这些只是所有物种中极小的一部分,大量的物种仍需要被测序.现代医学研究表明,几乎所有的疾病都与基因有关系.基因测序的结果有助于揭示遗传与变异的奥秘,并且被广泛应用于基因诊断、基因治疗、药物设计等领域.

基因测序主要有两个步骤:第 1 步,获取短序列 read 的序列信息.先将 DNA 分子进行扩增(扩增的倍数被称为覆盖度),然后这些 DNA 分子被随机打断成很多小片段(每个小片段被称为 read),再通过测序仪测定 read 的碱基序列(如图 1 所示).第 2 步,以 read 序列为基础,进行序列拼接,还原原始序列.序列拼接问题类似于寻找最短公共超字符串问题[1].

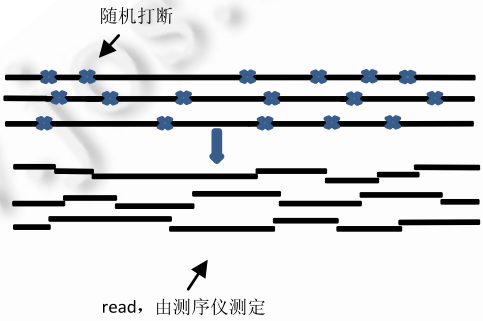

图1 DNA 与 read 的关系

序列拼接算法主要有两类.第 1 类是基于 overlap 图的算法.在 overlap 图中,每一个短序列 read 都被当作一个顶点,如果两个 read 之间存在重叠且重叠的长度超过一定阈值,那么就有一条有向边相连[2].因此序列拼接问题转换为在 overlap 图中寻找一条经过每个顶点的 Hamilton 路径,这是 NP-Hard 问题.第 2 类是基于 De Bruijn 图的算法.在 De Bruijn 图中,每一个 read 被切分成长度为 k 的小片段,称为 k-mer.每个 k-mer 为一个顶点[3].如果存在 read,使得两个 k-mers 相邻且重叠 k–1 个字符,那么它们之间存在一条有向边.这样,每个 read 被映射成图中的一条路径.这样序列组装问题变成了在 De Bruijn 图中寻找一条包含所有 read 的路径[4].测序仪在测序时会引入错误,下一代高通量测序仪错误率在 1%左右,同时原始序列中存在不同长度的重复片段,这两个问题使得序列拼接问题更加复杂.

早期基于 Sanger 测序法得到的序列片段,长度可以达到 1000bp.基于 overlap 图的拼接算法比较有效,但测序成本比较高.比如通过第一代测序技术完成的人类基因组计划,花费了 30 亿美元,耗时 3 年.当第二代测序技术(又称为下一代测序技术)如 Solexa、454、SOLID 技术出现后,基因测序才开始真正进入大规模应用.第二代测序技术有 3 个显著的特点,高通量、短序列、高覆盖[5].高通量,测序仪一次可以同时测定大量的 read 序列,极大降低了测序成本.短序列,序列长度一般在 25base~500base 之间.高覆盖,因为序列短,为了保证信息的完整性,需要极大地提高 DNA 的覆盖度(即 coverage)[6].但随着覆盖度的提高,read 数量成倍的增加,如果继续采用基于 overlap 图的算法,图的规模也会成倍的增长.如果采用基于 De Bruijn 图的算法,图的规模与 DNA 长度成线性关系.对同一基因组而言,其规模几乎不变.因此,面对第二代基因测序技术的大规模应用,De Bruijn 图的基因拼接算法有很大的优势.其相关的算法有 Euler[3],ALLPATHS[7],Velvet[8],IDBA[9],SOAPdenovo[10],分布式的 ABySS[11] 和分布式的 YAGA[12].其中 Euler,ALLPATHS,Velvet,IDBA 算法是串行算法,适用于小数据集的拼接,SOAPdenovo 是基于 SMP 大型机的多线程拼接算法,可拼接大型数据集如人类基因组,但最快拼接时间也需要 40 多个小时[10].

基于 De Bruijn 图的拼接算法大致包含以下几个步骤:(1) 构建 De Bruijn 图.(2) 错误剔除.对 De Bruijn 图中 3 种结构(Tips,Bubble,Spurious Link)进行剔除.(3) De Bruijn 图化简.将 De Bruijn 图中的单链进行合并,生成初步的 contig 信息.(4) 生成 scaffold.利用测序获得的 pair-end 信息,合并上一步得到的 contig,生成更长的 contig(即 scaffold).(5) 输出 scaffold 信息.拼接过程中,最消耗内存是第 1 步,时间开销最大是第 2 步和第 3 步.构建好的初始 De Bruijn 图极其稀疏,单链顶点的比例在 90%以上[13].

YAGA 算法[12]是一个分布式基因拼接算法,并行度和扩展性较好[12].本文借鉴了 YAGA 算法的思想,先构建一个分布式 De Bruijn 图,将 De Bruijn 图分布存储在每个处理器上,再并行遍历图中的所有单链进行化简.我们假设构建好的图经过了错误剔除,图中的顶点几乎是无错的,我们的工作主要集中在对图的并行化简进行优化.本文的主要贡献是,把图化简问题直接转换为图的遍历问题.在 YAGA 算法中,图化简问题转换成了 list ranking 问题,一共要进行 4 次排序,一次多轮递归.我们的算法直接从所有的端顶点出发访问所有的单链,而且访问每个顶点一次且最多不超过两次.算法的计算复杂度是 $O(g)$,通信复杂度是 $O(g)$.

本文第 1 节介绍 De Bruijn 图的定义及图的构建.第 2 节介绍图的并行化简.第 3 节介绍动态负载平衡.第 4 节给出实验结果并对全文进行总结.

1 De Bruijn 图的并行构建

1.1 De Bruijn图的定义

s 表示一个长度为 n 的字符串.s 的任何长度为 k 子字符串 $(s[j]s[j+1]...s[j+k-1], n-k+1 \geqslant j \geqslant 0)$称为 s 的 k-mer.s 的所有长度为 k 的子字符串集合称为 k-spectrum.对于一个 k-mer α,$\overline{\alpha}$ 表示 α 的反向互补序列 (比如 $\alpha = AAGTA$,那么 $\overline{\alpha} = TACTT$).通过把 α 的字符反向,再对每个字符求补,即可得到 $\overline{\alpha}$.字符的互补规则如: $\sum: \{A \to T, T \to A, C \to G, G \to C\}$.

一个 k-molecule 代表一对 k-mers $\{\alpha, \overline{\alpha}\}$,$\alpha$和 $\overline{\alpha}$ 反向互补.用$\geqslant$表示两个长度都为 k 的字符串的大小关系,比如 $\alpha \geqslant \overline{\alpha}$ 表明α在字母序上比 $\overline{\alpha}$ 大.我们把字符序大的一个称为 positive k-mer,表示为α^+,则另外一个称为 negative k-mer,表示为α^-.因此 $\alpha^+ \geqslant \alpha^-$.我们用$\alpha^+$作为 k-molecule $\{\alpha, \overline{\alpha}\}$的代表,即 $\alpha^+ = \{\alpha^+, \alpha^-\} = \{\alpha, \overline{\alpha}\}$.$k$-mer 和 k-molecule 的关系如图 2 所示.

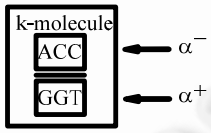

图 2 k-mer 和 k-molecule 的关系

1.2 De Bruijn图的结构

我们的 De Bruijn 图中的顶点和 YAGA 算法中 Bidrected DeBruijn 图的顶点的结构不同.顶点代表一个 k-molecule.把 k-molecule 中 α^+ 的字符序列按照规则转换成相应的无符号整型数,作为该顶点的 ID,如图3 所示.字符转换规则是{A:00, C:01 G:10 T:11}.ID 用 64 位 unsinged long long 存储,因此 k 的最大值为 31,它对应低 2k 位,高位用 0 填充.De Bruijn 图的存储以 k-molecule 为中心,边存储在顶点的 arc 域中.

DNA 由 ACGT4 种碱基构成,所以一个顶点最多与 8 条顶点相连,对应 8 条边.相连的顶点之间有 k–1 个字符重叠,仅有一个字符不同.因此只需标记不同的字符即可.我们用一个 8 位的 char 来存储可能的 8 条边的信息.如果该位为 1 则表示该边存在,如果为 0 则表示不存在.对应规则如下:前 4 位表示 postive k-mer 指向的边,0~3 位对应 ACGT,后 4 位表示 negative k-mer 指向的边,4~7 位对应 ACGT.图中阴影部分的比特位为 1 代表该边存在.因此,我们的 De Bruijn 图只用 9 个字节即存储一个顶点.

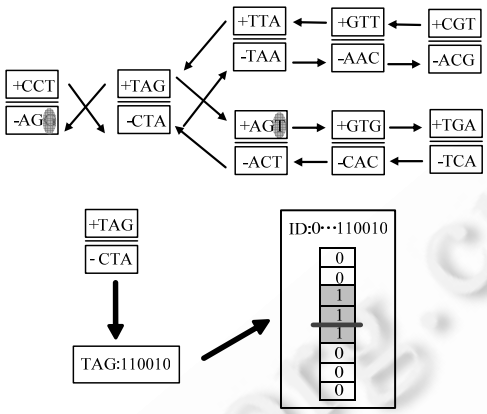

图 3 图和顶点的结构

1.3 De Bruijn图的并行构建

图的并行构建的主要思路:并行读取 read,并行切分 *k*-mer 和 *k*-mer 重分布.详细流程如下:

(1) 初始化,定位读入的文件块,并行读入 read 短序列.

根据处理器的编号,根据 read 文件的大小,各个处理器读取文件的不同块.假设文件的长度为 *L*,*n* 个处理器,那么第 *i* 个处理器读取从$((i-1)\times L)/n$ 到$(i\times L)/n$.

(2) 并行切分 *k*-mer(如图 4 所示).

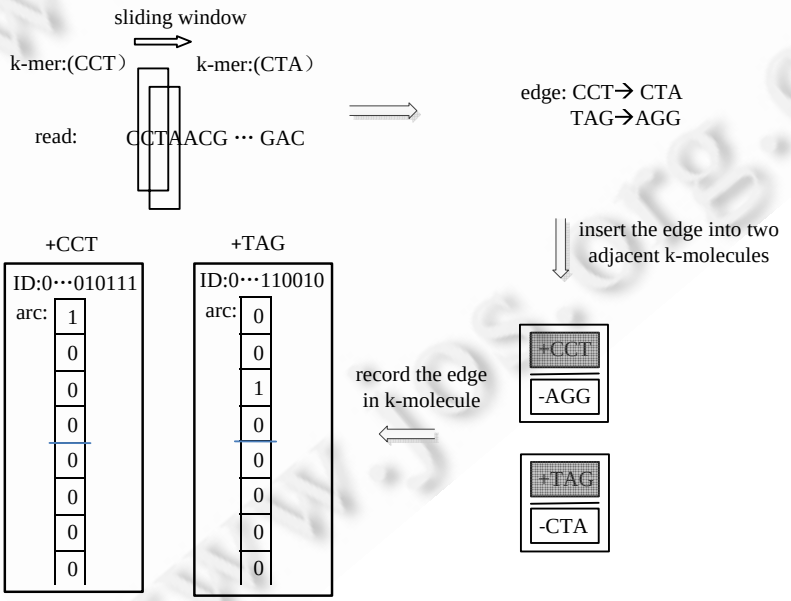

图 4 把 read 切分成 *k*-mer

每个处理器将读入的 read 切成 *k*-mer.在切分时,滑动窗口从第 1*k*-mer 滑动到最后一个 *k*-mer.算法对切分得到的 *k*-mer 作如下处理:

(a) 配对原则,推算出它的反向互补 *k*-mer,这样获得了一个 *k*-molecule $\{\alpha^+ \geqslant \alpha^-\}$,根据两个 *k*-mer 的字母序大小,将大的 *k*-mer 进行编码,作为 *k*-molecule 的 ID 值.

(b) 继续切分获得下一个相邻的 *k*-mer,做与上述(a)相同的处理.相邻 *k*-mer 存在一记录到对应的两个 *k*-molecule 中.

(3) 重分布 *k*-molecule.

按照一定的哈希规则(处理器号=hash(ID)),把 *k*-molecule 映射到不同的处理器上,这一步需要 1 次 All-to-All 的通信.在不同处理器上切分的具有相同 ID 的 *k*-molecule 被重分布到同一个处理器上,将他们的边的信息进行合并.该步完成后,任何处理器都能够根据相同规则直接定位任何一个顶点,这为后续处理中单链遍历和图的化简提供便利.

2 基于分布 De Bruijn 图的深度遍历的并行化简方法

2.1 De Bruijn图化简问题的定义

由于测序时引入了错误,De Bruijn 图构建完成后,存在大量的错误路径.错误有 3 类:Tip,Bubble 和 Spurious Link[12].拼接需要对构建好的图进行去错操作,去错后,图可看作由 error-free 数据构成的图,这时的图非常稀疏,存在大量的单链(如图 5 所示)[13].下一步进行单链化简,化简目的就是找出所有的单链.本文假设去错工作已经完成,采用 error-free 的数据,集中讨论对图的并行化简.

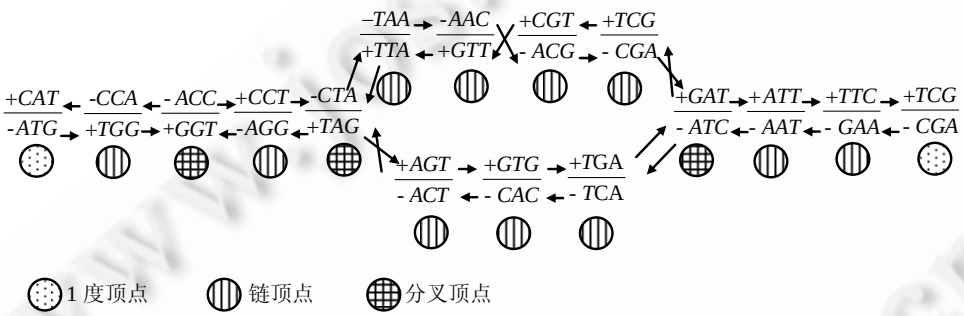

图 5 De Bruijn 图中的顶点类型

先定义几个概念,再对问题作详细描述.1 度顶点:度为 1 的顶点.链顶点:度为 2,且能通过的顶点.分叉顶点:度大于 2 的顶点或度为 2 但不能通过的顶点.单链:由链顶点构成的没有分叉的链.端顶点:单链两端的顶点,包括分叉顶点和 1 度顶点.由链顶点构成的单链是化简的对象.

2.2 图的并行化简方法

我们的算法直接从端顶点出发,遍历到链的另外一端,找到所有的链顶点,进行化简.

每个处理器上运行两个线程,A 线程和 B 线程(如图 6 所示).A 线程:负责计算任务,从本地端顶点出发进行遍历,如果链顶点不在本地,则向所在的处理器发送消息,请求该顶点.B 线程:负责接受其他处理器对本地顶点的请求,根据 endNodeID 属性,判定是否将该顶点发送给请求的处理器.

每个处理器上有两个 Map.一个是 locationMap,存储化简前的顶点.另外一个是 subGraphMap,存储图化简后的顶点.初始化时,subGraphMap 为空,在化简过程中,会不断地插入单链合并后剩下的顶点.

基于上面的结构,算法流程如下:

(1) 初始化,De Bruijn 图分布存储在每个处理器的 locationMap 中,同时新建一个 subGraphMap.

(2) 遍历单链.所有处理器同时从本地的 locationMap 中的端顶点出发遍历单链.如果不加限制,那么遍历单链时存在如下 3 种情况:同一条链,同一个处理器先后访问;同一条链,两个处理器先后访问;同一条链,两个处理器同时访问.为了减少顶点的重复访问,顶点中增加一个 endNodeID 变量,存放访问过它的端顶点的 ID 号.根据这个端顶点的 ID 号来判断当前遍历是否继续或者退出.

(a) 处理器从端顶点出发访问单链顶点,如果该顶点被第 1 次被访问,则记录下本次访问的端顶点 ID 号.如果访问了该单链的所有链顶点直到另一个端顶点都没有遇到冲突,那么该单链被分配给该处理器(如图 7 所示的 1、2).

(b) 如果在访问第 1 个链顶点时发现该链已被访问过(查看 endNodeID 值),则回退,继续访问下一条单链(如图 7 所示的 3).

(c) 如果访问的顶点(非第 1 个单链顶点)同时也被另外一个处理器访问,则比较自己端顶点 ID 号与对方端顶点 ID 号的大小,选择继续或放弃.如果当前端顶点 ID 号小,则继续访问,反之,则放弃,继续访问下一条单链(如图 7 所示的 4、5).

(3) 化简.将单链顶点全部删除,顶点信息合并到端顶点中,并将端顶点插入 subGraphMap 中.

(4) 选取下一个端顶点作相同处理,直到没有未访问的端顶点.

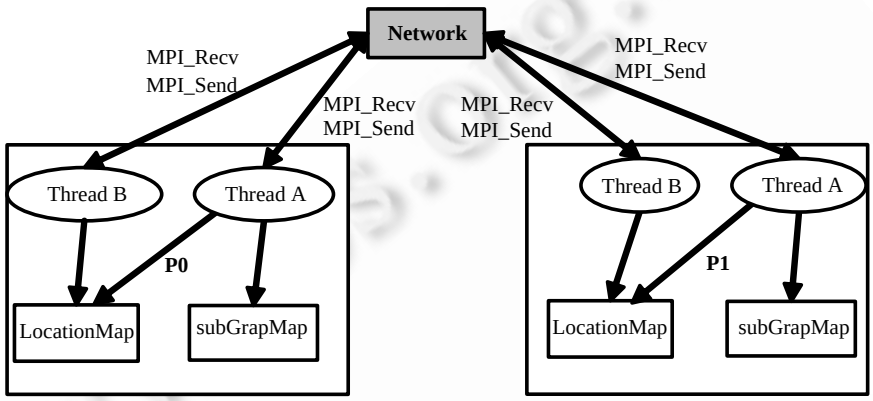

图 6 A 和 B 线程

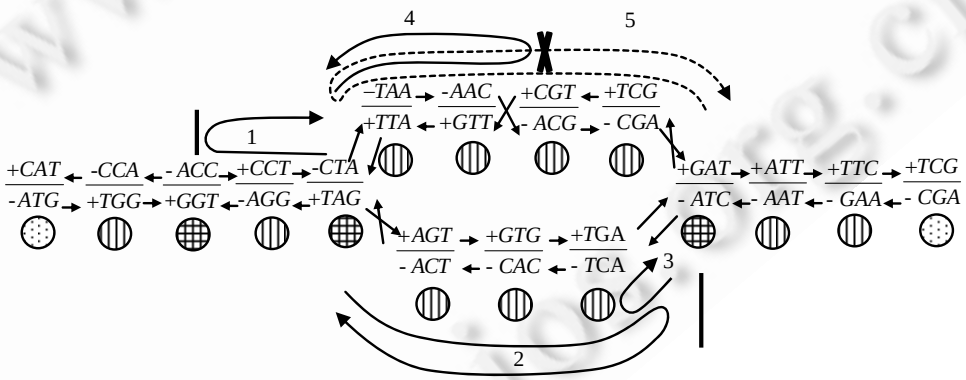

图 7 获取单链的过程

2.3 算法分析

YAGA 算法把寻找单链问题转换成 list ranking 问题,先进行 3 遍全局排序,再递归求解每个链顶点与端顶点的距离,最后进行一遍全局排序.这个过程有大量的数据移动,大部分顶点被移动了 4 次,时间和通信开销比较大.4 遍排序的计算复杂度是 $O((g/p)\ln(g/p))$,通信复杂度是 $O(g)$,还有递归过程的开销,因此计算复杂度和通信复杂度还能优化.顶点的全局移动开销很大,因此我们考虑减少顶点的移动次数.我们的算法在 De Bruijn 图构建后,每个处理器直接从本地的端顶点出发,遍历所有的单链.一条单链只有两个端顶点,所以每个链顶点最多被访问两次,同时算法尽量避免了同一条单链先后被访问两次的情况,因此只有极少数顶点移动 2 次,大部分顶点移动 1 次.计算复杂度是 $O(g/p)$,通信复杂度是 $O(g)$.

3 动态负载平衡

在最初的算法中,我们只进行了简单的负载平衡考虑,即在切分图的时候,把图中的顶点尽量均匀地哈希到每个处理器上.最初的考虑是,只要顶点的数量达到均匀,任务量就能达到平衡.

算法初步实现后,我们用 Yeast 数据集进行测试,结果显示处理器的任务量(用运行的时间来衡量)相差比较大,任务量相差最多可以达到 3 倍.算法的扩展性不好.虽然顶点数量大致相等,但是顶点的结构不同,加上切分的随机性,因此在实际运行中,有的处理器切分获得的顶点结构复杂,化简时间较长,有的处理器则化简时间很短.

我们对算法进行改进,考虑系统的动态平衡设计,采用 Master-Slave 模式.系统增加了一个 Master 节点,默认由 0 号处理器的 M 进程处理.M 进程负责维护所有顶点的位置信息,提供给所有的进程进行查询.当某个处理器的任务先完成,我们称为闲处理器,其可以向 M 进程发送信息,请求 M 进程进行任务切分,然后 M 进程根据查询的任务完成情况返回信息.具体结构如图 8 所示.

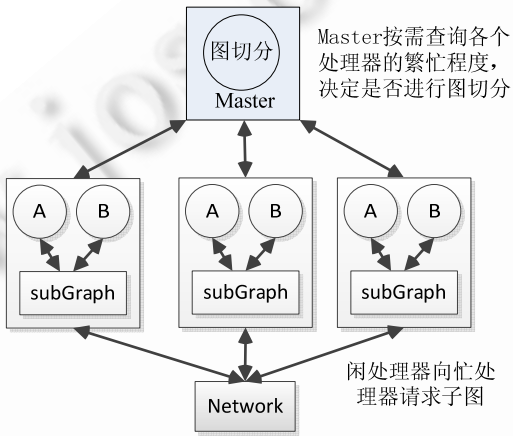

图 8 动态负载平衡模块

算法流程如下:

(1) 初始化,De Bruijn 图分布存储在每个处理器的 locationMap 中.

(2) 遍历单链.所有处理器同时从本地的 locationMap 中的端顶点出发遍历单链.如果请求的顶点不在本地,则向 Master 的 M 进程发送请求位置消息.M 进程收到请求,查询顶点的目标处理器,将目标处理器号发送给请求处理器.请求处理器拿到目标处理器号后,向目标处理器的 B 线程请求所需顶点信息.目标处理器 B 线程收到请求,将顶点信息返回,下一步处理类似第 2.2 节中的流程 2.

(3) 化简.将单链顶点全部删除,顶点信息合并到端顶点中,并将端顶点插入 subGraphMap 中.

(4) 选取下一个端顶点作相同处理,直到没有未访问的端顶点.

(5) 请求切分任务.如果某个处理器处理完毕,则向 M 进程发送请求切分任务的消息,M 进程查询每个处理器的任务完成情况,决定是否需要进行任务切分.如果存在需要切分的忙处理器,则将该处理器号发送给闲处理器.闲处理器接收到回复消息后,向目标处理器的 B 线程发送子图切分的消息.目标处理器收到消息后,对本地子图进行切分,将切分的子图顶点发送给闲处理器,同时将顶点位置信息发送给 M 进程.重复步骤(2)、(3)、(4),直到任务不能再进行切分且顶点都被化简完毕.

4 实验结果及分析

我们用 C++,MPI 实现了一个基因拼接软件.实验平台包括了 44 台曙光 5000 的高性能计算机,用 InfiniBand 网络互联.每台机器的配置为 16 核、32G 共享内存.实验选取了 E.coli,Yeast,C.elegans 和人类 1 号染色体(chr1)4

个基因组做测试.其中,E.coli 是真实数据集,参考基因序列长度为 4,639,221,包含 20,816,448 reads,read 长度 36bp.Yeast,C.elegans 和人类 1 号染色体是使用 Perl 脚本自动生成的模拟数据集,read 的长度从 36bp~50bp 不等,错误率为 0.Yeast 参考基因序列长度为 12,106,139,生成的 50 倍 Yeast 测试数据包括 17,007,362 reads;C.elegans 参考基因序列长度为 100,258,171,生成的 50 倍测试数据包括 140,396,108 reads;人类 1 号染色体(chr1)参考序列长度为 245,522,847,生成的 50 倍测试数据集包括 345,970,249 reads.实验的目标是验证 De Bruijn 图的构建和化简算法的扩展性.

运行过程被分为 3 个阶段,并行文件 I/O,构图,图的化简,对 3 个阶段分别计时.第 1 阶段,从一个分布式文件系统读入文件.第 2 阶段,构建一个 De Bruijn 图.第 3 阶段,对 De Bruijn 的进行化简.

首先测试了真实数据集 E.coli,运行时间如图 9 所示,当处理器的数目从 8 增加到 512 时,总时间从最开始的 345s 减少为 25s,平均加速比约为 13 倍.实验结果表明,算法有比较好的扩展性.但是当处理器的核数从 256 增加到 512 时,其扩展性不好.对于文件 I/O,因为实验平台由专门的存储节点供给数据,其带宽是有限的,在处理器数达到一定规模后,其文件 I/O 的效率反而会降低.同时在构图阶段会进行数据重分布,当处理器数达到一定规模,构图的通信开销也会很大,效率会降低.因此,对于文件 I/O 和构图阶段,其他数据集也存在这一现象.但是图化简扩展性比较好,尤其是大数据集,其加速效率更明显.

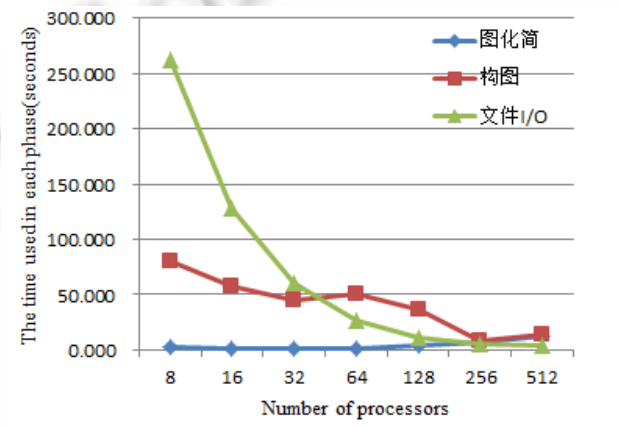

图 9 E.coli 数据集的时间开销

Yeast 数据集的运行时间如图 10 所示.当处理器的数目从 8 增加到 512 时,总时间从最开始的 308s 减少到 29s,平均加速比约为 10 倍.

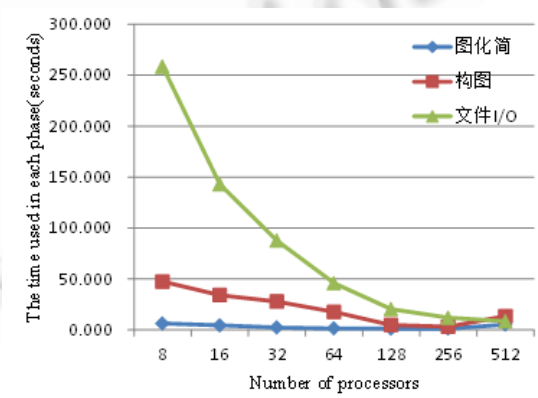

图 10 Yeast 数据集的时间开销

C.elegans 数据集的规模是 Yeast 数据集的 10 倍,运行时间如图 11 所示.当处理器的数目从 32 增加到 512

时,总时间从最开始的 701s 减少为 108s,平均加速比约为 7 倍.随着处理器的数量增加到一定规模,文件 I/O 读取效率反而会降低,但图化简的扩展性仍然很好.

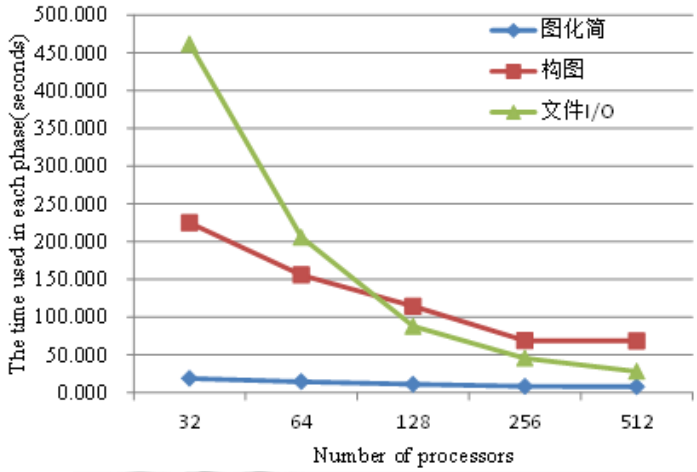

图 11 C.elegans 数据集的时间开销

最后我们测试了人类 1 号染色体(chr1),这是测试的最大数据集,其运行时间如图 12 所示.当处理器的数目从 32 增加到 512 时,总时间从最开始的 6 977s 减少到 713s,平均加速比约为 10 倍.随着处理器数量增加到一定的规模,文件 I/O 和构图的效率会降低,但图化简的扩展性仍然比较好.

5 总 结

本文针对 YAGA 算法在图化简过程中,顶点移动次数过多,造成通信开销和计算复杂度太大问题,提出了相应的改进算法.用四个数据集对算法进行测试,实验结果表明算法具有比较好的扩展性.在完成了图的化简后,图的规模急剧缩小,这说明对图的化简进行优化是很有价值的.下一步工作主要是用 pair-end 信息对图中的分支和重复进行处理.我们将在接下来的工作中解决这些问题.

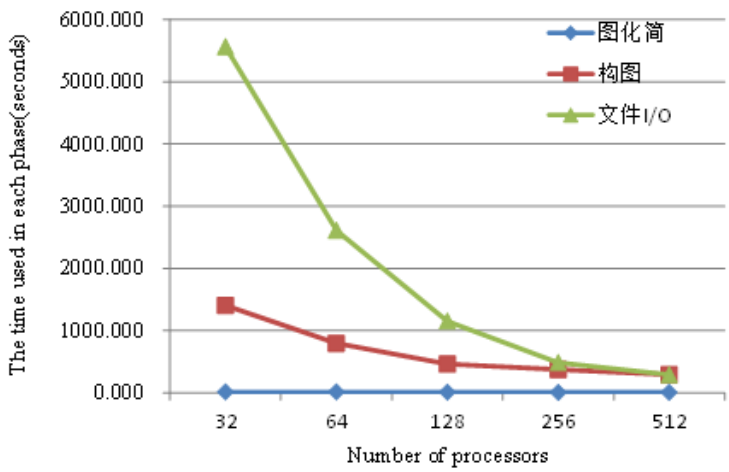

图 12 人类 1 号染色体(chr1)数据集的时间开销

References:

[1] Kundeti VK, Rajasekaran S, Dinh H, Vaughn M, Thapar V. Efficient parallel and out of core algorithms for constructing large bi-directed de Bruijn graphs. BMC Bioinformatics, 2010,11:560.

[2] Kececioglu JD, Myers EW. Combinatorial algorithms for DNA sequence assembly. Algorithmica, 1995,13(1):7−51.

[3] Pevzner PA, Tang HX, Waterman MS. An eulerian path approach to DNA fragment assembly. Proc. of the National Academy of Sciences of the United States of America, 2001,98(17):9748−9753.

[4] Medvedev P, Georgiou K, Myers G, Brudno M. Computability of models for sequence assembly. Algorithms in Bioinformatics, 2007. 289−301.

[5] Sun HX, Wang XJ. The development and future perspectives of DNA sequencing technology. Technology e-Science, 2009,2(3): 19−29 (in Chinese with English abstract).

[6] Jackson BG, Aluru S. Parallel construction of bidirected string graphs for genome assembly. In: Proc. of the 37th Int'l Conf. on Parallel Processing (ICPP 2008). IEEE, 2008. 346−353.

[7] Butler J, MacCallum I, Kleber M, Shlyakhter IA, Belmonte MK, Lander ES, Nusbaum C, Jaffe DB. Allpaths: De novo assembly of whole-genome shotgun microreads. Genome Research, 2008,18(5):810−820.

[8] Zerbino DR, Birney E. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. Genome Research, 2008,18(5): 821−829.

[9] Peng Y, Leung H, Yiu SM, Chin FYL. IDBA—A practical iterative de Bruijn graph de Novo assembler. In: Bonnie B, ed. Proc. of the 14th Annual Int'l Conf. on Research in Computational Molecular Biology (RECOMB 2010). Lisbon: Springer-Verlag, 2010. 426−440.

[10] Li RQ, Zhu HM, Ruan J, Qian WB, Fang XD, Shi ZB, Li YR, Li ST, Shan G, Kristiansen K, Li SG, Yang HM, Wang J, Wang J. De Novo assembly of human genomes with massively parallel short read sequencing. Genome Research, 2010,20(2):265−272.

[11] Simpson JT, Wong K, Jackman SD, Schein JE, Jones SJM, Birol I. ABySS: A parallel assembler for short read sequence data. Genome Research, 2009,19(6):1117−1123.

[12] Jackson BG, Regennittery M, Yangy X, Schnablez PS, Aluruy S. Parallel de Novo assembly of large genomes from high-throughput short reads. In: Parallel & Distributed Processing (IPDPS). Atlanta: IEEE, 2010. 1−10.

[13] Idury RM, Waterman MS. A new algorithm for DNA sequence assembly. Journal of Computational Biology, 1995,2(2):291−306.

附中文参考文献:

[5] 孙海汐,王秀杰.DNA 测序技术发展及其展望.e-Science 技术,2009,2(3):19−29.

曾理(1986－),男,四川宜宾人,硕士生,主要研究领域为并行计算,生物信息学.
E-mail: zengli2006a@163.com

涂志兵(1987－),男,硕士生,主要研究领域为高性能计算,分布式计算,图挖掘.
E-mail: tuzbing@gmail.com

成杰峰(1977－),男,博士,副研究员,主要研究领域为图挖掘,图数据库,海量图的可视化技术,海量图的并行算法.
E-mail: jiefengcheng@gmail.com

冯圣中(1968－),男,博士,研究员,主要研究领域为高性能计算,网格计算,生物信息学.
E-mail: sz.feng@siat.ac.cn

孟金涛(1982－),男,工程师,主要研究领域为并行计算,分布式算法设计,生物信息.
E-mail: jt.meng@siat.ac.cn