密级:（涉密论文填写密级，公开论文不填写）

中国科学院大学
University of Chinese Academy of Sciences

博士学位论文

高可扩展基因组组装应用研究

作者姓名：孟金涛

指导教师：赵晓芳　高级工程师　中国科学院计算技术研究所

冯圣中　研究员　中国科学院深圳先进技术研究院

学位类别：工学博士

学科专业：计算机体系结构

研究所：中国科学院计算技术研究所

2016 年　5 月

Highly Scalable De Novo Genome Assembly

By

Jintao Meng

A Dissertation/Thesis Submitted to

The University of Chinese Academy of Sciences

In partial fulfillment of the requirement

For the degree of

Doctor of Engineering

Institute of Computing Technology,

May, 2016

声　　明

我声明本论文是我本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，本论文中不包含其他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

作者签名：　　日期：

论文版权使用授权书

本人授权中国科学院深圳先进技术研究院可以保留并向国家有关部门或机构送交本论文的复印件和电子文档，允许本论文被查阅和借阅，可以将本论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编本论文。

（保密论文在解密后适用本授权书。）

作者签名：　　导师签名：　　日期：

摘 要

面对超大规模基因数据的组装分析，TB-PB 级基因组数据组装分析是高性能计算领域研究的重要课题之一。传统的串行基因组 De Novo 组装方法已无法处理 TB-PB 级别的基因组数据分析，而目前主流的并行基因组 De Novo 组装方法主要针对 TB 级以下的基因组数据，对 TB-PB 级基因组数据的组装分析存在内存溢出、并行效率低、扩展性差等问题，这就对基因组装算法在算法效率、内存优化策略等方面提出了更高的要求。如何结合下一代的基因测序的优势，通过并行算法设计和优化，研发针对 TB-PB 级的高效、高质量、高可扩展基因的 De Novo 组装分析算法是本文研究的重要问题。

针对大型基因组的高效可扩展组装分析方法，本文结合数学理论、计算理论、系统理论，分别对基因组高可扩展的并行组装的数学模型，高效并行的基因组装计算模型，并行基因组装的深度系统优化进行了深入研究，并作出了以下贡献：

1. **基因组组装理论模型重建：** 针对当前基于 De Bruijn 图的基因组装策略并行化难度高的特点，提出了双向多步 De Bruijn 图(MSG)，使得基因组装问题转化为一个可并行的半群计算模型。从而解决基因组组装上的数据耦合问题。本课题还将对其并行可行性进行分析与论证。基于 MSG 的并行组装策略的复杂度已经达到本领域最低，即计算复杂度和通讯复杂度上分别达到 $O(n)$ 和 $O(g(log(log(g))))$, 这里 n 是输入数据的碱基数量，g 是参考基因组的长度。
2. **高效并行的基因组装计算模型及组装软件：** 针对半群计算问题，本课题提出了 SWAP 的全局异步子集同步的并行计算模型。该模型可自动挖掘该类半群计算问题中的潜在的并行性，同时本文给出该计算模型的算法复杂度，通讯复杂度和加速比的计算公式。最后基于该计算模型本文开发了一个并行组装软件 SWAP-Assembler，该组装软件达到贡献 1 中所要求的计算和通讯复杂度。
3. **基因组装软件底层深度优化：**针对文件系统，通讯硬件设备、支撑硬件平台的多样性以及组装软件中计算、通讯、I/O 同步优化的干扰问题，本文为 SWAP-Assembler 组装软件的各步骤了参数空间，并基于该参数空间进行多层次多维度性能自动优化以挖掘底层计算平台性能极限。为深度优化组装算法的底层库，本文使用并行图数据的 streaming I/O 技术,内存数据压缩技术，消息通讯聚合压缩技术以提升现有 SWAP-Assembler 的问题规模，系统效率和扩展性。
4. **组装软件的高扩展性：** SWAP-Assembler 的扩展性测试实验结果表明 SWAP-Assembler 在组装现有最大的基因组数据-千人基因组数据（4TB）时，可以扩展到 131，072 核心，并保持 40%的系统并行效率。 除此以外，SWAP-Assembler 的在组装炎黄基因组时，运行速度是现有最快的基因组装软

件 HipMer 的 3 倍。在使用 65，536 核心时，可以在 1 分钟内完成组装，而 HipMer 最高只能扩展到 15，000 核心。

5. **组装结果的高质量高精度：**SWAP-Assembler 使用启发式 contig 扩展策略，以及变长 kmer 扩展技术来提高 contig 长度并保证 contig 的高精度。在组装规模较小的两个基因组(S.aureus 和 R.sphaeriodes)数据集时 SWAP-Assembler 组装结果的 N50 长度在参与比较的五个组装软件中居于第二，而其组装的 contig 的精度位据第一。对大规模的基因组数据（Hg14, Fish 和 Yanhuang）进行组装的结果在所有五个组装软件中，SWAP-Assembler 的 N50 长度最长，组装精度依然最高。

基于以上研究，本文开发了高可扩展的基因组序列组装软件 SWAP-Assembler，其开源地址为 https://sourceforge.net/projects/swapassembler。实验结果表明，SWAP-Assembler 组装软件对规模较小的两个基因组(S.aureus 和 R.sphaeriodes)的组装结果的 N50 长度在五个基因组组装软件中位于前列，组装精度最高，而对大规模的基因组数据（Hg14, Fish 和 Yanhuang）进行组装的结果在所有五个组装软件中，其 N50 长度最长， 组装结果最好。

关键词：基因组装,半群系统,并行计算,可扩展性,并行效率

Abstract

Genome assembly on huge gene dataset is essential on biology data analysis. However previous De Novo genome assembly strategy cannot resolve the data analysis requirement on Tara or even Peta bytes sequencing data, and nowadays the main stream parallelized genome assembly method can only fix several Giga bytes data. Because of the huge dataset of Tara bytes, one can see that the huge memory usage, low parallel efficiency, and poor scalability is the main challenge for the problem of genome assembly on huge sequencing data, these challenges posed new requirement on better solutions for genome assembly strategies. Therefore how to take advantage of the next generation sequencing technology with new parallelized assembly strategies to develop a highly scalable, highly efficient de novo genome assembly with high quality results is a challenge research area in both biology and computer sciences.

In this paper, we dig into the mathematical model, parallel computing model, and system optimization of highly scalable genome assembly for huge sequening data, and this work contributes the following five points:

1. **Scalable mathematical model for genome assembly**: regarding to the previous De Bruijn graph based genome assembly strategy, we proposed a multi-step bi-directed De Bruijn graph (MSG) to abstract the genome assembly problem. MSG has transformed the tranditional genome assembly problem to an edge-merging operation on a given semi-group, and this edge-merging operation can be processed out of order. Finally after merging all the semi-extended multi-step bi-directed edges in MSG into full-extended multi-step bi-directed edges E_S^*, all the labels attached on these edges are contigs. Two properties of MSG confirms that the mathematical model of MSG is equal to the problem of genome assembly, and data dependency of computation in edge merging operation has also being resolved by our mathematical model of MSG, which means that this model can be easily parallelized. With MSG genome assembly strategy, the complexity of our method is the lowest compared with other strategies, the complexity analysis results shows that its computational complexity is $O(g/p)$, commummication complexity is $O(g\log(\log(g))/p)$, and the communication round is $O(\log(\log(g))$。

2. **Highly efficient computing model for genome assembly**: in order to process the edge merging operations in MSG more efficient, we have proposed a small world asynchronous computing model (SWAP) for the computation of MSG. This computating model can maximize the parallism of the edge merging operations in the

semi-group. Then we have implemented SWAP computing model and also analysised its complexity in computing, communication, and scalablity. SWAP-Assembler is a genome assembler developed with MSG mathematical model and SWAP computing model. The complexity of this software meets the requirement of our contribution 1.

3. **System optimization of parallelized genome assembler**: In our optimization strategy, we prefer to use these non-archtechture related parameters automatical optimization solutions to maximize the generality of our strategies on various computing platforms. An auto-adjust strategy is used to approaching the computing hardware, communication hardware and distributed file system's physical performance limitation. In order to avoid the interference of computation, communication and I/O in SWAP-Assembler, data buffer is used to isolate these operations and also provide a well definied parameter space for performance auto-tuning, the overlap technology is also used here to boost the system performance. What's more, the streaming I/O technology for graph data, memory compression, and message aggregation and compression technology is also helpful in improving the problem size, efficiency, and scalability of SWAP-Assembler.
4. **Scalablity of SWAP-Assembler**. Two experiements have been used to test the scalability of SWAP-Assembler. Test results show that SWAP-Assembler can be used to assemble 1000 human genome dataset (4T bytes), and scale to 131, 072 cores (this is the highest records in parallel genome assemblers) with an efficiency of 40%. On assembling the Yanhuang dataset (300G bytes), with 16,384 cores the runtime of SWAP-Assembler is 3 times faster than the fastest genome assembler HipMer, and with 65,536 cores, SWAP-Assembler can assemble the Yanhuang dataset in 1 minitues.
5. **Accuracy and Quality of SWAP-Assembler's results**. SWAP-Assembler uses a Heuristic strategy on contig extension. This strategy includes tip removal, bubble merging, multiple edges removal, and cross node resolve and vitural cross resolve. The quality test results shows that, with two small genome datasets (S.aureus and R.sphaeriodes) SWAP-Assembler havs a rank 2 in all five genome assemblers, but its accuracy gets the first positon. With three large datasets (Hg14, Fish and Yanhuang) SWAP-Assembler's N50 size is longest and the accuracy is also highest in all the five assembly tools.

With the above research results, We development a highly scalable de novo genome assembler, which can be freely downloaded in https://sourceforge.net/projects/swapassembler。The experiemnts results shows that, compared with SOAPdenovo's aggressive strategy on

contig extension, SWAP-Assembler’s strategy is more conservative. SWAP-Assembler firstly try to keep the accuracy of thse contigs, and then try to extend the length of contigs to pursue better quality. SWAP-Assembler is a better choice for those users who are more care about the accuracy than the length of contigs (or N50 size).

Keywords: genome assembly, semi-group, parallel computing, scalability, system efficiency

目 录

图目录

表目录

第一章 引言

近年来, 在美英等发达国家基因检测已经比较普及。美国去年有 500 多万人接受基因检测，达 30 亿美元的市场规模。在英国，基因检测已经在健康超市里出现。在国内，华大基因推出了 Genebook 应用，在提供健康穿戴设备监控人体健康的同时，也开始提供向高级用户提供基因检测，基因组装等基因分析服务。随着我国科技与经济的快速发展，人们生活水平不断提高，消费能力和健康保健意识愈加强烈，对基于基因检测的服务需求将会逐年增加。按照我国基因检测的人群在 5% 左右估计，每年基因检测量至少在 300 万人次以上，并且逐年递增，市场空间巨大[1]。

基因技术使人类生活产生巨大变化。通过基因检测与辩证用药，可以给病人找到最合适的治疗方法，不仅降低了不必要的医疗支出，更避免对人体造成不必要的伤害。例如华大基因目前已展开多项基因技术服务，其中包括乳头瘤病毒 (HPV) 基因分型检测，宫颈癌的筛查，无创产检等。相比国外这些基因检测费用在中国非常低廉，其中无创产检低至 1105 元[2]。基于最近在肿瘤研究的突破，其他一些基因技术公司联合医院合作开展全基因组低覆盖度测序，以此深入研究癌症发生、发展机制，从而为诊断、治疗提供了新的思路并辅助开展临床应用检测。美国 23andMe 公司试图将基因和个人健康联系起来，并推出了 99 美元的基因诊断服务，该服务读取人的特定遗传信息并预测一些常见疾病的风险。这些预测性的结果仅具有指导性，而最终的诊断结果则需要听取医生或者专业健康咨询机构意见。实际上并发原因是多方面的，环境因素或生活方式都影响了疾病发生。基于基因技术的疾病诊断在研究和技术上还未达到临床标准，因此美国食品药品监督管理局 FDA 最近对该项服务提出了警告，并终止了该项服务[3]。各国医疗政策的制定一般比较保守，新诊断、新疗法需要较长时间才能被医疗监管机构所接受。然而华大基因已经申请到中国药监会发放的执照，可以探索性的和医院开始进行相关的基因分析相关服务，现在已经推出的部分测试型产品，例如酒精耐性检测。

除了基因检测和分析技术外，基因编辑技术也开始出现。其中主要的基因编辑技术是 CRIPSPR/Cas, ZEN, TALEN。 其中 CRIPSPR/Cas 技术主要原理是根据细胞本身用以保护自身对抗病毒的一个系统，也是一种对付攻击者的基因武器。它可以添加，删除，激活或抑制其他生物体的目标基因。美国坦普尔大学研究员发现他们使用该技术可以将艾滋病病毒从人类细胞中删除。该技术已经孵化出了创业公司，并受到谷歌公司风投以推进基因领域的抗癌技术研发。尽管高科技生物技术公司正在努力提供更多的基因相关技术和服务，而各国政府在相关政策制定上滞后的主要原因还是担心相关服务的不成熟。

然而，如何对超大规模基因数据进行组装分析是基因检测技术发展过程中必须解决的关键问题。传统的串行基因组 De Novo 组装方法已经不能满足 TB 级别的基因组数据分

析需求[3]。而目前主流的并行基因组 De Novo 组装方法主要针对 TB 级以下的基因组数据，对 TB 级基因组数据的组装分析存在内存溢出、并行效率低、扩展性差等问题[4,5]，这就对基因组分析算法在算法效率、内存优化策略等方面提出了更高的要求。结合下一代测序技术的优势，通过并行算法优化和设计，研发和设计针对 TB-PB 级的高效、高质量、高可扩展基因（包括宏基因组）De Novo 组装分析方法是当前基因组研究亟待解决的问题。

因此，针对当前 TB-PB 级基因组数据的组装算法存在的问题，结合市场需求，（采用什么研究方法，什么技术，什么实验平台）重点研究海量基因数据的快速、准确的信息挖掘方法。这也是提高基因检测技术服务的效率，促进人类疾病与健康产业的发展的关键技术。

1.1 研究背景和意义

1.1.1 研究背景

生物信息学研究的宗旨是对海量生物信息数据进行分析，从而提取和挖据新的生物信息知识。其中，涉及到计算机技术中的机器学习、模式识别、书籍分析与挖掘、组合数学、随机模型、字符串、图算法、分布式计算、高性能计算、并行计算等知识。其中，基因组组装分析的研究是当前生物信息学研究的核心之一。目前基因大数据的组装分析处理成为当前生命科学产业化面临的巨大挑战之一。

DNA 测序(DNA sequencing)作为一种重要的实验技术，是基因组分析（包括基因组组装）的重要数据来源。早在 DNA 双螺旋结构(Watson and Crick,1953)被发现后不久就有人报道过DNA 测序技术，但是当时的操作流程复杂，没能形成规模。随后在 1977 年 Sanger 发明了具有里程碑意义的末端终止测序法，同年 A.M.Maxam 和 W.Gilbert 发明了化学降解法。Sanger 法，即第一代基因测序技术，因为既简便又快速，并经过后续的不断改良，成为了迄今为止 DNA 测序的主流。然而随着科学的发展，传统的 Sanger 测序已经不能完全满足研究的需要，对模式生物进行基因组重测序以及对一些非模式生物的基因组测序，都需要费用更低、通量更高、速度更快的测序技术，第二代测序技术(Next-generation sequencing)应运而生。第二代测序主要有三个技术平台，这三个技术平台各有优点。454 FLX 的测序片段比较长，高质量的读长(read)能达到 400bp；Solexa 测序性价比最高，不仅机器的售价比其他两种低，而且运行成本也低，在数据量相同的情况下，成本只有 454 测序的 1/10；SOLID 测序的准确度高，原始碱基数据的准确度大于 99.94%，而在 15X 覆盖率时的准确度可以达到 99.999%，是目前第二代测序技术中准确度最高的。综合来讲，第二代测序技术的主要数据特点主要有四点，测序通量大，测序序列长度短，测序精度高，测序数据有一定偏倚（即测序序列的两端误差高，中间误差

低）。

基因组测序分析研究首先要获得物种的基因组，因此基因组组装分析是基因组研究必不可少的一步，尤其是基因组 De Novo 组装，对于基因组研究意义重大。第二代基因测序技术的快速发展在全球范围内产生了海量的基因数据。目前，基因组研究产生的数据正以每 12~18 个月 10 倍以上的速度增长，已远远超过著名的摩尔定律。以深圳华大基因为例，目前的基因库数量级达到数十 PB。单个人类基因组的测序原始基因数据（reads）可达约 500GB，对于该基因组的建图组装分析需要 TB 级的内存[3]。单次基因组测序的原始数据量可以达到 TB-PB 级，例如，仅仅 1 克土壤样品的基因组测序便可产生 50TB 的原始基因数据；尤其是当基因组样品中某些物种的丰度极低时，需要超高深度测序[4,5]。这就对基因组组装分析形成了巨大挑战。

其次，由于测序实验仪器误差等原因，原始的基因组测序数据有很多测序错误，这也使得基因组的 De Novo 组装变得更加复杂。还有宏基因组样品中可能包括众多的未知物种，且其丰度分布严重不均。由于很难将原始测序基因数据准确高效划分为各物种的基因数据，这就对宏基因组的 De Novo 组装造成了更大的挑战。比如，地球微生物组计划（Earth Microbiome Project）产生了 PB 级别的微生物群落基因组数据，其中一些宏基因组中含有众多未知物种，最多的宏基因组包括 18000 个物种[6]。以上这些复杂因素都对超大规模的基因组组装分析造成了困难。

第三代测序技术相对于二代测序技术通量稍低，可以得到更长的基因输出片段，这为获得更长而完整的基因组参考序列带来了很大希望。第三代测序技术原理是对单个模板分子的双链合成反应荧光或电流信号形成单碱基读出，因而对应质量值较低。三代测序技术也需要将细胞 DNA 打断，但片段可以长达 50k 以上，从而文库制备的序列偏倚大大降低。但其相对高成本的测序费用仍是当前应用的巨大障碍，同时因第三代测序技术错误率要远远高于第二代测序，这就使得基于第三代测序技术的基因组装分析变得复杂。

再者如何面对超大规模基因数据（TB-PB）组装分析已经是必须要面对的课题。传统的串行基因组 De Novo 组装方法已经不能满足 TB-PB 级别的基因组数据分析需求[3,7]。而目前主流的并行基因组 De Novo 组装方法主要针对 TB 级以下的基因组数据，对 TB-PB 级基因组数据的组装分析存在内存溢出、并行效率低、扩展性差等问题[4,5]，这就对基因组分析算法在算法效率、内存优化策略等方面提出了更高的要求。如何结合二三代混合测序的优势，通过并行算法优化和设计，研发和设计针对 TB-PB 级的高效、高质量、高可扩展基因（包括宏基因组）De Novo 组装分析方法是当前基因组研究亟待解决的问题。该项目的实施将帮助从海量的基因数据中快速及时地挖掘出重要信息，间接提高基因检测服务的效率，促进产业发展。

1.1.2 研究意义

测序技术是研究基因序列的第一步。一台 DNA 测序仪（例如 Illumina HiSeq 2000）产生长度为 50bp 到 100bp 的样本 DNA 序列片段（read）[8]，然后，这些数据需要组装技术的处理，将其重新连接在一起构成更长的序列段(contig)来重构 DNA 源序列[9-11]。另外后续的数据处理方式，还包括将组装结果与现有序列数据库中的已知参考序列进行比对，最后找出待测序列样本中含有哪些基因，这些重构的 DNA 源序列或比对的结果可以被进一步的评估和分析，可以成为解决生物问题的线索，如寻找致病病毒、进行药物设计、研究如何将纤维物质转化为生物燃料等等。其中，整个基因测序流程如图 1.1 所示。

基因测序后的分析结果有助于揭示遗传与变异奥秘，推动基因诊断、基因治疗、药物设计。由于基因数据量巨大，一台测序仪（例如 Illumina HiSeq 2000）能够在几天里产生两亿条序列片段，通常比较大的基因研究中心拥有上百台这样的测序仪[12]。测序算法以及序列拼接技术、序列比对技术是整个测序流程中的关键技术。

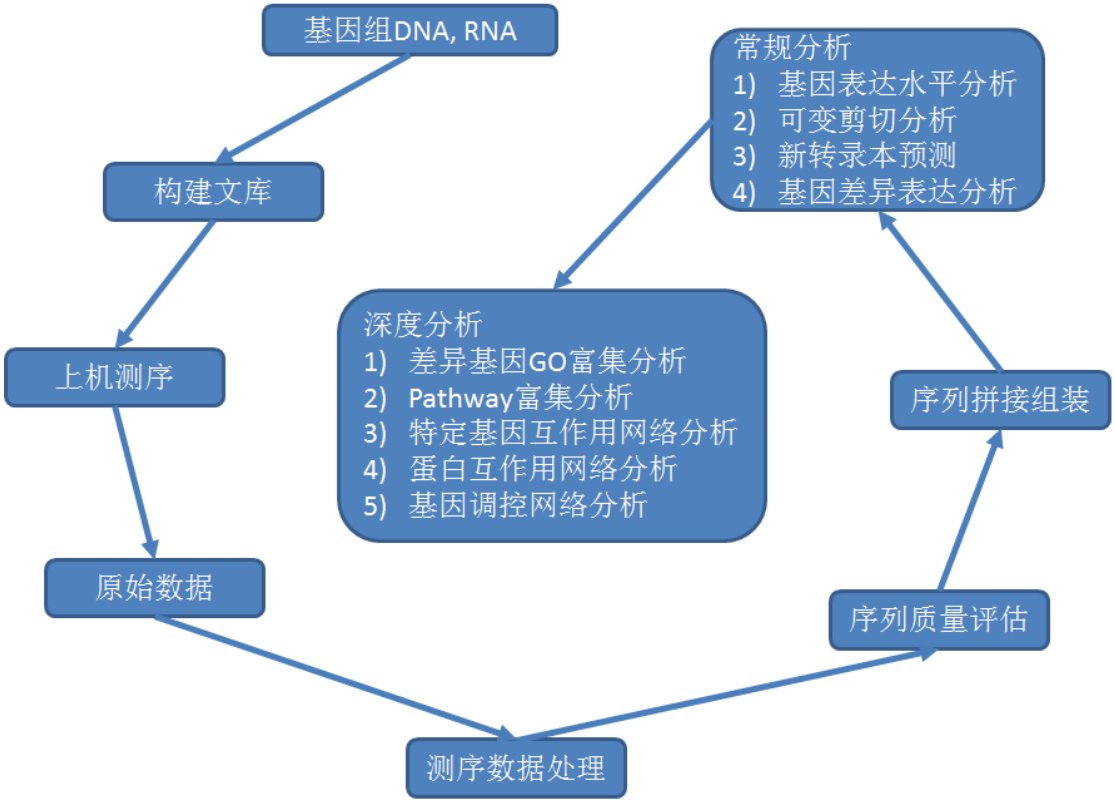

图 1.1生物测序分析流程

然而，随着 DNA 自动测序技术的快速发展，DNA 数据库中的核酸序列公共数据量正以超过摩尔定律的速度告诉增长，海量生物信息数据的处理对生命科学研究提出了严峻的挑战。在第二代高通量测序仪器广泛使用以来，一台 DNA 测序仪（例如 Illumina HiSeq 2000）产生长度为 50bp 到 100bp 的样本 DNA 序列片段（read）每天可以达到 TB 级别，然后，这些 TB 级序列片段需要基因组装技术将其重新连接在一起构成更长的序

列片段（contig）来重构 DNA 源序列，才能实现基因检测。当前最快的单机基因组组装软件（例如华大基因的 SOAPdenovo[7]）需要 2 天时间才能完成人类基因组的基因组装，实时分析效果并不理想，**大部分数据也无法得到及时分析和处理的**。

近年来，生物信息产业作为下一代新兴绿色产业并受到各国政府资助，在新一轮经济变革中处于决定性地位。随着基因组测序计划的展开、分子结构测定技术的突破和生物信息处理算法的发展，国家级的大型生物基因数据库如雨后春笋般迅速出现和成长，例如，知名的三大基因数据库有 EBI[13], NCBI[14,15], JGI[16]。

除了政府机构和科研机构外，大型跨国公司对于基因产业也开始逐步重视并开始布局。例如，2014 年初，谷歌公司收购了 NCBI 数据库， 美国国家能源局 DOE 整合了 JGI 基因数据库，中国的华大基因在发改委的资助下于深圳大鹏建立的国内最大的基因组数据库。

除此以外，基因产业也逐渐开始带来新一轮的产业革命。例如，2015 年，基因编辑技术公司 crispr，获得美国国家癌症中心 580 万美元的癌症基因组计算分析项目的 Seven Bridges Genomics 公司。 除此以外，受到谷歌风投资助的面向基因疾病关联分析的健康预测公司 23andMe。

海量生物信息的处理对生命科学研究提出了严峻的挑战。传统生物学解决问题的方式是实验，而如今，新的生命科学的研究方式主要是依靠理论分析与大规模的计算。其中，挑战最大的问题就是基因组 De Novo 组装问题。由于 DNA 自动测序技术的快速发展，DNA 数据库中的核酸序列公共数据量正以超过摩尔定律的速度指数增长，海量生物信息数据的处理对生命科学研究提出了严峻的挑战。因此，本文的研究工作对于现今生命科学的海量数据处理技术至关重要。

1.2 研究现状与存在的问题

新一代基因测序技术所产生的序列片段具有序列短、高覆盖率、额外的双端信息等特点，这使得原有的一些传统的拼接技术无法使用，加大了拼接问题的复杂度。面对如此海量的数据，高效的基因组装技术成为处理测序数据的关键。

基因组装问题主要转化为图论的知识进行求解。当前，基因组装问题的解决途径主要有两条。一类是传统的先重叠后扩展方法，即 OLC 方法[17]。虽然基于 OLC 的基因组装在技术上容易实现，但是该方法仅局限于比对大于某个阈值的read之间的信息，忽略了多个read之间的相互信息，从而使得该方法在处理重复区域问题上受到了很大的限制；另外，该方法进行序列比对时，无论使用贪心算法或者 BWA 算法，都需要耗费大量的内存，同时在任意两个 read 序列之间比对，使得该问题的算法复杂度为$O(n^2)$，其中 n 为序列片段的个数。因此，无论从时间和空间上考虑，该算法均难以用于拼接长达百万碱基的基因组样本或者由百万条以上的 DNA 序列片段组成的测序数据。基于 OLC

的拼接主要适用于基于 Sanger 测序原理的拼接[18]。测序样本也主要是基因组较小的生物,例如病毒，真菌[19]。目前，PHRAP[20]，TIGR[21]，CAP3[22]，CELERA[23]，ARACHNE[24,25]，PHUSION[26]，SSAKE[27]，VCAKE[28]，SHARCGS[29]等基因组装软件均是在此基础上开发出来的。

基因组装的另外一种解决途径，是基于 De Bruijn 图的基因组装方法。该方法主要是将 DNA 基因组装问题转化为欧拉路径进行求解[18]。基于 De Bruijn 图的基因组装技术的实现主要是通过构造并简化 De Bruijn 图来实现整个拼接过程。基于 De Bruijn 图的拼接方法理论上需要找到一条欧拉路径来重构源基因组参考序列，但实际上只能通过将构造好的 De Bruijn 图中的无分支路径进行收缩最终合并为完整的 contig 进行输出。目前，Velvet[29]，SOAPdenovo[30]，IDBA[31]，ABySS[32], PASHA[33]等基因组装软件均是在此基础之上开发出来的。

与 OLC 相比，基于 De Bruijn 图的基因组装方法有很多的优点。例如在过滤错误信息、重复区域发现、解耦以及利用双端信息对 contig 进行扩展等方面。然而，由于大基因组构造的 De Bruijn 图异常庞大以至于单个计算机的内存无法存放。例如，对人类基因组的数据进行拼接[34]，构建的 De Bruijn 图中有 30G 的顶点，而存储这样的图通常需要消耗大约 500G 到 1T 的内存。同时，由于在图的数据结构中顶点关联的随机性，使得数据预取技术失效，导致最终对图的简化也将耗费几周的时间。在实际应用中，基于 De Bruijn 图实现的基因组装技术已经应用于新一代的测序数据，然而，在拼接大型基因组和宏基因组测序数据需要高效并行的基因组组装策略，但是拼接策略并行化还面临（1）基因组高可扩展的并行组装的数学模型；（2）高效并行的基因组装计算模型；(3)并行基因组装的深度优化以及多种数据源的混合组装问题。

（1）基因组高可扩展的并行组装的数学模型问题

为处理新一代测序技术所产生的大规模的基因数据，本文需要一种高效可扩展组装策略来应对生物信息学领域的信息爆炸。虽然高性能计算、云计算以及众核技术在序列对比，SNP 查找，表达分析等应用上展现了强大的性能提升。但在紧耦合的图算法应用特别是基因组组装，高可扩展的解决方案一直是近十年来的难题。现有的 De Novo 组装策略是 M.S Waterman 等人于 2001 年提出的 De Bruijn 图策略[17]，该策略接近线性的算法复杂度使其广泛用于下一代高通量基因序列的组装。同时一些并行基因组装软件，例如 ABySS[36], Ray[37], PASHA[38], YAGA[39], HipMer[40]等，也对该策略进行了并行实现。然而 M.S Waterman 等人于 2001 年提出的基于 De Bruijn 图的组装由于其路径收缩计算时的数据依赖，如果直接对该策略进行并行实现并不能提升该策略的计算并发度。为提高组装算法的并行度，深入解耦组装算法中的计算数据依赖，本文需要一种适合并行的高可扩展的并行序列组装数学模型。现有的 De Novo 组装软件已经取得了许多重大进展，然而这些算法并不能高效快速处理 TB 级别的基因组组装，对于 TB 级的基因组分析愈来愈强地受到高质量、高可扩展 De Novo 组装策略的的限制。

（2）高效并行的基因组装计算模型问题

基于 De Bruijn 图的基因组装算法本质上是图的路径收缩算法，而高可扩展的图算法是当前研究的热点。 顶尖大学和主流数据分析公司均在研发自有的图分析算法或系统。早期的并行图分析系统是 2005 年发布的 Parallel Boost Graph Lab(PBGL)[42,43], 现广泛应用于互联网公司业务系统开发，但该系统只能处理 2.5 亿顶点，扩展性能(128 个核)有限。Pregel[44]及其衍生系统 Giraph[45], HAMA[46], GPS[47]随后出现，这一系列系统系统效率不高，可处理的顶点规模也在 10 亿规模，扩展性（几百个核心）有限，同时 Pregel 对拓扑结构不断变化的图的支持度现在依然并不成熟。基于 Spark 框架的 GraphX 图计算系统现在支持了六种图算法，但对于基因组装的图收缩算法依然还处于前期研究阶段。

为优化大规模图的计算并保证其在高性能平台上的扩展性和高效率，由此本文需要一种高效并行计算机制以最大化的提高图算法中计算的并行度。同时在实现该并行计算模型时， 本文需要考虑如果从更高的抽象层次上去抽象表达具体的图算法，如何证明该并行模型具有高扩展性。具体实现中如何保证高效通讯模式，负载均衡策略，以及如何设计 API 支撑图算法表达。

(3)基因组装的深度优化 （计算，质量，多数据源）

为追求更高性能和处理更大规模的 TB 级别的基因组组装，本文需要对基因组装算法及系统做极致的优化。这里面临下面四个问题： 1) 基因组组装等价于在一个 De Bruijn 图上的一个旅行售货员问题，而这个问题是 NP 难的问题。现在研究者更多是在寻找一种近似算法来解决该问题[48]。 2)由于海量的基因测序数据会产生超大规模的 De Bruijn 图。 例如在千人基因组数据中大约有 200TB 的人类基因测序数据[49,50]，由于测序数据中的每个碱基可对应于一个 kmer，那么这套数据中产生的 De Bruijn 图将会产生大约 2^{47} 个 kmer，而这个图是现在 Graph 500 排名的第一名可处理的图规模还要大 128 倍[51]。3)由于测序仪通常测序产生的序列带有错误，所以这里通常会使得大约 50%～80%的 k-mer 是错误的[52]。因此图中多数的顶点和边的可信度是依赖于用户的容忍度的。4) 对于多样的并行文件系统，通讯硬件设备及其支撑的硬件平台[53]，其达到最大性能的 IO 读写文件块的大小， 通讯消息的负载，消息压缩的算法是不同的，如何使用多重优化的技术，使得整个基因组装的数学底层库达到最佳的系统性能，并使得整个系统的性能得以最大的释放。

面对目前超大规模（TB-PB 级别）基因组 De Novo 组装分析的挑战，本项目将重点设计面向大规模数据的基因组装数学模型；高效并行的基因组装计算模型问题；基于内存优化和通讯深度优化的基因组装软件。基于超级计算机和高性能并行算法设计，本项目将针对 TB-PB 级基因组数据，结合基因数据纠错，contig 扩展以及精度保证，研发高可扩展的基因组 De Novo 组装高效分析方法，进而大大提高基因大数据分析的效率和质量。

1.3 本文的贡献

为处理 EB 级的海量基因组数据，本文研究如何结合超级计算机和新型并行计算模型来设计新的高可扩展的基因组装算法，并开发高可扩展，高质量，高精度的基因组组装软件。从而从根本上解决 EB 级生物信息数据的处理速度和处理规模，并使组装结果在质量和精度上都达到现有软件标准。本文的主要贡献包括：

1. **基因组组装理论模型重建：** 针对当前基于 De Bruijn 图的基因组装策略并行化难度高的特点，提出了双向多步 De Bruijn 图(MSG)，使得基因组装问题转化为一个可并行的半群计算模型。从而解决基因组组装上的数据耦合问题。本课题还将对其并行可行性进行分析与论证。基于 MSG 的并行组装策略的复杂度已经达到本领域最低，即计算复杂度和通讯复杂度上分别达到 $O(n)$ 和 $O(g(log(log(g))))$, 这里 n 是输入数据的碱基数量，g 是参考基因组的长度。
2. **高效并行的基因组装计算模型及组装软件：** 针对半群计算问题，本课题提出了 SWAP 的全局异步子集同步的并行计算模型。该模型可自动挖掘该类半群计算问题中的潜在的并行性，同时本文给出该计算模型的算法复杂度，通讯复杂度和加速比的计算公式。最后基于该计算模型本文开发了一个并行组装软件 SWAP-Assembler，该组装软件达到贡献 1 中所要求的计算和通讯复杂度。
3. **基因组装软件底层深度优化：**针对文件系统，通讯硬件设备、支撑硬件平台的多样性以及组装软件中计算、通讯、I/O 同步优化的干扰问题，本文为 SWAP-Assembler 组装软件的各步骤了参数空间，并基于该参数空间进行多层次多维度性能自动优化以挖掘底层计算平台性能极限。为深度优化组装算法的底层库，本文使用并行图数据的 streaming I/O 技术,内存数据压缩技术，消息通讯聚合压缩技术以提升现有 SWAP-Assembler 的问题规模，系统效率和扩展性。
4. **组装软件的高扩展性：** SWAP-Assembler 的扩展性测试实验结果表明 SWAP-Assembler 在组装现有最大的基因组数据-千人基因组数据（4TB）时，可以扩展到 131，072 核心，并保持 40%的系统并行效率。 除此以外，SWAP-Assembler 的在组装炎黄基因组时，运行速度是现有最快的基因组装软件 HipMer 的 3 倍。在使用 65，536 核心时，可以在 1 分钟内完成组装，而 HipMer 最高只能扩展到 15，000 核心。
5. **组装结果的高质量高精度：**SWAP-Assembler 使用启发式 contig 扩展策略，以及变长 kmer 扩展技术来提高 contig 长度并保证 contig 的高精度。在组装规模较小的两个基因组(S.aureus 和 R.sphaeriodes)数据集时 SWAP-Assembler 组装结果的 N50 长度在参与比较的五个组装软件中居于第二，而其组装的 contig 的精度位据第一。对大规模的基因组数据（Hg14, Fish 和 Yanhuang）进行组装的结果在所有五个组装软件中，SWAP-Assembler 的 N50 长度最长，组装精度依然最高。

最好因此对于那些更加注重精准的 contig，同时也希望 contig 比较长的用户 SWAP-Assembler 是首先工具。

1.4 论文的组织

本文使基于性能集群对新型并行计算模型研究并实现了高可扩展，高质量，高精度的 TB-PB 级基因组装，从根本上提高了 TB-PB 级生物信息数据的处理速度和处理规模，保证了基因组装的质量和精度。论文共分七个章节对本文的研究课题进行论述，主要章节安排如下：

第一章，引言部分对本课题研究背景、研究意义、研究现状存在的研究问题，以及本文组织架构、研究内容作了概述。

第二章，阐述了基因组序列测序技术和组装技术发展历程，对相应技术发展过程中所兴起的组装策略进行技术分析和对比，同时本文就了基于 De Bruijn 图并行化基因组装策略进行介绍，并对现有主要的六个基因组并行组装软件的技术细节进行技术介绍和优缺点分析，同时使用了三个数据集对其中的四个已经开源的组装软件在单个服务器上的扩展性进行了分析和对比。

以上一章为铺垫，第三章结合基因组组装问题的 De Bruijn 图的策略，提出了更加适合并行的双向多步图的数学模型来抽象基因组组装问题。通过推理与证明，该数学模型的两个重要性质直接证实该数学模型的与原始问题等价性和正确性。同时，由于双向多步图的数学模型满足结合律，所以图中无共同顶点(k 分子)的半扩展边的合并是可以独立合并，而且乱序执行的， 而这样的性质使得该数学模型更适合并行化。

第四章节中，本文提出了异步并行计算模型 SWAP，并开发了基因组装软件 SWAP-Assembler，SWAP 模型可以最大化的半群系统上边合并操作的并行度。接着，本文对该异步并行计算模型的具体实现算法，进行计算通讯复杂度抽象和分析，最后从理论上证明该异步计算模型及其组装软件 SWAP-Assembler 可以达到近线性的扩展性。

第五章，本文对基因组装软件 SWAP-Assembler 每个模块的运行时间进行分析，并发掘最耗时的三个模块，接着对这三个模块进行性能瓶颈分析并对每个模块提出性能优化方案以大幅提高序列组装软件的扩展性和并行效率。最后整合以上优化技术的组装软件被命名为 SWAP2，该软件相比于 SWAP-ASssembler 在使用 16，384 核心时计算性能加速了 13 倍，同时在 4TB 的千人基因组数据上可以扩展到 131，072 个核心。

第六章，基于收集的 5 个物种的测试数据集（即 S.aur, R.sph, Fish, Yanhuang, 千人基因组数据）以及四个主流的基因组装软件（Velvet, SOAPdenovo, Ray, ABySS），对本文所研发的基因组组装软件的性能和结果质量精度从多方面进行了综合分析和评价。

最后，对整个论文进行了总结，并总结了下一步的研究工作。

第二章 基因组装前沿技术分析

基因测序技术的推进和基因数据分析技术的更新带来了基因领域的革命。生物学家和计算科学家正协同研究和开发整个基因组的测序技术，而这个技术如果离开一项复杂的分析工具–**基因组装工具**–就不可能完成。 这里本文将对过去 35 年来，从组装技术出现开始到现在所有基因组装技术的关键算法做比较全面的介绍。

本文中，本文只关注 DNA 测序仪将非常多的 DNA 碎片测序读出后，利用这些数据对未知基因组的 DNA 参考序列进行重构的过程，即**基因组组装**。基因组组装，简而言之，就是设想一本杂志被复制成多份，将每份杂志均以不同的方式剪切，将多份剪切的杂志放在一起。在剪切的过程中，一些碎片丢失，一些碎片被污渍浸染，一些碎片存在着重叠现象。根据上述情况来寻找恢复原始杂志的方法。这是 DNA 序列组装问题的现实模型描述。

2.1 基因组序列组装的技术发展

当今 DNA 测序技术的迅猛发展，当前的基因测序设备依然只能够读取 DNA 参考序列中一个非常小的片段(read)， 而这个片段的长度和精度（错误率 Error rate）随着测序技术的变化而变化，例如 Illumina 测序仪可以测出的序列片段(read)的长度是大约 100bp，每个碱基(base pair)的错误率大约是 1～3%，这里的碱基错误率有偏倚特性即中间错误率低，两端错误率高[54]。Pacific Biosciences 测序仪能够测出的序列片段（read）的长度大约是 10～20kb，碱基测序错误率大约是 15%，这里的错误率没有偏倚[55]。然而，本文实际中需要测序重构的 DNA 参考序列可能达到 3Gb 的碱基，即使是非常小的细菌也有大约 1Mb 的碱基。所以还原原始基因组的 DNA 参考序列需要将所有的测序片段拼接起来。

早期 DNA 参考序列的重构主要依靠不断的对重叠基因区域的测序，见图 2.1。这里每个对应序列的位置和方向都在实验前设置好了，最后只需要将每个独立的片段连接起来形成一个完整的参考序列。虽然上面这种实验的做法非常简单，也不需要计算机的帮助，但具体的实验过程，实验员只能在前一段测序好后再测后面的一段，整个过程非常耗时耗力。这个方法一直延续到 1979 年，直至一种新的技术叫做“鸟枪测序”的出现[56]。这个方法通过将原始 DNA 序列扩增后再随机的打断为多个片段，接着同时对这些片段进行测序，以提高测序技术的数据通量。 由于原始 DNA 序列是随机打断的，所以这些序列片段会和其他的序列片段有重叠，本文可以通过比较这些序列片段来找到重叠的区域，从而将相邻的序列片段拼接起来。这里研究人员使用计算机来提高这个计算过程的速度，随之也就诞生了最早的基因组组装软件，这是**基因组装的第一个转折点**。图 2.1

为 DNA 测序技术流程图。其中蓝色线是未知的被测序 DNA 参考序列，灰色线是测序得到的序列，其中 (a) 引物测序，(b)随机分解测序，(c)鸟枪测序。

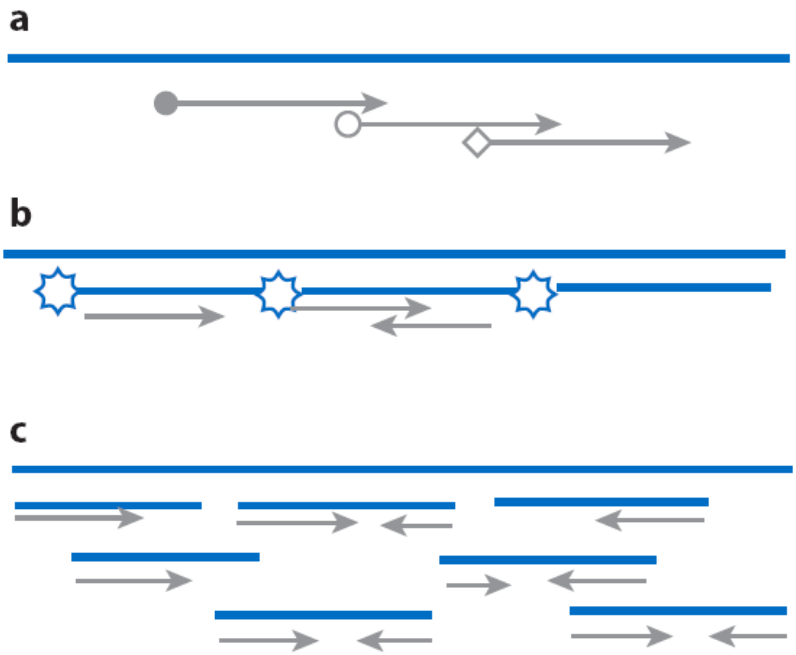

图 2. 1 DNA 测序技术

“鸟枪测序”的简洁性吸引了大量数学和计算机方面的科学家的关注，并带动了基因组序列组装理论基础的快速发展。最开始科学家们关注在什么条件下基因组参考序列是可以被重构的问题。Lander & Waterman[57] 基于数据的关键特征提供了一套数学公式用于确定重构的可行性问题，结果表明基因组被扩增至少 10 倍时基因组参考序列可被有效重构回来。接着 Ukkonen 和其他科学家研究了组装问题的计算复杂度问题，并将组装问题抽象为最短公共超串问题[58]，即最终被组装的基因组参考序列是包含所有测序序列为其子串的一个最短 DNA 序列。 他们的研究结果表明找到正确的参考序列需要指数级别的计算复杂度，因此该问题是 NP 难的。尽管这是一个消极的结果，但科学家依然尝试去找到一个近视的超串，而且这个超串被证明是在一定误差容忍限度内接近于理想的最短超串，并由此诞生了第一个实际可用的序列组装软件。

最短公共超串忽略了一个基因组的重要复杂特征：重复序列。很多基因组的 DNA 片段可能会在其他位置重复出现一次或者多次。换句话说，正确的基因组 DNA 参考序列可能不是最短的。因此显式的将重复区域考虑到新的基于图模型的基因组装中会对现有的基因组装软件形成偏见。 总的来说，重复区问题，以及该问题在组装过程中引入了更高的复杂性，使得基于“鸟枪测序”的基因数据的基因组精确组装依然是无法计算的[59, 60, 61, 62]。

这里本文对基因组组装的早期研究工作做了简要介绍，本文想强调的结果是在过去几十年间，理论结果的不可行性和实际组装工具的成功明显脱节。 这一切成熟的理论和实践之间的差距的一种解释是，理论性的研究结果是基于最坏的情况，这在实践中很少

发生。序列组装的实际复杂度取决于序列的大小和重复[63]，当测序序列的长度长于重复区域时，组装问题可以解决，而当测序序列的长度短于重复，组装结果是不明确的，因为可以重建指数级的基因组来包含输入测序序列[64]。本文将在接下来的讨论中重新审视这一权衡。

2.2 主要的基因组装策略

正如上文所述， 科学家们使用数学理论工具分析和实际算法程序来对基因组组装问题的复杂性进行理解和探究，其主要的方法和策略可以被归纳为以下几种。

图 2.2 为组装策略分类。(a)包含若干重复区域的 DNA 参考序列以及鸟枪测序的测序序列，其中相同颜色片段即为重复序列。(b)贪心策略，因为重复区问题导致其组装失误。(c) OLC 策略，其中每个测序序列都是图中的一点，互相重叠的序列有边相连。 (d) De Bruijn 图策略，其中图中的每个顶点都是 k-mer。(e)串图策略，这里图中的传递边已经被删除，可以看到这里的重复区比 OLC 图中的更为明显。

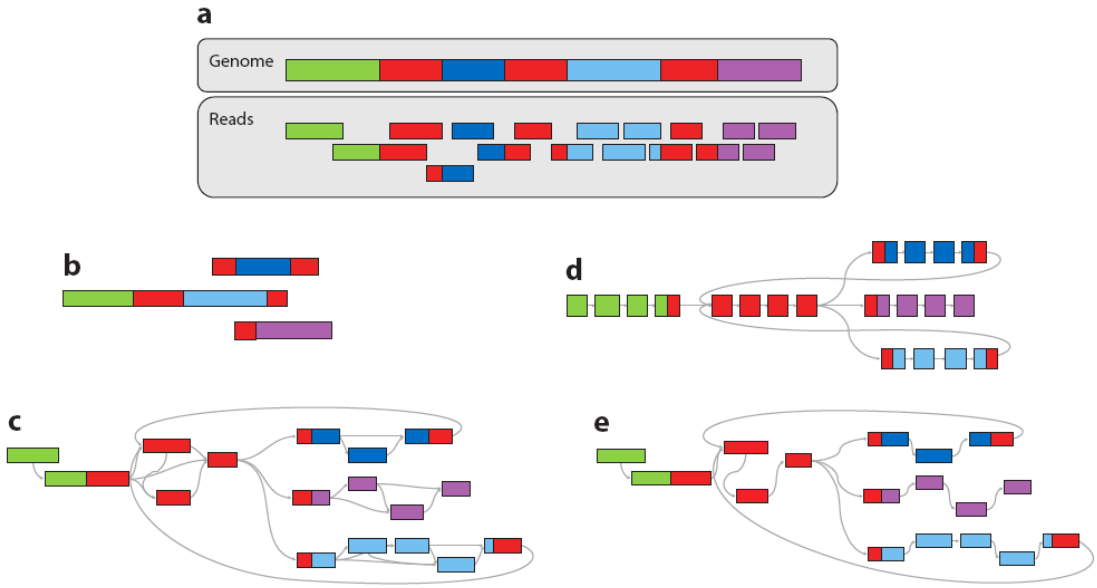

图 2. 2 组转策略分类

2.2.1 贪心策略

基因组序列组装中的一个最简单的策略就是按照两两序列重叠区域的长短顺序（或者按照其碱基质量加权后的长度）迭代的将序列拼接起来。换句话说，这个过程开始将两个互相重叠最多（或者长度最佳，碱基质量加权后长度最佳）的测序序列拼接为一条序列，然后重复这个过程直到两个序列间的长度达到前期预先设置的重叠长度的阈值。作为结果，新生成的重叠序列（即 contig）的长度延展主要是通过与其他测序序列或者其他重叠序列(contig)合并来达成的。测序序列如果与已有的重叠序列冲突则会被忽略。这一组装策略称为贪婪策略，因为它每一步总是做贪婪的局部最优选择。尽管该策略比较简单，这种方法和它的变种算法提供了一个很好的近似最佳装配方法。

许多早期的基因组序列装配依赖于这样一个贪婪策略。在这些软件中，本文想强调软件 phrap[65]，他是人类基因组计划中使用的的最主要的公开基因组序列组装软件。TIGR 组装工具[66]，它是第一个对活着的生物有机体进行测序后重建其基因组参考序列的软件；接着就是 CAP 系列组装软件[67]，它已被大量用于重建人类和许多其他生物的转录组。贪婪方法主要用于对早期第一代测序仪产生的数据组装，同时也探索了一些可能的方式来处理最早期的二代测序仪所产生的测序数据[68, 69]。

尽管其早期巨大的成功，贪婪的策略有其严重的限制：因为它的局部最优的性质，它不能有效地处理重复区域(repeats)，如图 2.2 所示。由于基因组测序的大小和复杂度的增加，贪婪策略被替换为更复杂的图为基础的策略，该策略能够更好地抽象和解决高重复的基因组序列。

2.2.2 基于图的策略

对基因序列组装的理论研究促进了基于图表示的基因数据[59, 70]的实际组装软件的开发。基于图的组装模型使用图中的顶点和边来表示基因测序序列和他们的关联关系。基因序列所构造的图描述了一个基因序列的拼接顺序。序列组装试图找到一个最佳路径来重建潜在的基因组参考序列，并尽力避免由重复区域(repeats)所造成的错误路径。

2.2.3OLC 策略

OLC 图是最简单的图模型，每个基因测序序列被表示为图中一个顶点，两个基因测序序列如果有重叠则对应的这一对顶点有边连接（见图 2）。另一种替代的表示方式是用一对顶点表示一个基因测序序列，一个顶点代表基因测序序列的开始，一个代表基因测序序列的结束，两个顶点之间的边存有这个基因测序序列的字符串信息[71]。

在这种表达方式中，在一个测序序列和另一个测序序列的终端顶点的重叠区域信息被表达在两个序列之间的边上。不管图的具体表示形式如何，组装过程通常遵循三个主要阶段。首先，重叠区域检测(overlap)。其次，图的构造(layout)，并找到一个基因测序序列适当的顺序和方向（布局）。最后，用图中测序序列的顺序和方向计算出一个 consensus 序列，所有 consensus 序列将由组装程序输出为 contigs。这种模式被称为基因组装装配方法的三个主要阶段：Overlap, Layout, and Consensus。

其中 Overlap 步骤通常需要大量的计算时间。你需要用动态规划算法计算所有的测序序列之间是否有显著的重叠（由重叠的长度和在重叠区域的相似性确定的）。这样的算法需要 $O(n^2)$的计算时间，其中 n 是测序碱基的总数。这种蛮力的方法可以用来组装非常小的基因组的序列。为了加速 Overlap 的重叠检测，可以用一个基于 k-mer 的索引来构造图，其中 k 的大小介于 16～24 之间。该索引是用来快速筛选的测序序列之间可能的重叠区域，然后进行动态编程，加以验证重叠。这种技术大大减少了搜索空间，并得到了广泛的应用[66, 72, 73]，特别是为了避免对重复 kmer 的重复比对，相关应用做了很多改进。

因为在 layout 阶段的全局最优解的计算在理论上是不可行的，layout 步骤通常试图产生 unitigs，即使用不太激进的策略将不会拼接错的测序序列先拼接起来[70]。组装程序通常会去除低质量序列和重叠，然后删除多余的边，这一过程是具有传递性的，例如，如果图中包含的三角形 A-B-C，然后边 A-C 可以删除，因为它是 A-B-C 的冗余路径。最后，layout 算法可以找到一个简化的确定图。最后根据测序序列的顺序和方向，一个多字符串比对算法将最终的一组 consensus 序列构造出来输出为 contigs。

这种组装方法已经非常成功。采用基于毛细管测序的样本, Celera 构建了人类基因组序列的全基因组鸟枪法测序技术[74]。尽管 OLC 方法在基于毛细管测序的序列数据上取得了成功，这一策略却无法处理第二代 DNA 测序仪器产生的大量短序列测序数据。此外 overlap 重叠计算步骤也是一个性能瓶颈。这个策略在面对上百 GB 的大规模测序序列时，已经变得不切实际。即使是一些优化索引技术来减少相似序列比对的搜索空间，但由此产生的重叠图的大小也被证明有问题。重叠图的边数随着测序深度的提高和重复区域的拷贝数量的平方成正比。当面对高深度的数据和许多错误的重叠信息时，产生的图可能是不切实际的大。由于这些原因，组装高通量短序列测序数据的大部分工作都依靠 De Bruijn 图的方法，下文将有详细介绍。

2.2.4De Bruijn 图策略

基于 De Bruijn 图的组装策略，其理论工作开始于从上个世纪 80 年代后期，Pevzner[75]研究了一个通过使用连续的已知 k-mer 来重建基因组参考序列的问题。这一理论工作和其后续发展[76, 77]奠定了基于 De Bruijn 图的全基因组组装策略的基础。

在这种类型的组装策略中，每个测序序列(read)分解成一系列连续重叠 k-mers。在测序序列中连续的 k-mers 会用边连接。然后，这个组装问题可以被抽象为寻找一条遍历每条边有且仅有一次的路径的问题，即欧拉路径问题。在实践中，测序错误和抽样偏差也隐藏在 De Bruijn 图中，所以寻找整个图的一个欧拉路径通常是做不到的[29, 77, 78]。即使在整个图的欧拉路径可以发现，由于存在的重复它也不可能准确反映基因组的参考序列，因此这里有指数级的欧拉路径，但其中只有一个是正确的[79]。在大多数情况下，本文的组装软件都只是试图重构 De Bruijn 图中无模糊，无分支区域的序列作为 contigs。

虽然 De Bruijn 图开始只是为毛细管测序数据而开发的组装工具，但它一直到高通量短序列的二代测序技术的出现，才在组装领域显现出实用价值。De Bruijn 图在计算方法上相比于 OLC 策略的一个显著的优势是：它不需要测序序列两两间的重叠匹配区域，所以也不需要耗时的动态规划算法来计算重叠区域。相反，测序序列之间的内容重叠信息是隐藏在图结构中的。而构建 De Bruijn 图也非常容易：首先，从测序数据中切割出 k-mers 作为图的顶点，接着，测序序列中相邻的 k-mers 被添加有向边。只要用合适的数据结构存储这个图，那么这个构造过程将在硬盘中读取完测序数据后立刻构造完毕。

早期的 Zerbino&Birney[79]和 Chaisson&Pevzner [80]将该策略应用到在细菌基因组数据的组装中。这个组装策略的结果受到短序列长度的局限（超过 k-mer 长度的 repeat

无法解耦），**但该策略在计算复杂度上的革新是该领域的第二个转折点**。

然而，由于 De Bruijn 图策略的内存使用的成本太高，高通量基因组测序数据的大型组装需要进一步的算法优化。基因测序数据中的每个碱基都近乎可以产生一个 k-mer，这样为哺乳动物所构造的 De Bruijn 图将会有几十亿个顶点。而测序仪引入的测序错误也被整合到了这个图中从而导致基因组参考序列被打乱。而这些测序错误会引入新的顶点和边到这个图中，这会显著扩大其规模（通常会扩大 5 倍左右）。那么如何用最小的内存量来表达 De Bruijn 图成为这个领域的一个关键的问题。

ABySS 组装软件[32]介绍了一种利用分布式哈希表来存储 k-mer 的数据结构。由于每个 k-mer 都只有 8 种可能的边来扩展到 k-mer，所以只需要一个字节来表达边是否存在。而这个哈希表被分布在一组计算机中，当遍历这个 De Bruijn 图，分布式哈希表可以查询并恢复它的邻居顶点。这种分配策略允许从长度为 36 的测序序列中将一个人类基因组参考序列的组装出来。

接着使用 k-mers 来表达 De Bruijn 图的方法被 Conway & Bromage 使用 [81]。他们的核心思想是使用一个在区间[0, 4^k)的数来表达 k-mers，并将对应比特数组的位数设置为 4^k。这个比特数组可以被查询并用于恢复其邻居顶点和边。 而存储一个大小为 4^k 的数组显然不太实际。为了解决这个问题，Conway & Bromage[82]使用了稀疏位数组编码技术来表达这个位数组。最终当使用 27-mer 时，该算法的平均用来存储每个 27-mer 的内存大约是 28.5 比特。这相比于 ABySS 的每个 kmer 需要消耗 64 比特位是一个很大的提升。

近年来，使用布隆过滤器(Bloom filter)来存储 k-mers[83]也开始得到普及。布隆过滤器只用了一个很小的比特数组来表达一组数的存在与否。和 Conway & Bromage 的方法不一样，它直接用一个比特数组来对一个 k-mers 进行索引，而布隆过滤器的使用多个 hash 函数计算，并将 k-mer 映射到的一系列比特位置 1 以表示该 kmer 的存在。在查询时间时，使用相同的哈希函数，找到相应的映射位并检查是否全部为 1，如果设置，布隆过滤器声称 k-mer 是存在的。该算法保证了如果一个 k-mer 被添加到了布隆过滤器中，那么他一定会被查询到。但是，相反的则是不一定能保证正确，即如果一个 k-mer 没有被事先加入到布隆过滤器中，也有可能被检查出来该 k-mer 是存在的。这种假阳性发生的概率是由底层的位向量的大小和使用的哈希函数的数量控制的。为了减少内存使用，就必须要容忍较高的假阳性率。

布隆过滤器最早被 Melsted & Pritchard Pell et al.[84]用于 k-mer 统计。接着 Pell et al.[85]等人利用布隆过滤器，将宏基因组构造的 De Bruijn 图分割为多个子图，然而再对各个子图进行分别组装。 他的论文结果表明图分割和子图组装可以容忍大约 15%的假阳性率，这时每个 k-mer 只消耗大约 4 个比特。接着 Chikhi & Rizk [86]通过扩展该理论并开发了一个全基因组组装软件。

Ferragina & Manzini [87]提出的 FM-index 数据结构开始是被用于基因序列比对中[88, 89, 90, 91]，现在该技术也开始被用于序列组装。通过对测序序列构造 FM-index 结

构可以查询任何k-mer的存在或不存在[92]，而这个表达所需的内存量与k无关。FM index可以用来表达一个 k 阶 De Bruijn 图，这里 k 最大可达测序序列的长度。 最近有两个研究组开发出了使用 FM index 的数据结构来表达的 De Bruijn 图[93, 94, 95]。而 De Bruijn 图的一些性质可以被利用来减少内存消耗，Ye 等人[8]观察到只有一小部分的 k-mers 需要直接存储为顶点，该技术被整合到 SOAPdenovo 中用来减小组装程序的内存消耗[71]。

2.2.5 串图策略

De Bruijn 图有一个非常好的重复区属性：一个重复区域(repeat)所有副本在图中表现为一个具有多个进入和多个出去的点独立的大块序列片段。这为基因组结构提供了一个简洁的表示。在 2005，Myers[96]也在 OLC 策略构造的图中通过两步转换也发现了类似的属性。首先，把包含在由其他测序序列拼接得到的字符串中的测序序列删除，接着，传递的边也被删除，最后得到的图形，称为串图，该图的许多特性与 De Bruijn 图相似。Edena 组装软件就是使用串图策略实现的早期短序列基因组装软件[97]。 接着后续就有针对为大型基因组的基于 FM index [98]的高效内存的串图策略的理论研究，在此期间[99, 100, 101]，一些高效串图构建算法也被开发出来。

2.2.6 并行 De Bruijn 图策略

为了处理大规模的基因测序序列，除了压缩 De Bruijn 图的内存消耗，同时也需要提高 De Bruijn 图策略的计算性能，因此并行 **De Bruijn 图策略的出现是该领域的第三个新的转折点**。

由于基于 De Bruijn 图的组装策略的算法复杂度接近线性，这一特性使其广泛用于下一代高通量基因序列的组装中，也使其成为基因组装策略的主要并行化策略。为了便于介绍并行化 De Bruijn 图策略，本文简要介绍 De Bruijn 图组装的基本过程：

1）假设基因序列集合为$S=\{S_1,S_2,\ldots,S_n\}$，其中每个基因序列可以被切割为若干连续碱基组成的 k-mer 集合$K=\{k_1,k_2,\ldots,k_n\}$，每个 k-mer 都对应于图中的一个顶点。其中，k-mer 的切割方法为：选取一个长度为 L 的基因序列，先以基因序列的左端为起始位置，截取长度为 k 的碱基字符串，再将起始位置向右移动一位，截取第二个长度为 k 的碱基字符串，最后直到到达基因序列的右端，这样总共可以切割出 $L-k+1$ 个 k-mer 出来， 这些 k-mer 组成了 De Bruijn 图中的顶点。这里本文为了防止 DNA 序列中互补双链中截取的 k-mer 相同，在此k 值为奇数；

2）在上述构造的k-mer集合$K=\{k_1,k_2,\ldots,k_n\}$中，若存在两个k-mer，其中k_1的后$k-1$个碱基与k_2的前$k-1$个碱基字符子串相同，且这两个 k-mer 在一个序列中为连续的 k-mer，那么k_1、k_2之间存在一条由k_1指向k_2的一条边。根据基因序列中连续分割开的 k-mer 相邻信息，本文将两个 k-mer 用有向边进行连接，同时每条基因序列实际上也对应着图中的一条 kmer 组成的路径；

3）根据上述构造的 De Bruijn 图中的有向路径集合，本文寻找一条经过所有边一次且仅一次的路径，即将拼接问题转化为图论中寻找欧拉路径求解。根据 DNA 序列中的双端 pair-end 信息，对欧拉路径进行解耦，最终找到一条近似的连续的 DNA 序列。

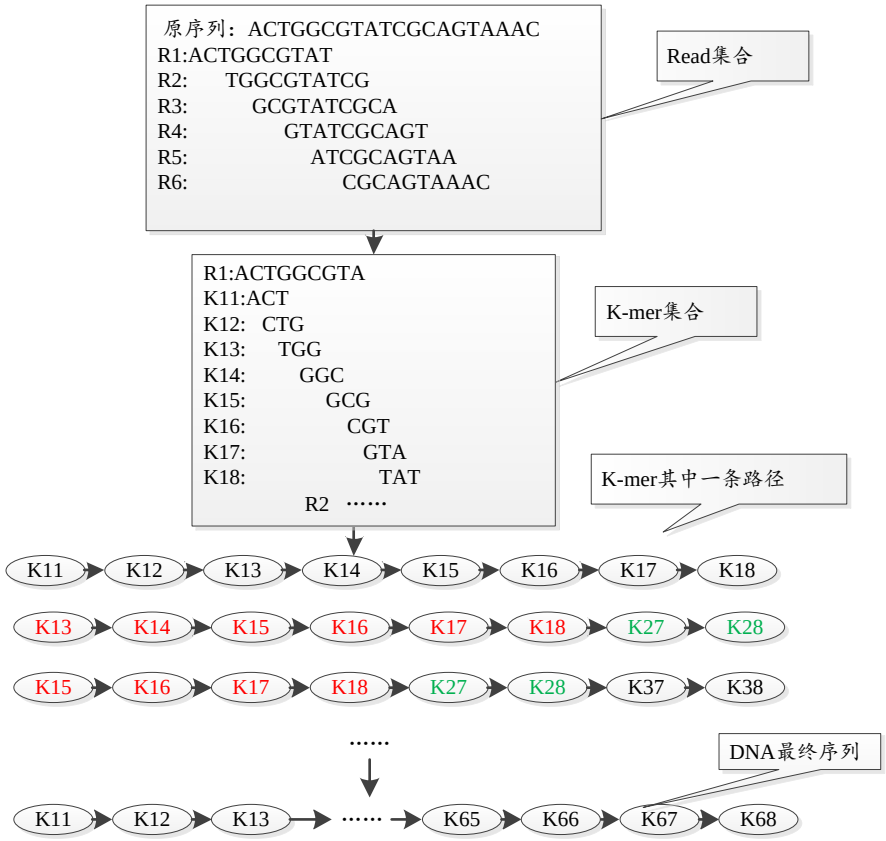

图 2.3使用基因序列构造 De Bruijn 图的过程

基于De Bruijn图的序列组装策略相比于基于Overlap-Layout-Consensus(OLC)的序列组装方法，由于以下两个优点使得其并行化的效果比较显著：

1）在 OLC 中的第一步 Overlap 步骤中，采用了序列比对算法来寻找基因序列之间的两两重叠信息，该算法的时间复杂度为$O(n^2)$，其中n为 DNA 序列中 read 的数量。当前二代高通量基因测序数据的序列长度越来越短，对一个物种进行测序，每小时将产生 TB 级别的数据，这使得 OLC 策略的计算量增加到无法承受；而基于 De Bruijn 图策略的基因组装原理，抛弃了 OLC 中关键的序列比对算法，取而代之的是采用以 k-mer 为图中顶点构建图，从而去掉序列比对算法所消耗的时间，提高了算法的效率；

2）在 OLC 策略中，其最终将基因组装算法转化为寻找哈密尔顿路径，而在图算法中寻找哈密尔顿路径问题本身是一个典型的 NP 完全问题，难以求得最优解；在基于 De Bruijn 图的序列拼接中，最终将拼接问题转化为无分叉路径的收缩算法，而该算法本身是一个多项式可解问题，具有更优的时间复杂度。

但是基于 De Bruijn 图的求解，也存在一些问题。在计算过程中，由于重复序列、测序错误，使得基于 De Bruijn 图的组装，最终可以得到的结果非常多的可行解，但这些解

中可能只有一个的最优的，为了找到尽可能最优的解，本文需要借助大量的生物辅助信息来确定最终的组装路径；另外基于 De Bruijn 图的组装的基本结构是 k-mer，这使得构建 De Bruijn 图中的顶点数目巨大，而图的规模增大会直接导致对内存的需求程度提高。在下一节中本文将对现有的并行组装软件进行理论分析和性能测试。

2.3 现有并行组装软件分析

目前基于 De Bruijn 图实现的并行基因序列组装软件并不多，这里选取当前影响力较大的六个并行组装软件，SOAPdenovo[30] ，PASHA[33], ABySS[32], Ray[102], YAGA[39], HipMer[40, 41] 进行核心技术的分析和评价，最后本文用实验对性能进行性能评测。

2.3.1 SOAPdenovo

为了解决大规模基因组数据的组装问题，2009 年华大基因 Ruiqiang Li 提出基于 De Bruijn 图策略开发实现了 SOAPdenovo 组装软件[30]，该软件可以处理 300G 左右的人类基因测序数据，测序数据的测试深度可达 71 倍。

SOAPdenovo 的组装步骤主要分为纠错处理、De Bruijn 图的构建、Tip 移除、重复序列解耦、bubble 融合、contig 构建六个步骤。其主要特点是：在构建 De Bruijn 图之前，采用了预处理纠错机制，该机制使用哈希表结构来存储每个 k-mer 出现的频率，并设置了 k-mer 频率阈值，将 k-mer 出现频率低于阈值的 k-mer 去除，预处理，使得 k-mer 的错误率大大降低，简化了图的结构；SOAPdenovo 借鉴了 Velvet 也采用 Dijkstra 算法来处理序列中的 bubble 信息，使其平均覆盖度增加；另外，SOAPdenovo 利用了双端信息匹配重叠区域将 read 合并，最后生成了以 contig 图的顶点的图，大大简化了 contig 图的复杂性。

SOAPdenovo 仅在预处理纠错机制上，引进了多线程技术，在多线程上建哈希表结构以完成纠错机制，同时降低了内存消耗，提高拼接效率。数据显示该软件处理 300G 人类的数据量，总耗时为 48 小时。在整个拼接过程中，基因序列的预处理纠错时间是 24 小时，占整个拼接过程的 50%的时间，耗时最多；在构建 De Bruijn 图时内存消耗最多，高达 140GB。

2.3.2 PASHA

通过观察和分析基于 De Bruijn 图的基因组组装的各个步骤的时间分布结果，Yongchao Liu[33]发现基于 De Bruijn 图的组装策略的主要耗时步骤是 k-mers 的切割，k-mers 分布式存储，以及 De Bruijn 图的构建和简化。Yongchao Liu 使用线程化技术将这两个步骤进行优化并开发了 PASHA 组装软件。然而由于 De Bruijn 图中的无分叉节点的路径收缩过程中具有计算依赖性，每次对每个无分叉节点的路径可以最多分配两个线程

进行运算。虽然该简单的优化策略虽然比 SOAPdenovo 更加深入，但依然无法解耦基因组组装过程中的计算依赖问题，即无法向一个无分叉路径中投入更多的线程。同时由于其只有线程级别的优化，所以该软件只能在单机上进行扩展，并受到单机的内存大小的限制，而无法对大型基因组的海量基因数据进行组装。

2.3.3 YAGA

YAGA[39]是第一个使用边来表示 De Bruijn 图的并行组装软件。该算法为了避免合并两个不同计算机上存储的两个 k-mer 带来的低效率高延迟的问题，YAGA 使用了一个 list ranking 的算法来将属于一个无分叉路径上的边都通过排序聚合到一个计算机上。 这样在对无分叉路径上的边进行合并时，都是本地化的计算操作，而不需要远程的通讯。但是该算法的核心是一个分布式的 list ranking 算法，而这个算法的计算，通讯，通讯轮数的复杂度界定了 YAGA 算法的复杂度。这里 YAGA 可以达到 O(n/p)的计算复杂度和通讯复杂度，其计算轮数的复杂度达到 O(log(n))， 这里 n 是测序序列的碱基总数，p 是进程数。但是 YAGA 算法所使用的 list ranking 还需要的空间复杂度是 O(nlog(n)/p)，所以会导致该算法需要耗费大量的内存。

2.3.4 ABySS

为了解决新一代测序仪产生的大型基因组序列组装问题，J.T. Simpson 等人开发出 ABySS [32]。ABySS 采用了 MPI 并行编程技术，将 De Bruijn 图构建于并行计算机上，减少了单个计算机内存的压力，突破了服务器内存消耗的限制，减少了序列组装时间。

ABySS 的组装过程主要分为四个步骤： 分布式 De Bruijn 图建立、错误信息处理、De Bruijn 顶点融合、contig 合并。其主要贡献是：使用 MPI 并行编程技术将 De Bruijn 图中顶点分散存储于各个进程，降低了数据存储对单机造成的内存消耗；在 De Bruijn 图的构建过程中，将相邻 k-mer 信息存储于边的信息中，从而简化了顶点搜索和查找的复杂度；最后通过充分利用 Pair-end 信息，在组装过程中将长度不等的 contig 进行融合得到更长的 contig，从而提高了组装质量。实验数据显示，ABySS 使用 MPI 并行编程技术，降低了单机的内存消耗，加快了序列的组装速度。

2.3.5 RAY

Boisvert 于 2010 年通过开发一个通用的分布式计算引擎来并行化 De Bruijn 图的组装算法，并形成了一个并行基因组装软件 Ray[102]。在这个软件中，他通过使用启发式的贪心策略来计算两两测序序列两个方向上的重叠区域，并将 k-mers（原文称之为 Seeds）扩展成为 contigs。论文中的性能结果显示，Ray 在处理人类基因组第 14 号软色体的时候可以扩展到 512 核心。然而这个扩展性对于很多大型基因组例如千人基因组（大约 200T 的数据），植物基因组，玉米地土壤的宏基因组还是不够的。接着他本人期望使用 1973

年提出的并发 Actor 模型来实现一个轻量级的基于 Actor 模型，并开发一个新的具有更高扩展性的分布式计算引擎来提高基因组组装的并行性能。

2.3.6 HipMer

HipMer[40,41]是加州大学 Berkey 分校基于其并行化编程语言 UPC[103]而对 Meraculous 组装软件代码进行移植的一款并行基因组装软件。首先，HipMer 明显提高了复杂重复基因组的 k-mer 频谱分布的并行 k-mer 分析扩展性。接下来，HipMer 对 De Bruijn 图的遍历，通过采用一种新的通信避免的并行算法。最后，通过使用 UPC 并行编程语言的单边通讯机制对 Meraculous 的 scaffolding 模块进行并行化，同时有效减少负载不平衡。在 Cray XC30 使用大规模基因组数据（人类数据 290GB 和小麦数据 477GB）的组装结果表明，HipMer 的高效的性能和上万核心的可扩展性。总的来说，Meraculous 的能够在短短 8.4 分钟的时间里使用 15，000 核芯完成人类基因组组装。

2.3 并行组装软件性能对比

上节介绍的六个并行基因组装软件中，除了 YAGA, HipMer 没有开放源代码外，其他四个组装软件，即 SOAPdenovo, PASHA, ABySS, Ray，均已经开放源代码。同时为了容易进行性能对比，本文也加入了 Velvet 组装软件作为性能基准。由于以上五个组装软件在图的构建步骤上的优化策略不同，各自适用范围也不同，并各有优势。为综合比较这五个组装软件在性能上差异，本文选取了基因组大小不等的物种对它们进行性能测试。

这里本文选取 3 组数据来对这五个基因组装软件的性能进行对比测试。这三套数据分别是 GAGE 组装竞赛的 S.aureus, R.sphaeroides, Hg14 三个数据集[104]，其具体数据信息见下表 2.1。

表 2.1 测试的物质基因数据信息

物种	S.aureus	R.sphaeroides	Hg14
数据大小(GB)	0.684	0.906	14.2
序列长度(bp)	37,101	101	101
序列数（百万条）	4.8	4.1	62
覆盖度(倍)	90X	90X	70X
k-mer 数量（个）	31	31	31
数据源规模（MB）	2.90	4.60	88.6

这些数据的主要特点描述如下：

（1）S.aureus：含有 480 万条 Illumina 测序仪产生的序列， 这个数据包括两个文库，一个文库的序列长度是 101bp，pair-end 的 insert 长度是 180bp；另一个文库的序列长度是 37bp， pair-end 的 insert 长度是 3.5kbp 。该数据也可在 NCBI 网站 SRA 数据库中下载，下载编号是 no. SRS004751, no. SRS004752。

（2）R.sphaeroides：包含 410 万条 Illumina 测序仪产生的长度为 101bp 的序列，该序列数据也包括两个文库，两个文库的 insert 长度分别为 180bp 和 3.5kbp。该数据也可以在 NCBI 网站的 SRA 数据库中下载，下载编号为 no. SRX033397）。

（3）Hg14：人类基因组的第 14 条染色体测序数据，该数据共有 2650 万条长度为 101bp 的 Illumina 测序仪产生的序列，这些序列分布于三个文库，这三个文库的 insert 长度分别为 155bp, 2.8kbp, 35kbp。该数据在 NCBI 网站的 SRA 数据库中的下载编号为 no. SRP003680。

在本次性能测试中，本文分别记录了五个组装软件完成组装任务所需的时间，从而说明它们各自的组装效率，测试结果如图 2.4 所示。

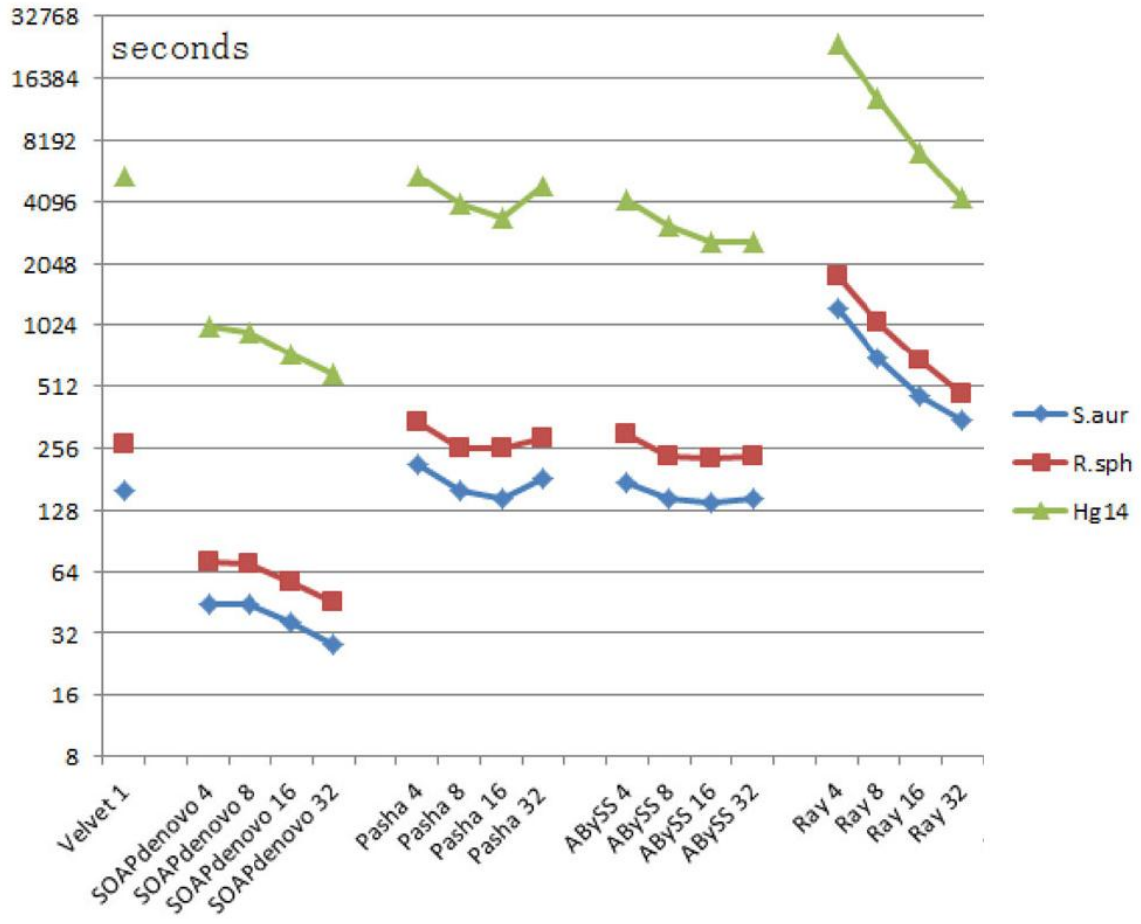

图 2.4五个组装软件共享内存服务器上的在运行时间消耗

图 2.4 为五个组装软件在单个 CPU 下完成整个组装任务时的时间对比可以看出，SOAPdenovo 的效率最高， Ray 的单核效率最低。而 Ray，PASHA，ABySS 的性能非常接近，这也说明这 3 个基因组装软件使用的策略是基本类似的。接着当计算核心的数量

增加时，各个基因组组装软件的性能开始各有变化。其中 Ray 的性能基本成线性加速，加速效果明显，而 ABySS 和 PASHA 随着计算核心从 1 成倍提升到 32 核心时，其运行时间先随着核心数的增加而减少，最后当核心数超过 16 核心后，运行时间反而增加，这说明这两个软件的扩展性能有限。 SOAPdenovo 因为只在 k-mer 分析一步使用了线程并行化而其他步骤并没有并行化，所以它虽然没有像 Ray 一样达到近线性加速，但是也随着核心数的提高，计算时间稳健的降低。

综上可知，以上这四个组装软件中都是使用了并行组装策略的软件，通过 MPI 并行编程技术，在并行计算机集群系统上建立分布式 De Bruijn 图，减少了单个计算机的内存消耗，提高了单个计算机的组装速度。但是，ABySS，PASHA 也存在不足之处，经实验测试，该拼接技术在 16 个进程中取得较好的加速效果，当扩展到 32 个进程时，并行加速效果开始退化。

对于大型基因组的拼接，单机的基因组装策略在拼接时间和内存消耗方面均存在着不足之处，并行基因组装策略虽然采用 MPI 或者 UPC 并行编程技术，但是其并行扩展性还是有待提升。因此，如何优化 De Bruijn 图结构，提高 De Bruijn 图的并行扩展性，同时降低单机的内存消耗成为当前解决大规模并行基因序列组装问题的关键所在。

2.3 本章小结

本章首先阐述了基因组序列测序技术和组装技术发展历程，同时对相应技术发展过程中所兴起的组装策略进行技术分析和对比，同时本文就基于 De Bruijn 图并行化基因组装策略进行介绍，并对现有主要的 6 个基因组并行组装软件的技术细节进行介绍和优缺点分析。在此基础之上，本文选取四种典型并行基因组组装软件，即 SOAPdenovo, PASHA, ABySS, Ray，在三个测序数据上，进行性能对比试验。性能分析结果可以看出，基于单机运行的 De Bruijn 图组装技术，内存消耗已成为当前序列组装的一个瓶颈；而并行基因组装策略虽然采用 MPI 或者 UPC 并行编程技术，但是其并行加扩展性仍然有待提升。因此，如何优化 De Bruijn 图结构，提高 De Bruijn 图的并行扩展性，同时降低单机的内存消耗成为当前解决大规模并行基因序列组装问题的关键所在。

第三章 并行基因组组装模型

为处理新一代测序技术所产生的大规模的基因数据，本文需要一种高效可扩展组装策略来应对生物信息学领域的信息爆炸。虽然高性能计算、云计算以及众核技术在序列对比，SNP 查找，表达分析等应用上展现了强大的性能提升。但在紧耦合的图算法应用领域，特别是基因组组装，高可扩展的解决方案近十年来一直是未解之难题。

现有的基因组装策略主要分为两类，其一是 Overlap-Layout-Consensus 策略，主要用于第一代测序技术产生的基因序列，其算法需要两两比对故其算法复杂度达到平方级 $O(n^2)$。 该策略由于算法复杂度太高极少应用于下一代大规模高通量基因序列的组装。另一种策略是基于 De Bruijn 图的 De Novo 组装，该算法接近线性的算法复杂度使其广泛用于下一代高通量基因序列的组装。同时一些并行基因组装软件，例如 ABySS, Ray, PASHA, YAGA, HipMer 等，也对该策略进行了并行实现。正如上一章分析所述这些组装软件的工作只是对基于 De Bruijn 图的组装进行了并行实现，然而 Pevzner PA 与 2001 年提出的基于 De Bruijn 图的组装由于其路径收缩计算时的数据依赖，并不适合直接的并行实现。为提高组装算法的并行度，深入解耦组装算法中的计算数据依赖，本文在本章提出了一种适合并行的高可扩展并行序列组装数学模型—双向多步 De Bruijn 图。

本章组织如下：首先第一节本文对基因组序列组装问题做了阐述和定义，接着分别在第二、三、四节介绍了双向多步图的顶点定义，双向边定义以及边合并运算；然后在第五节介绍双向多步图的性质及其与序列组装问题的等价性证明；最后第六节小结本章。

3.1 基因组序列组装数学定义

给定一个基因组的样本的参考序列 $w \in N^g$，其中 $N=\{A,T,C,G\}$，$g=|w|$, 以及从测序仪上产生的测序序列集合 S, $S=\{s^1, s^2, ..., s^h\}$。 基因组序列组装实际上就是使用这些序列集合 S 中的序列重新构造恢复该基因组样本参考序列 w 的处理过程。

但现实中随着二代高通量测序技术产生的海量基因数据， 下面这些问题使得基因组组装并不容易并行：

1. 基于二代测序技术产生的序列集合 S 中的序列都比较短，多数序列的长度在 30bp 到 120bp 之间。
2. 通常这些高通量测序技术测序能够产生超大规模的序列片段，而使用这些测序片段构造的图的规模也极其巨大，例如千人基因组序列片段的数量达到十亿级别，构造 De Bruijn 图过程中将产生大约 2^{47} 个图顶点，这个规模是现今 Graph 500 [105]排名所能处理的最大图的规模的 128 倍。
3. 基因组本身的 GC 含量差异，Repeat 分布和含量百分比，测序仪器带入序列本

身的测序误差和测序空隙等特异性使得序列组装问题本身更加复杂。

4. 主流的 De Bruijn 图策略等价于旅行售货员问题，求取其最有解实际上是 NP 难问题。

因此重构基因组的参考序列实际上无法在多项式时间内解决。同时在实际中存在的测序空隙，路径分支，错误序列片段，重复序列等都使得重构一个和参考序列完全一致序列几乎不可能。为此在实际中研究人员退而求其次，去尽力恢复参考序列的一个子序列集合，本文称之为 contigs。在实际计算过程中，本文通过对无分叉路径的收缩而得到这些近似的 contigs。中本文将上述基因组组装问题归纳为如下标准基因组装问题(Standard Genome Assembly)， 而本节的重心就是为该问题找到一种高可扩展的数学模型来解决传统的 De Bruijn 图策略所存在的计算依赖问题。

表 3.1 标准基因组组装问题定义

标准基因组装问题(Standard Genome Assembly)
输入： 给定一组无测序错误的测序序列集合 S, $S=\{s_1, s_2,..., s_h\}$
输出： 一个 contig 集合 C, $C=\{c_1, c_2,..., c_q\}$
要求：每一个 contig 都和测序序列集合 S 所构造的 De Bruijn 图中的一个无分支路径等价。

3.2 双向多步图的顶点定义

基因组参考序列具有 DNA 互补双螺旋结构，可沿两个方向扩展，碱基的读取方向均是从 5’端读到 3’端。根据碱基互补配对准则，基因组双链互补结构模型如图 3-1 所示，该基因序列从不同方向读有两种不同的读取形式，可以记为“TAGTCGAGGCTTAG”，也可以记为“CTAAGCCTCGACTA”。而这两种记录方式都是等价的，均可用来指代下图的基因组序列。

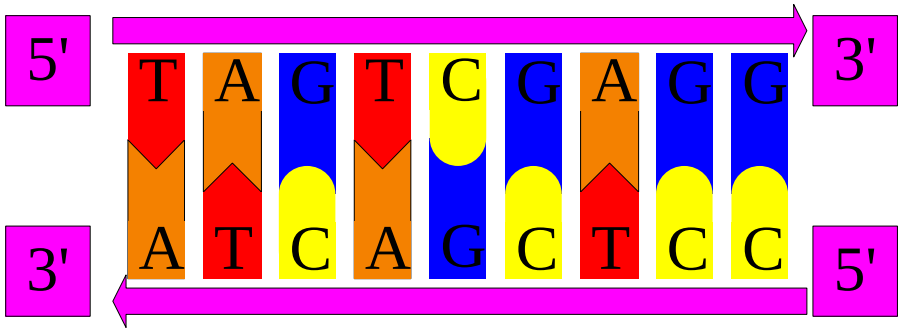

图 3.1DNA 双链互补结构

基因组序列是由四种碱基组成的序列，这四种碱基集合可被定义为 $\Sigma=\{A,C,G,T\}$。这些碱基满足互补配对原则 $M=\{A\to T, C\to G, G\to C, T\to A\}$。本文通常按照其英文字母表中的排序来人为的定义相应的碱基大小，即 $A<C<G<T$。对于任意一个长度为 l 基因序列 s，即由 l 个碱基连接而成的字符串，其中 $s\in\Sigma^l$。取 s 中

任意一个长度为 k 的子串 α，$\alpha = s[j]s[j+1]\cdots s[j+k-1]$，$0 \le j \le l-k+1$。$\alpha$ 称为 s 中的 k-mer，所有来自于 s 长度为 k 的 k-mer 集合可记录为 $\mathbb{Z}(s,k)$，其中 k 取奇数。对于任意一个 k-mer α，α' 被定义为 k-mer α 的互补 k-mer。根据碱基互补配对准则，$M = \{A \to T, C \to G, G \to C, T \to A\}$，则 $s[i] = s[k-i+1]'$。

另外对于 k-mer α，本文定义 $surf(\alpha, k-1)$ 为 k-mer α 的长度为 *k-1* 的后缀，$pre(\alpha, k-1)$ 为 k-mer α 的长度为 *k-1* 的前缀。在接下来的讨论中，本文用“$\otimes$”定义碱基的笛卡尔积运算，例如 $M = \{\{A\}\}$，$N = \{\{G\},\{T\}\}$，$M \otimes N = \{\{AG\},\{AT\},\{TG\},\{TT\}\}$。用“$\circ$”代表碱基的连接运算，例如 s_1=”abc”，s_2=”def”，那么 $s_1 \circ s_2$=”abcdef”。对于图中的任意顶点 α，*degree*(α)表示顶点 α 的出度和入度之和。

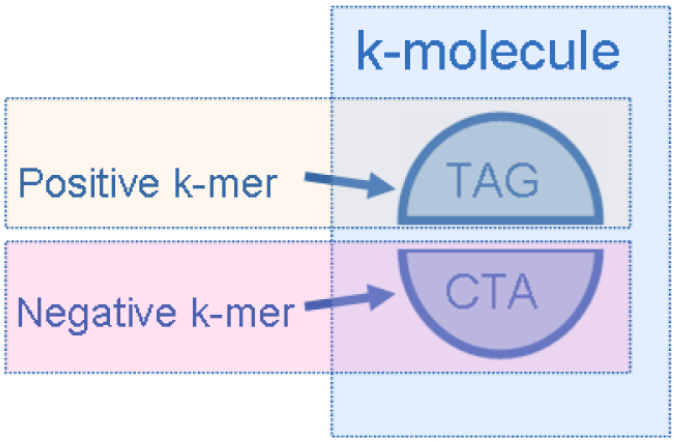

图 3.2 k-分子结构，每个 k-分子由 2 个互补的 k-mer 组成

本文中互补的长度为 k 的 k-mer 称作 k-分子，记为 $\hat{\alpha}$，其中 $\hat{\alpha} = \{\alpha, \alpha'\}$，如图 3-2。根据字符串比较规则，一对互补的 k-mer α，α'，其中较大的 k-mer 记为 α^+，较小的 k-mer 记为 α^-。用较大的 k-mer α^+ 作为代表元来标记整个 k-分子，即 $\hat{\alpha} = \alpha^+ = \{\alpha^+, \alpha^-\} = \{\alpha, \alpha'\}$。k-分子 $\hat{\alpha}$ 即为本文接下来即将定义的双向多步图中的顶点。对于一个给定的碱基序列 s，s 中所有的 k-分子记为 $\mathbb{S}(s,k)$。在双向多步图中，对于给定的一系列基因序列集合 $S = \{s_1, s_2, \ldots, s_h\}$，本文双向多步图的顶点集合被定义由 S 产生的所有长度为 k 的 k-分子集合：

$$V_S = \mathbb{S}(S,k) = \sum_{i=1}^{h} \mathbb{S}(s_i,k) \qquad \text{公式(3-1)}$$

这里本文用下表 3.2 来表示本小节中定义的数学符号，数学符号的意义，以及其典型例子。

表 3.2 相关数学符号定义

数学符号意义	数学符号	符号典型用例
序列中碱基集	Σ	$\Sigma=\{A,T,C,G\}$
基因组参考序列	w	$w="TAGTCGAGG"$
序列片段集	S	$S=\{"TAGTCG","AGTCGA","TCGAGG"\}$
互补的 k-mer	α,α'	$\alpha="TAG",\alpha'="CTA"$
正向的 k-mer	α^+	$\alpha^+="TAG"$
反向的 k-mer	α^-	$\alpha^-="CTA"$
k-分子	$\hat{\alpha}=\alpha^+=\{\alpha^+,\alpha^-\}$	$\hat{\alpha}=\{"TAG","CTA"\}$
序列 s 构造的 k-mer 集	$\mathbb{Z}(s,k)$	$Z("TAGTCG",3)=\{"TAG","AGT","GTC","TCG"\}$
序列 s 构造的 k-分子集合	$\mathbb{S}(s,k)$	$\mathbb{S}(s,k)=\{\{"TAG","CTA"\},\{"AGT","ACT"\},\{"GTC","GAC"\},\{"TCG","CGA"\}\}$

3.3 双向多步图的双向边定义

3.3.1 双向一步图的边定义

定义 1. 一个给定基因序列片段 s 生成的长度为 k 的 k-分子组成的 k 阶**双向一步 De Bruijn 图**的定义如下：

$$G_k^1(s)=\{V_s,E_s^1\} \quad \text{（公式 3-1）}$$

在接下来的章节中本文将双向一步 De Bruijn 图简称为**双向一步图**。在公式 3-1 中，由 s 构造的 **k 阶顶点（k-分子）**集合:

$$V_s=\mathbb{S}(s,k) \quad \text{（公式 3-2）}$$

同时若存在顶点 $\hat{\alpha}$ 、 $\hat{\beta}$ 在 s 中是两个相邻的 k-分子，则图中顶点 $\hat{\alpha}$ 和 $\hat{\beta}$ 之间存在有一条双向一步边相连。双向一步图中**双向一步边**的五元组集合 E_s^1 被定义为：

$$E_s^1=\{e_{\alpha\beta}^1=(\alpha,\beta,d_\alpha,d_\beta,C_{\alpha\beta}^1)\mid\forall\hat{\alpha},\hat{\beta}\in\mathbb{S}(s,k),\ (suf(\alpha,k-1)=pre(\beta,k-1))\wedge(\alpha\circ\beta[k-1])\in(\mathbb{Z}(s,k+1))\vee(\mathbb{Z}(s',k+1)))\} \quad \text{（公式 3-3）}$$

公式 3-3 中， d_α 表示 k-mer α 的方向，如果 $\alpha=\alpha^+$， 则 $d_\alpha='+'$， 否则 $d_\alpha='-'$。 $C_{\alpha\beta}^1$ 是这个双向一步边上的内容或者字符， $C_{\alpha\beta}^1$ 最开始被初始化为 $\beta[k-1]$，即 $C_{\alpha\beta}^1=\beta[k-1]$ 。那么本文也可以推理得到 $suf(\alpha\circ C_{\alpha\beta}^1,k)=\beta$。

推论 1. 双向一步图中的每一个顶点（k-分子）均是由一对 k-mer 组成的，因此图中的任意两个顶点（k-分子），实际中有四种可能的连接方式的边来连接，根据顶点（k-分子）中 k-mer 两两之间的不同连接情况，图 3-3 中连接顶点（k-分子）的边一共有如下四种类型：

1） $e^1_{\alpha^+\beta^+}=\{\alpha^+,\beta^+,+,+,C^1_{\alpha^+\beta^+}\},e^1_{\beta^-\alpha^-}=\{\beta^-,\alpha^-,-,-,C^1_{\beta^-\alpha^-}\}$ （公式 3-4）

2） $e^1_{\alpha^+\beta^-}=\{\alpha^+,\beta^-,+,-,C^1_{\alpha^+\beta^-}\},e^1_{\beta^+\alpha^-}=\{\beta^+,\alpha^-,+,-,C^1_{\beta^+\alpha^-}\}$

3） $e^1_{\alpha^-\beta^+}=\{\alpha^-,\beta^+,-,+,C^1_{\alpha^-\beta^+}\},e^1_{\beta^-\alpha^+}=\{\beta^-,\alpha^+,-,+,C^1_{\beta^-\alpha^+}\}$

4） $e^1_{\alpha^-\beta^-}=\{\alpha^-,\beta^-,-,-,C^1_{\alpha^-\beta^-}\},e^1_{\beta^+\alpha^+}=\{\beta^+,\alpha^+,+,+,C^1_{\beta^+\alpha^+}\}$

上面的四种双向一步边中，每种边都有两种表达方式，而这两种写法都是互相等价的。然而不同的是每种边的表达方式中的第一种是用于关联（或存储在）k-分子 $\hat{\alpha}$，第二种是用于关联（或存储在）k-分子 $\hat{\beta}$。在图 3.3 中本文对这四中边进行了举例阐述，例如在图 3.3 (a)中，一个正向 k-mer “TAG”指向正向 k-mer “AGT”，那么这条边可记录为 $e^1_{TAG\cdot AGT}=(TAG,AGT,+,+,T)$，在图 3.3 (b)中，一个反向 k-mer “CTC”指向正向 k-mer “TCG”，那么这条边可记录为 $e^1_{GAG\cdot TCG}=(GAG,TCG,-,+,A)$，在图 3.3 (c)中，一个正向 k-mer“CCT”指向反向 k-mer “CTC”，那么这条边可记录为 $e^1_{CCT\cdot CTC}=(CCT,GAG,+,-,C)$，在图 3.3 (d)中，一个反向 k-mer“AGT”指向正向 k-mer“TAG”，那么这条边可记录为 $e^1_{AGT\cdot GTA}=(AGT,GTA,-,-,A)$。

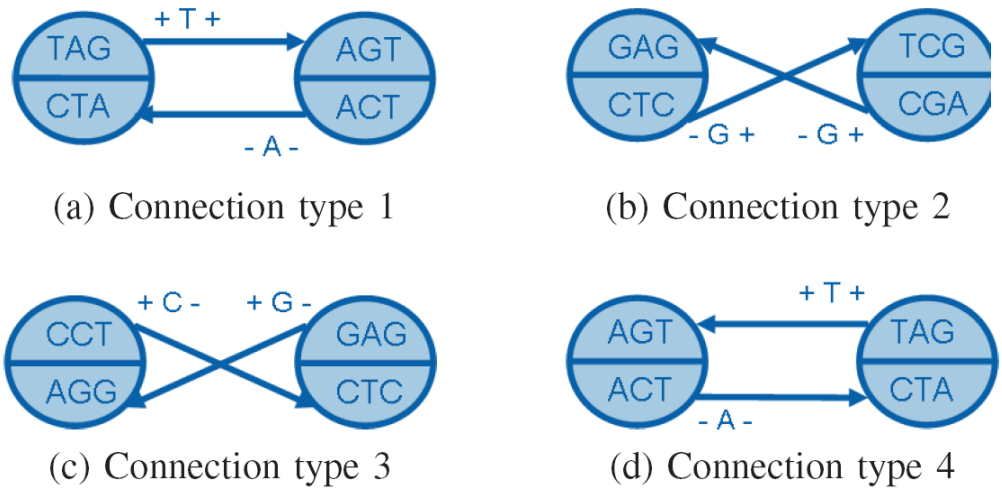

图 3.3双向一步图中连接顶点（k-分子）的双向一步边的四种类型

对于给定的 k-分子集合 $\mathbb{S}(s,k)$，这里取 $\mathbb{S}("CCGTATGGTT",3)$ 为例，来说明图中双向一步图中顶点与边的表示。图 3.4 给出了此双向一步图的顶点和双向一步边的结构。

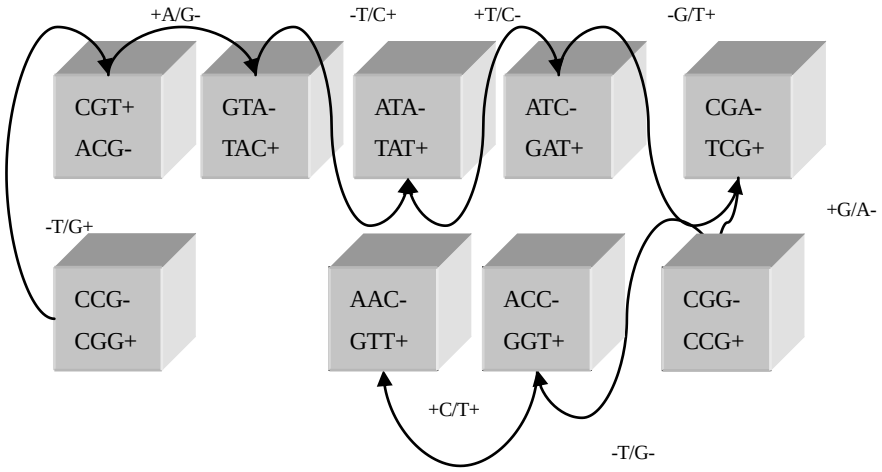

图 3.4双向一步图实例图

在上图 3.4 中，双向一步图中四个顶点，每一个立方体均表示图中的一个顶点。顶点之间分别用一实线与一虚线相连。其中，顶点之间的实线表示由 read 序列“CCGTAA”所得 k-mer 信息的连接；顶点之间的虚线表示由 read 互补序列所得 k-mer 信息的连接，由此可以确定双向 De Bruijn 图中的邻接顶点之间的连接关系。

定义 2. 综上所述，对于一个给定基因序列集合$S=\{s_1,s_2,\ldots,s_h\}$，由这个序列集合中的序列生成 k-分子集合$\mathbb{S}(S,k)$，以及相应的双向一步边集合E_s^1，则本文可以得到一个双向一步图，并抽象表示如下：

$$G_k^1(S)=\{V_s,E_s^1\}=\{\bigcup_{1\le i\le h}V_{s_i},\bigcup_{1\le i\le h}E_{s_i}^1\}$$ （公式 3-5）

对于给定基因组参考序列“TAGTCGAGG”，本文随机抽取 3 个基因序列片段“TAGTCG”，“AGTCGA”和“TCGAGG”。 当 k 取值为 3 时，本文可以构造如下图 3-5 所示的一个双向一步图。

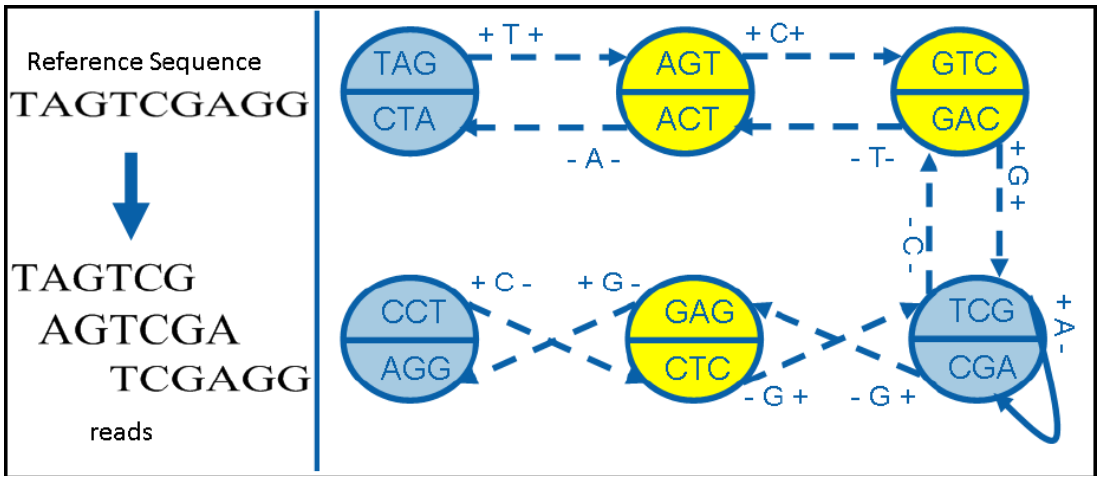

图 3.5双向多步图实例

3.3.2 双向多步图的边定义及边合并运算

定义 3. 给定两条双向一步边 $e^1_{\alpha\beta}=\{\alpha,\beta,d_\alpha,d_\beta,C^1_{\alpha\beta}\}$ 和 $e^1_{\beta\gamma}=\{\beta,\gamma,d_\beta,d_\gamma,C^1_{\beta\gamma}\}$， 如果 $e^1_{\alpha\beta}\cdot d_\beta=e^1_{\beta\gamma}\cdot d_\beta$ 并且 degree($\hat{\beta}$)=2, 那么这两条双向一步边可以合并得到一个双向二步边 $e^2_{\alpha\gamma}=\{\alpha,\beta,d_\alpha,d_\gamma,C^2_{\alpha\gamma}\}$，这里 $C^2_{\alpha\gamma}=C^1_{\alpha\beta}\circ C^1_{\beta\gamma}$。本文用符号⊗是表示一个顶点上的两条方向相同的双向边的**边合并操作**：

$$e^1_{\alpha\beta}\circ e^1_{\beta\gamma}=e^2_{\alpha\gamma} \quad \text{（公式 3-6）}$$

或

$$e^1_{\gamma\beta}\circ e^1_{\beta\alpha}=e^2_{\gamma\alpha} \quad \text{（公式 3-7）}$$

其中等式 3-6 和等式 3-7 是等价的，他们只不过从不同的合并方向来表示的是一个边合并操作。所以对边 $e^1_{\alpha\beta}$ 和 $e^1_{\beta\gamma}$ 进行边合并等价于对边 $e^1_{\gamma\beta}$ 和 $e^1_{\beta\alpha}$ 进行边合并操作。

为构造完整的代数系统，本文需要定义一个零边 0 。这个零边 0 表示的是所有不存在的双向边，例如 $0\otimes e^x_{\alpha\beta}=0$， $e^x_{\alpha\beta}\otimes 0=0$。这里两个可合并的双向多步边可以按如下公式合并：

$$e^z_{\alpha\gamma}=\begin{cases}e^x_{\alpha\beta}\otimes e^y_{\beta\gamma}, & if\ \exists\beta, e^x_{\alpha\beta}\neq\mathbf{0}, e^y_{\beta\gamma}\neq\mathbf{0}, z=x+y, e^x_{\alpha\beta}.d_\beta=e^y_{\beta\gamma}.d_\beta, degree(\hat{\beta})=2\\ \mathbf{0}, & \text{otherwise}\end{cases}$$

（公式 3-8）

定义 4. 给定一个基因序列集合 $S=\{s_1,s_2,\ldots,s_h\}$， 由这个基因序列集合构造的双向多步图可记为：

$$G_k(S)=\{V_s,E_S\}=\{\bigcup_{1\le i\le h}V_{s_i},\bigcup_{1\le i\le g}\bigcup_{1\le j\le h}E^j_{s_i}\} \quad \text{（公式 3-8）}$$

这里 g 是参考序列 w 的长度，双向多步边的集合是

$$E^j_{s_i}=\{e^j_{\hat{\alpha}\hat{\beta}}\mid\forall\hat{\alpha},\hat{\beta}\in\mathbb{S}(s_i,k)\} \quad \text{（公式 3-9）}$$

定义 5. 对于一个给定的 *x* 步-双向边 $e^x_{\alpha\beta}=(\alpha,\beta,d_\alpha,d_\beta,C^x_{\alpha\beta})$，如果存在 $e^y_{\gamma\alpha}$ 或者 $e^z_{\beta\gamma}$，满足 $e^y_{\gamma\alpha}\oplus e^x_{\alpha\beta}\neq 0$ 或者 $e^x_{\alpha\beta}\oplus e^z_{\beta\gamma}\neq 0$，那么 $e^x_{\alpha\beta}$ 能与 $e^y_{\gamma\alpha}$ 或者 $e^y_{\gamma\alpha}$ 融合，本文称 $e^x_{\alpha\beta}$ 是半扩展双向边，α 或 β 是半扩展顶点；否则 $e^x_{\alpha\beta}$ 是全扩展双向边，α 或者 β 是全扩展顶点。

在图 3-5 中，本文把半扩展顶点标注为黄色，全扩展顶点标注为蓝色。半扩展边标注为断点线，而全扩展边标注为实现。

3.4 双向多步图的性质和等价问题证明

双向多步图的性质 1：在双向多步图 $G_k(S)=\{V_S,E_S\}$ 中，E_S 里面的所有全扩展边的集合记为 E^*_S， E^*_S 里面每条边上的字符串标注集合可以表示为：

$$L^*_S=\{c^x_{\alpha\beta}\mid c^x_{\alpha\beta}=(\alpha,\beta,d_\alpha,d_\beta,c^x_{\alpha\beta}),e^x_{\alpha\beta}\in E^*_S\} \quad \text{（公式 3-9）}$$

那么如果C^*是序列组装后的 contigs 集合，即$L_S^* = C^*$。

证明：

1)对于任意 ATCG 组成的字符串$a \in C^*$，a是一个 contig，根据 contig 的定义，contig a对应双向一步图$G_k^1(S) = \{V_S, E_S^1\}$中的一个无分叉路径 P$_a$，那么令$P_a = t_1 t_2 ... t_m$，其中$t_i \in V_S$，$t_1, t_m$是全扩展顶点（分叉节点），其它中间的顶点为半扩展顶点（非分叉节点）。那么必然有$P_q = e^1_{t_1 t_2} \oplus e^1_{t_2 t_3} \oplus e^1_{t_3 t_4} \oplus ... \oplus e^1_{t_{m-1} t_m} = e^{m-1}_{t_1 t_m}$，而由于$t_1, t_m$是全扩展顶点（分叉节点），那么$e^{m-1}_{t_1 t_m}$是全扩展边。由于 contig a 对应于连接路径 Pa 上的每条双向一步边的字符标注，即$a = e^{m-1}_{t_1 t_m} \in L_S^*$。所以$C^* \subset L_S^*$。

2)对于任意元素$e^{m-1}_{t_1 t_m} = \{t_1, t_m, d_{t_1}, d_{t_m}, c^{m-1}_{t_1 t_m}\} \in E_S^*$，这里$c^{m-1}_{t_1 t_m} \in L_S^*$。那么全扩展边$e^m_{t_1 t_m}$对应于双向一步图中的一条路径 P$_a$，这里$P_a = e^m_{t_1 t_m} = e^1_{t_1 t_2} \oplus e^1_{t_2 t_3} \oplus e^1_{t_3 t_4} \oplus ... \oplus e^1_{t_{m-1} t_m}$，其中$t_1, t_m$是全扩展顶点（分叉节点），而$t_2, ..., t_{m-1}$是半扩展顶点（非分叉节点）。那么该边对应的路径$P = t_1 t_2 ... t_m$是一条无分叉路径，而这条路径上的字符串标注即对应一条 contig。那么根据标准基因组装的定义本文可得$c^{m-1}_{t_1 t_m} \in C^*$。所以$L_S^* \subset C^*$。

综合 1)、2）可证明 $L_S^* = C^*$，性质 1 得证。

双向多步图的性质 2：双向多步边的融合操作在双向边的集合$E_S \cup 0$上满足结合律，即代数系统$< E_S, \oplus >$是一个半群。

证明：对于代数系统$< E_S, \oplus >$，根据“$\oplus$”的定义可知，“$\oplus$”二元运算符是封闭的。取E_S集合中的任意三条边e^x_{ab}，e^y_{cd}，e^z_{ef}，可分两种情况进行证明：

1）若$e^x_{ab} \oplus e^y_{cd} \oplus e^z_{ef} \neq 0$，那么$(e^x_{ab} \oplus e^y_{cd}) \oplus e^z_{ef} = e^x_{ab} \oplus (e^y_{cd} \oplus e^z_{ef}) = e^{x+y+z}_{af}$；

2）若$e^x_{ab} \oplus e^y_{cd} \oplus e^z_{ef} = 0$，那么$(e^x_{ab} \oplus e^y_{cd}) \oplus e^z_{ef} = e^x_{ab} \oplus (e^y_{cd} \oplus e^z_{ef}) = 0$；

为了简化计算，先对双向多步边的合并操作做规范定义。对于双向多步图中的两条双向一步边$e^1_{\alpha\beta} = (\alpha, \beta, d_\alpha, d_\beta, C^1_{\alpha\beta})$，$e^1_{\beta\gamma} = (\beta, \gamma, d_\beta, d_\gamma, C^1_{\beta\gamma})$，如果满足$e^1_{\alpha\beta} d_\beta = e^1_{\beta\gamma} d_\beta$，并且$\deg ree(\beta) = 2$，那么则可以对$e^1_{\alpha\beta}$，$e^1_{\beta\gamma}$进行合并，并得到一条双向二步边记为$e^2_{\alpha\gamma} = (\alpha, \gamma, d_\alpha, d_\gamma, C^2_{\alpha\gamma})$，$C^2_{\alpha\gamma} = C^1_{\alpha\beta} \otimes C^1_{\beta\gamma}$。则邻边融合的操作可以表述为：$e^1_{\alpha\beta} \oplus e^1_{\beta\gamma} = e^2_{\alpha\gamma}$或者$e^1_{\beta\gamma} \oplus e^1_{\beta\alpha} = e^2_{\gamma\alpha}$，二者的表述等价。性质 2 证实双向多步边的融合操作运算是满足结合律的，那么代数系统$< E_S, \oplus >$是半群。根据性质 2，在双向多步图中，由于双向边融合满足结合律，那么任意两个邻边的合并顺序的先后，不会改变最终结果。同时双向多步图中的双向边合并操作没有时序关系，那么该操作本身是可以并行实现的。

双向一步图的一个关键性质是每个基因序列片段 s（无测序错误）都唯一对应于双向一步图上的一个路径，这个路径是以 s 的第一个 k 分子开头，以 s 的最后一个 k 分子结束。同样每一个染色体也对应于双向一步图上的路径。然而由于测序的空隙(gap)，测序误差，以及基因组的重复序列(repeats)等造成染色体断开为若干个子片段，或 contig。在双向多步图中，每一个 contig 都对应于一个全扩展边上的标注，该性质已被性质 1 所证明。另外性质 2 表明双向多步边，以及双向多步边的合并操作，满足结合律，并形成了一个半群代数系统。该性质实际上证明了标准基因组装问题可以由半群代数系统上的

边合并操作完成。而且只要本文将双向多步图中所有的半扩展边合并为全扩展边，那么最终本文得到的全扩展边上的标注字符串就是本文需要 contig。

最后并且更重要的是由于双向多步图的数学模型满足结合律，所以图中无共同顶点(k 分子)的半扩展边的合并是可以独立合并，而且乱序执行的。双向多步图的这个数学模型已经将边合并操作的计算依赖解耦，所以该数学模型因此更适合于并行基因组组装。

本章用双向多步图的数学模型来抽象基因组组装问题，并将基因组装问题等价的转化为一个半群系统上的一个可并发乱序执行的边合并操作。该数学模型的主题思想为：对基因序列的集合 $S=\{s_1,s_2,\ldots,s_h\}$ 进行切割，得到双向多步图的顶点(k 分子)集合 $V_S=\mathbb{S}(S,k)$，根据顶点(k 分子)之间的关联信息，得到双向边集合 $E_s^{\,1}$；由此本文可以构建出双向一步图的结构 $G_k^1(S)=\{V_s,E_s^1\}=\{\bigcup_{1\le i\le h}V_{s_i},\bigcup_{1\le i\le h}E_{s_i}^1\}$，其中每一条序列片段对应双向双向一步图中的一条路径；根据边合并条件，对图中所有半扩展顶点进行遍历，当满足双向边合并条件时，则将对应的双向边进行合并并删除相应的半扩展顶点，最终得到所有的全扩展双向多步边集合 $E_S^{\,*}$，而这些全扩展双向多步边上的标注序列集 $L_S^{\,*}$ 得到最后的 contig。其具体实现步骤可描述如下：

1）读入基因序列集合 $S=\{s_1,s_2,\ldots,s_h\}$ 进行处理；

2）对每条基因序列进行分割，得到双向多步图的顶点集合 $V_s=\mathbb{S}(s,k)$；

3）根据每条基因序列中连续顶点的相邻信息，提取双向多步图的双向一步边集合 $E_{s_i}^j=\{e_{\widehat{\alpha}\widehat{\beta}}^j\mid\forall\widehat{\alpha},\widehat{\beta}\in\mathbb{S}(s_i,k)\}$；

4）对任一半扩展顶点 $\widehat{\beta}$，若存在两条双向半扩展边满足 $e_{\alpha\beta}^x\neq 0,e_{\beta\gamma}^y\neq 0$，$e_{\alpha\beta}^x\cdot d_\beta=e_{\beta\gamma}^y\cdot d_\beta$，$\deg ree(\beta)=2$，$z=x+y$，则对顶点 $\widehat{\beta}$ 关联的这两条半扩展边 $e_{\alpha\beta}^x$ 和 $e_{\beta\gamma}^y$ 执行边合并操作，即 $e_{\alpha\gamma}^z=e_{\alpha\beta}^x\oplus e_{\beta\gamma}^y$，否则，不进行处理；

5）执行步骤 4，直至图中不存在半扩展顶点而不能进行边合并操作。最终得到全扩展双向边集合 $E_S^{\,*}$，所有全扩展双向多步边上的标注序列集 $L_S^{\,*}$ 即为 contig 集合 C^*。

3.5 本章小结

本章结合基因组组装问题的传统基于 De Bruijn 图的策略，提出了用改进的双向多步图的数学模型来抽象基因组组装问题，并将基因组装问题等价的转化为一个半群系统上的一个可并发乱序执行的边合并操作。该数学模型的主题思想为：对基因序列的集合 $S=\{s_1,s_2,\ldots,s_h\}$ 进行切割，得到双向多步图的顶点(k 分子)集合 $V_S=\mathbb{S}(S,k)$，根据顶点(k 分子)之间的关联信息，得到双向边集合 $E_s^{\,1}$；由此本文可以构建出双向一步图的结构 $G_k^1(S)=\{V_s,E_s^1\}=\{\bigcup_{1\le i\le h}V_{s_i},\bigcup_{1\le i\le h}E_{s_i}^1\}$，其中每一条序列片段对应双向双向一步图中的一条路径；根据边合并条件，对图中所有半扩展顶点进行遍历，当满足双向边合并条件时，则将对应的双向边进行合并并删除相应的半扩展顶点，最终得到所有的全扩展双向多步边集合 $E_S^{\,*}$，而这些全扩展双向多步边上的标注序列集 $L_S^{\,*}$ 即为最后的 contig。

双向多步图的两个性质，直接证实上述数学模型的与原始问题的等价性和正确性。其次更重要的是由于双向多步图的数学模型满足结合律，所以图中无共同顶点(k 分子)的半扩展边的合并是可以独立合并，而且乱序执行的。双向多步图的这个数学模型已经将边合并操作的计算依赖解耦，所以该数学模型相比于基于 De Bruijn 图的策略，更适合用于并行可扩展的基因组组装。

第四章 SWAP 异步并行计算模型与并行基因序列组装

为了高效处理双向多步图构造的半群系统上的边合并操作，本章提出了异步并行计算模型 SWAP，该模型可以最大化的提高半群系统上边合并操作的并行度。接着本文阐述该异步并行计算模型的具体实现算法，并对其计算通讯复杂度抽象和分析，最后本文从理论上证明该异步计算模型具有高扩展性。基于该计算模型本文描述了并行序列组装软件 SWAP-Assembler 的每一步实现算法，同时最后通过理论证明该软件总的计算复杂度达到 $O(g/p)$，而整个通讯过程的通讯次数为 $O(g/p)$，进程间通讯量为 $O(g\log(g)/p)$，这里 g 是基因组参考序列的长度，p 是参与计算的 CPU 核数。

4.1 SWAP 异步并行计算模型

本小节首先给出 SWAP 异步并行计算模型所解决的问题，接着阐述其编程接口和实现算法的细节，最后本文给出该计算模型的算法复杂度分析，并讨论该计算模型和其他计算模型的优缺点。

4.1.1 异步并行计算模型的问题定义

定义 4.1 集合 A 及其关联的可结合操作 R: $A\times A\to A$ 构成的一个半群 SG(A,R)，$R(a_i,a_j)$ 可以被用来表达一个在操作数 a_i 和 a_j 上满足结合律的操作，其中 $a_i,a_j\in A$。这里操作数 a_i，a_j 以及操作 $R(a_i,a_j)$ 被定义为操作 $R(a_i,a_j)$ 的小世界，并用 $[a_i,a_j]$ 来表示。即 $[a_i,a_j]=\{R(a_i,a_j),a_i,a_j\}$。

在实际应用中，半群中的操作对应于一个给定算法的基本操作单元。例如，在基于双向多步图的基因组装中，基本操作单元是边合并操作；对于拓扑排序，对一对顶点的排序操作也可以被定义为基本操作单元。

定义 4.2 操作集合 κ 包括了所有在 SG(A,R)中的操作，即

$$\kappa=\{R(a_i,a_j)\mid a_i,a_j\in A\} \qquad \text{（公式 4-1）}$$

所有在半群 SG(A,R)中的基本操作被定义为**活动 ACT(A, σ)，这里操作集 σ 是** κ 的一个子集。

对于任意两个小世界 $[a_i,a_j]$，$[b_i,b_j]$，只要满足 $a_i\neq b_i$，$a_i\neq b_j$，$a_j\neq b_i$，$a_j\neq b_j$，那么对应的操作 $R(a_i,a_j)$ 和 $R(b_i,b_j)$ 就可以被独立计算且互不干扰。由于半圈的结合律的性质，使得只要这些操作（可以乱序执行）都被完成结果就都是一样的。因此半群

SG(A,R)所衍生的活动在计算过程中实际上是具有潜在的并行性的。为此本文设计了一个异步并行计算模型 SWAP 用于这种抽象数学问题的计算。 SWAP 计算模型的基本工作机制是**加锁-计算-解锁**。 对于任意一个 σ 中的操作 R(a,b)，这 3 个步骤的具体工作机制如下：

1). 加锁 对 R(a,b)的小世界[a,b]进行加锁；
2). 计算 对操作 R(a,b)进行计算，并对 a,b 的值进行更新；
3). 解锁 对 R(a,b)的小世界[a,b]的小世界进行解锁。

4.1.2 异步并行计算模型的实现

在活动 ACT(A, σ)中， σ 中的操作分布于网络上的多个处理器上， 每个处理器上的进程通过消息获取相关的操作数，例如 a,b，并计算 R(a,b)。这里每个进程还要同时与其他进程协作以发送或者更新本地变量的数值。为此本文为每个进程启动了 2 个线程，一个线程叫做 SWAP 线程，主要负责 SWAP 三个步骤中的第二个步骤，即操作的计算；而另一个线程叫做服务线程，主要负责侦听，回复，并执行其他远程 SWAP 线程的命令。

活动 ACT(A, σ)中的操作数集合 A 和操作集合 σ 可以被当作一个图 G(σ,A),其中 σ 是顶点，A 是边。本文使用邻接链表来分布式的存储 G(σ,A)。本文中一个卷积哈希函数 hashFun(x)被用于对操作集合 σ 在 p 个处理器上进行哈希划分和存储。 这里 σ_i 和 A_i 是存储于处理器 p_i 上的操作集合和操作数集合。 其中 $\sigma=\bigcup_{i=0}^{p-1}\sigma_i$ ， $\bigcap_{i=0}^{p-1}\sigma_i=0$, A_i 是图 G(σ,A) 中和 σ_i 有关联的边的集合， $ACT(A,\sigma)=\bigcup_{i=0}^{p-1}ACT_i(A_i,\sigma_i)$ 。在图 4.1 中阐述了进程之间操作数集合和操作集合的存储。 在图 4.2 和图 4.3 中也列出了 SWAP 线程和服务线程的伪代码。

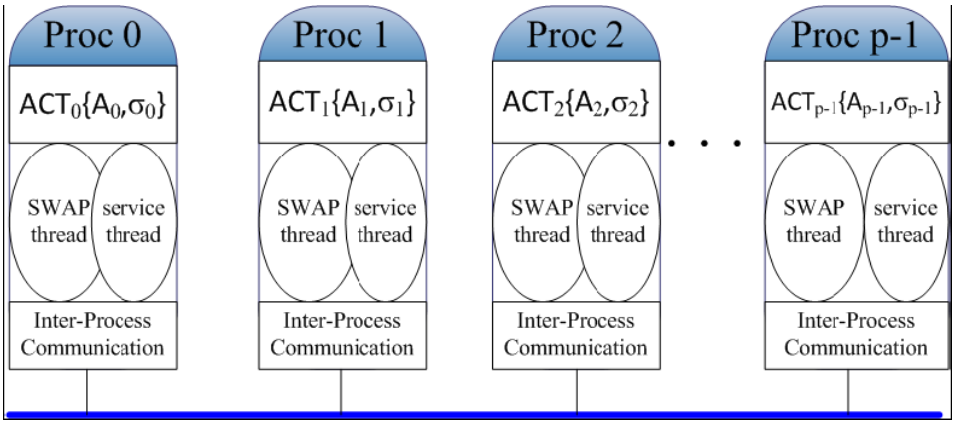

图 4.1 SWAP 异步并行计算模型在集群上的数据存储和实现框架

图 4.2 中算法 1 描述的是 SWAP 线程执行 SWAP 异步并行计算模型三步骤中的计算，而图 4.3 中算法 2 中描述的是远程线程可以与本地 SWAP 线程协作来完成 SWAP 异步并

行计算模型三步骤中的加锁和解锁两个步骤。算法 1 和算法 2 中伪代码调用的一些用户自定义函数的定义将表 4.1。

图 4.3 中算法 2 的实现借鉴了类似于 802.11 协议[106,107]中 CSMA/CA[108]机制中的随机退避算法来对加锁冲突进行自动退避。这里在不失一般性的情况下，本文在 SWAP 中使用二进制随机退避算法。值得注意的是，这里每个进程中所有冲突的操作将共享且仅共享一次二进制退避。所以这里只要 σ_i 足够大，退避算法所带来的性能损失可以忽略不计。

活动 ACT(A, σ)中互相不干扰的边合并操作都可以被并行执行，SWAP 异步并行计算模型利用这个小世界同步和全局异步的机制最大化的挖掘了半群中可结合操作在计算中的潜在并行度。

Algorithm 1: Pseudocode of SWAP thread

```
Input: The activity ACT_i(A_i, σ_i).
Output: A_i*
begin
    while σ_i ≠ Φ do
        for R(a,b) in σ_i do
            if trylock(a) ≠ Locksuccess or trylock(b) ≠
            Locksuccess then
                unlock(a);
                unlock(b);
                continue;
            end
            [a,b] ← GetSmallWorld((a,b));
            Replya ← Msg_Lock(a, proc(a));
            Replyb ← Msg_Lock(b, proc(b));
            if Replya ≠ Locksuccess or Replyb ≠ Locksucess
            then
                if Replya = Locksuccess then
                    Msg_Unlock(a, proc(a)) ;
                end
                if Replyb = Locksuccess then
                    Msg_Unlock(b, proc(b)) ;
                end
                unlock(a);
                unlock(b);
                continue;
            end
            Recva ← Msg_Read(a, proc(a));
            Recvb ← Msg_Read(b, proc(b);
            (Recva, Recvb) ← Operation(Recva, Recvb);
            Msg_Write(a, Recva, proc(a));
            Msg_Write(b, Recvb, proc(b));
            Msg_Unlock(a, proc(a));
            Msg_Unlock(b, proc(b));
            σ_i ← σ_i - R(a,b);
        end
        Wait for a random backoff time;
    end
end
```

图 4.2 SWAP 异步并行计算模型中 SWAP 线程的伪代码

Algorithm 2: Pseudocode of Service thread

```
Input: Subset A_i
begin
    while true do
        Receive a message Msg to a from Operation R(a,b) at
        ProcID;
        if Msg = Msg_Lock then
            if trylock (a) = success then
                Msg_Locksuccess(a, R(a,b), ProcID);
                continue;
            end
            Msg_Lockfailed(a, R(a,b), ProcID);
            continue;
        end
        if Msg = Msg_Unlock then
            unlock(a) ;
            continue;
        end
        if Msg = Msg_Read then
            Msg_ReadBack(a, R(a,b), ProcID) ;
            continue;
        end
        if Msg = Msg_Write then
            x ← y;
            continue;
        end
        if Msg = Msg_End* then
            break;
        end
    end
end
* When all SWAP threads finish, process 0 will broadcast an
End message to stop all Service threads.
```

图 4.2 SWAP 异步并行计算模型中服务线程的伪代码

表 4.1 SWAP 线程和服务线程所调用的内部函数和用户自定义函数说明

Class	Function Name	Function Description
Message Functions	Msg_Lock(a, p)	Lock a in process p
	Msg_Unlock(a, p)	Unlock a in process p
	Msg_Read(a, p)	Fetch associated values of a in process p
	Msg_Write(a, *newa*, p)	Update associated values of a with *newa* in process p
	Msg_Locksuccess(a, $R(a,b)$, p)	Send Locksuccess Message back to $R(a,b)$ in process p
	Msg_Lockfailed(a, $R(a,b)$, p)	Send Lockfailed Message back to $R(a,b)$ in process p
	Msg_ReadBack(a, $R(a,b)$, p)	Send associated value of a back to $R(a,b)$ in process p
	Msg_End()	Command to stop the service thread
Internal Functions	proc(a)	Get process ID of a
	trylock(a)	Lock a
	unlock(a)	Unlock a
User-defined Functions	GetSmallWorld($R(a,b)$)	Get small world $[a,b]$ from operation $R(a,b)$
	Operation(a,b)	Compute the operation $R(a,b)$

4.1.3 异步并行计算模型的复杂度分析

本小节对 SWAP 计算模型的复杂度分析在其运行的集群上有以下两个假设：

1). 每个计算节点，都有一个处理单元和本地内存，

2). 任意两个计算节点之间都可以点对点的通讯。

这个集群的主要的参数特征抽象如下：

L: 任意两个计算节点之间点到点通讯一个字节所带来的通讯延迟的上界，

S: 通讯启动延迟，

p: 计算节点(CPU)的个数。

接着本文所要计算的活动 ACT(A, σ)做如下假设：

1). σ 中的操作的个数为 m, 这些操作随机的存储在 p 个计算单元，且 $\sigma_i \approx m/p$，

2). A 中的操作数的个数为 e, σ_i 关联的操作数 A_i 都存在第 i 个处理单元，$A_i \approx e/p$，

3). 每发送一个消息都会带有 L 比特报头，并需要 S 秒来启动消息发送。在图 4.2 和图 4.3 的算法 1 和算法 2 中，消息可以被分为 2 类，一类是指令消息，一类是数据消息。指令消息包括 Lock，Unlock，ACK（Lock 消息的回复），数据消息包括 Read，Write，Reply（读消息的回复）。总的数据消息的总数据通讯量记为 F. 指令消息都是固定长度，所以这里本文分析复杂度的时候忽略指令消息，

4). 每个操作的计算时间是 H，H 代表的是用户自定义的操作函数 Operation(a,b)的算法复杂度，

5). 如果把 σ 作为点集，A 作为边集， 那么本文假设图 G(σ,A)的直径是 d_{max}。

根据假设 1 和 2，SWAP 计算模型每个处理单元的空间复杂度是 O((m+e)/p) 。处理单元数 p 绝大多数情况是远小于 m 的，因此 SWAP 的运行时间主要是其计算时间和通讯时间：

RunTime = CompTime + CommTime （公式 4-2）

在 σ_i 中每个操作的计算都需要 r 次指令或者数据通讯，这里 r 的实际值在图 4.2 和图 4.3 的算法 1 和算法 2 中介于 4 到 12 之间。因此每个处理单元的通讯启动时间为：

StartTime = rmS/p （公式 4-3）

为了简化分析，本文假设总的通讯数据量 F 被 p 个处理单元均匀承担。那么消息通许传输时间为

TranTime = FL/p （公式 4-4）

合并公式 4-3 和公式 4-4，每个处理单元的通讯时间为

CommTime =StartTIme+TranTime=(rmS+FL)/p （公式 4-5）

根据假设 1 和 4，每个处理单元有大约 m/p 个操作，而且每个操作耗时 H 那么每个处理单元的计算时间为

CompTime = mH/p （公式 4-6）

最后根据公式 4-5 和公式 4-6，SWAP 计算模型的运行时间为

RunTime =(rmS+FL+mH)/p （公式 4-7）

当处理单元的数量接近于甚至大于 m 的时候， Runtime 将不能用公式 4-7 来确定。对于多数应用而言，σ 中的操作无法一次全部运行完，因为 SWAP 的加锁机制使得互相干扰的两个操作（在双向多步图中有共同节点相邻的双向边）不能同时计算。在图 4-4

中给出了一组操作 $a_1 \oplus a_2 \oplus \cdots \oplus a_z$,每个操作 $a_i \oplus a_{i+1}$ 需要对$[a_i, a_{i+1}]$加锁，所以一轮通讯最多只有一半的操作可以被执行。每一轮通讯操作都会减半，对这 z 个操作总的通讯轮数大约是 $b\log(z)$, 这里 b 是一个常数。

根据假设 5，图 G(σ, A)中最长的一条路径是图的直径，即 d_{max}，那么对 ACT(A, σ)计算的通讯轮数可按如下公式计算

CommRound=$b\log(d_{max})$ （公式 4-8）

根据公式 4-8，这个通讯轮数实际上和图 G(σ, A)的结构有关。然而在大多数情况下，处理单元数 p 要远小于操作数 m，所以大多数情况使用公式 4-7 来分析 SWAP 的运行时间。

由于在共享内存的计算机上计算 ACT(A, σ) 的计算复杂度是 O(mH)， 那么本文用有 p 个处理单元的集群来计算 ACT(A, σ)时的加速比为

Speedup = pmH/(rmS+FL+mH) （公式 4-9）

公式(4-7)和公式(4-9)可以用来分析 SWAP 异步并行计算模型在计算半群上衍生的 ACT(A, σ)时的计算复杂度和加速比。公式中的参数 F, H, m 是有具体问题本身和用户自定义函数共同确定的。

图 4.4 活动 ACT(A, ⊕) 的计算样例。其中 $A=\{a_1, a_2, \cdots, a_8\}$，$ACT(A,\oplus)=\{(a_1,a_2),(a_2,a_3),\cdots,(a_7,a_8)\}$。在每一轮中最有有一半的操作会被计算。

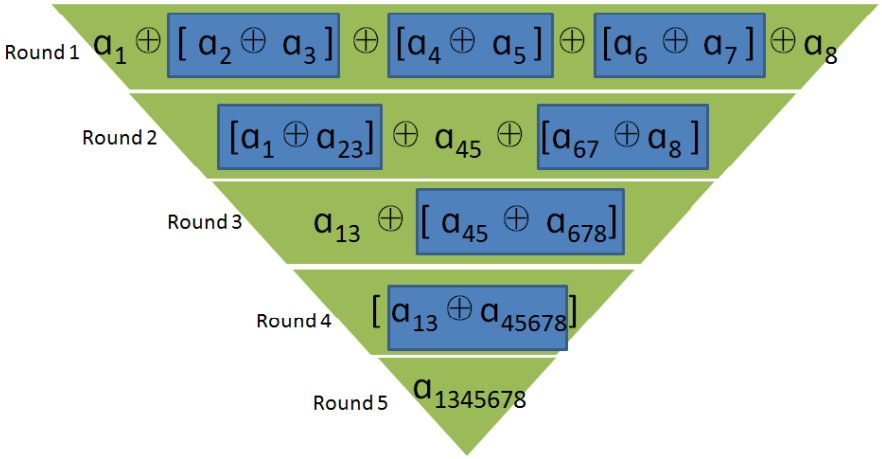

图 4.4 计算样例

4.1.4 异步并行计算模型与其他计算模型的比较

早期就有基于大同步计算模型(BSP)设计一个探索性的子集同步语言模型，但相关研究者认为实现在任意时间将多个并发集共享在一个大同步里几乎是不可能的。然而在自然世界中，这样的并发集是广泛存在的，而强制性的使用一个共享的大同步是没有必要的，在资源利用上也是不合理的。在本节中，所提出的所有并发集都在本集合中实现同步，而在全局上异步推进的，这里没有并发集的全局大同步的负担，所以本节所提出的 SWAP 计算模型更适合这种并发子集的相关操作。

在半群系统中，如果小世界[*a*,*b*], [*c*,*d*]没有任何关联，那么相应的计算操作(*a*,*b*), (*c*,*d*)将自动成为两个并发子集。SWAP 计算模型设计的原意就是为了能够将所有可抽象为半群的问题进行并行化。 SWAP 在实现中引入了加锁-计算-解锁的计算机制，并利用并发子集内的同步和全集异步的计算模型应用于半群上的操作计算，以最大化相应问题的并行度。

对于大多数的图算法问题都可以被抽象为半群系统，那么 SWAP 异步计算模型实际上可以用来计算超大规模图（例如本论文所讲述的基因组装问题的双向多步图）。BSP[109, 110]模型是一个全局同步的计算模型， 并已经用于大图计算。然而，由于其本身的大同步机制使得所有无关的并发子集都必须共享一个大同步，这既影响系统效率，也影响了其模型的表达力，BSP 模型所能表达的图算法多局限在迭代性图算法。LogP 模型[111]是一个更加灵活的异步计算模型，它本身能够用于实现非常复杂图算法，但这需要丰富的开发经验和性能优化的技巧。 Mapreduce[112, 113]计算模型是由 BSP 模型衍生的一种云计算数据分析模型，该计算模型比较适合易并行统计型的作业，但对于复杂的图算法其表达能力也有限。 对于像半群计算这种只满足结合律的操作集合，已经超出了 Mapreduce 的表达范围。SWAP 在设计上权衡了系统效率和开发难度，并达到子集同步全局异步的要求，使得 SWAP 模型更适合半群（并发子集）的计算。在 SWAP 模型中，用户只需要对表 4.1 操作函数 Operation 进行重定义就可以实现相应的算法。 而公式 4-7 和公式 4-9 可以用来分析该算法的运行时间和加速比。

4.2 基于异步计算模型的并行组装实现

这里本文基于上一章的双向多步图和前面设计的 SWAP 异步并行计算模型，实现了并行序列组装软件 SWAP-Assembler。 该软件由 5 个模块组成，包括并行 I/O 存储，双向一步图构建，图过滤，边合并，contig 扩展，并行序列组装软件 SWAP-Assembler 的软件结构图见图 4-5. 在最后对该软件的算法复杂度进行分析。

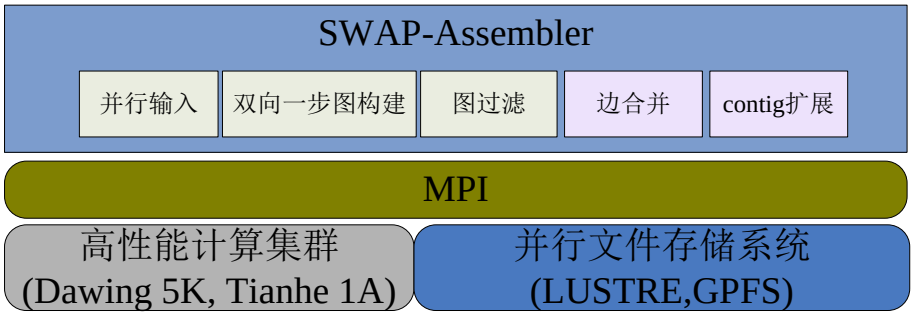

图 4.5 序列拼接系统结构图

4.2.1 并行 I/O

由于下一代基因测序技术产生的海量基因数据达到 TB 深圳 PB 级别， 将这些数据载入到内存中需要耗时若干个小时，因此一个高效并行的基因数据的输入模块对于序列

组装软件是至关重要的第一步。Velvet、IDBA 等软件使用串行读取的方式，软件数据输入效率比较低， 而 Ray, YAGA 等并行软件和本文的软件采用的是并行数据输入的方式来加速这一过程。然而，在对数据的实际处理过程中，首先要完成的是数据文件分布存储，并且保证每个进程数据读取的负载均衡。集群系统通过并行文件系统管理分散于各个进程中的文件块，以最优的性能完成用户应用程序的 I/O 请求。良好的并行文件系统不仅需要完成普通文件系统的存储功能，能够完成交换空间和其它临时存储，而且能达到并行文件输入输出高可扩展性。同时，并行计算中负载均衡是一直备受关注的一个问题。如何避免将过多的负载分到一个计算节点上进行计算，而导致其余节点处于等待或闲置状态，必然造成资源的浪费，使得计算性能降低。

为了降低单机存储数据的内存压力，同时便于后续构建分布式双向多步图，本文设计并实现了基于 MPI-I/O 的并行 I/O 模块，将基因数据并行读入并分块存储于集群中的多个进程中，其任务分配策略采用“块分配”策略。整个并行 I/O 存储模块的实现核心算法，如图 4.6 伪码描述。

输入：CPU 编号或者进程号 i, read 集合 $S=\{s_1,s_2,\ldots,s_h\}$, (0<i<p+1)

输出： S_i^*

其中 $\bigcup_{i=1}^{p} S_i^* = S$， $\bigcap_{i=1}^{p} S_i^* = \varnothing$， 并且每个子集合 $|S_i^*| \approx |S|/p$，使得数据分配均衡。

伪代码：

```
ParaFileIO(rank, FilePath)  {
Begin
1.  pFile ← File_open(Filename) ; //打开测序数据文件句柄
2.  size  ←File_getsize(pFile);  //文件总大小，即总的字节数
3.  step  ← size/p;              //文件分块步长
4.  start_pos  ← i * step;       //获得进程起始文件读入数据位置
5.  end_pos    ← (i+1) * step; //获得进程终止文件读入数据位置
6.  do
7.      ch ← File_Read_at(start_pos);
8.      start_pos++;
9.  While  ch ≠ '>'
10. start_pos--;          //找到 start_pos 后面第一个出现'>'的位置，作为该进程的起始地址
11. do
12.     ch ← File_Read_at(end_pos);
13.     end_pos++;
14. While  ch ≠ '>' //找到 end_pos 后面第一个出现'>'的位置，作为该进程的终止地址
15. File_Seek(pFile, start_pos, end_pos)  //为进程设置文件块子视图
```

```
16.  vector <string> ret;
17.  k ←0;
18.  while  read ← gets(pFile)  ≠  NULL
19.       if(k++%2=0)  then  continue;          //丢弃序列片段的标记信息
20.       ret.push_back(string(read));        //收集在[start_pos, end_pos]内的序列片段 reads
21.  Endwhile
22.  return ret;
End
}//FileIO
```

图 4.6 并行 I/O 模块伪码

根据图 4.6 所述，对已给定的一个长度为 g 的参考序列 w，测序产生了 h 个序列片段，序列片段的总长度（碱基总数）为 n。 本文将输入文件均分为 p 个虚拟文件块并分配给 p 个进程分别读取。接着本文对该并行 I/O 模块的计算复杂度、通讯复杂度、进程通讯量三个方面做复杂度分析。通过上述伪码分析，第 1 到 17 行均可用常数次计算完成，而第 18 行将读取 DNA 测序数据文件地址偏移在[start_pos,end_pos]内的所有 reads，这部分的碱基的个数是 n/p，则第 18 到 21 行将读取 n/p 个碱基，并且进程间没有通讯。综上分析，并行文件 I/O 模块的计算复杂度为 $O(n/p)$，进程通讯次数为 $O(1)$，进程通讯量为 $O(1)$。实验结果表明图 4-6 的并行 I/O 存储模块使用 64 核心处理 E.coli 数据集（4.4GB）时需要大约 4 秒，而 YAGA 需要 516.5 秒。 在读取 300G 的炎黄基因组时，SWAP-Assembler 需要 10 分钟，而 Ray 需要大约 2 小时 42 分。同时图 4.7 可以看出，本文中并行 I/O 模块的设计实现了文件分块近乎均衡，保证了 read 数量的分布一致性。

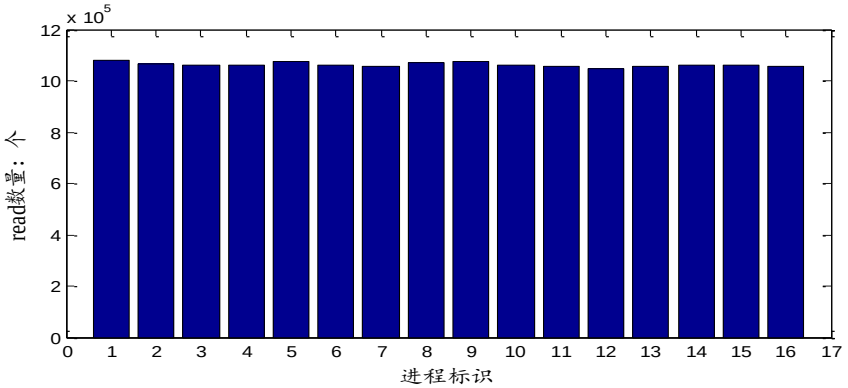

图 4.7 Read 数量分布统计图

4.2.2 双向一步图构建

上一小节已经将序列片段集合均分存储在 p 个进程，得到分块存储的序列片段集合信息为 $S=\{S_1^*,S_2^*,\cdots,S_p^*\}$。本节将根据双向一步图的定义利用每个进程存储的序列片段集合

构建分布式的双向一步图$G_k^1(S)=\{V_s, E_s^1\}=\{\bigcup_{1\le i\le h} V_{s_i}, \bigcup_{1\le i\le h} E_{s_i}^1\}$。这一步中本文使用一个长度为k的窗口在每个序列片段上滑动以切割出k分子的一半（k-mer），同时将相邻的k分子用边关联起来构成一个双向一步图。

为便于对图中顶点进行计算与存储，本文先将所有碱基用二进制编码。表 4.2 列举了碱基编码规则。

表 4.2 二进制编码信息表

编码对象名称	碱基表示	二进制编码
腺嘌呤	A	00
胞嘧啶	C	01
胸腺嘧啶	G	10
鸟嘌呤	T	11

在双向一步图中，需要记录顶点的相关信息，包括 k 分子$\hat{\alpha}$二进制编码信息，k-mer α，α'的二进制编码，顶点出现的频度、顶点的前驱节点及后继节点等等。相关信息以及对应符号见表 4.3 所示。

表 4.3 双向一步图中变量信息表

字符意义	符号表示
k-mer α 的字符编码	kmerID
k-mer α 的二进制编码	PrimeNodeID
k-mer α' 二进制编码	TwinNodeID
k-分子$\hat{\alpha}$ 的二进制编码	CurrNodeID
k-mer α 频度	FrequencyA
k-分子$\hat{\alpha}$ 前驱节点 k-分子$\hat{\beta}$ 二进制编码	PreNodeID
k-mer β 的二进制编码	PrePrimNodeID
k-mer β' 二进制编码	PreTwinNodeID

为便于顶点信息的查找与计算，本文采用 hash_map 结构存储顶点的结构信息，取 k-分子的二进制编码作为该顶点的键值。在计算过程中，所有进程并发的对本身进程中的序列片段集进行处理。在实现过程中，k 分子的 kmerID 用 long long int 类型保存，所以 k 的取值范围为 13-31 之间的奇数。

双向一步图的构建过程包含如下几个步骤：

1) 根据$S_i^*(0\le i\le n-1)$中的 read 集合生成顶点 k-分子集合$V_{S_i^*}=\mathbb{S}(S_i^*,k)$；

2) 对每个 k-分子进行判断，当已经存储到 hash_map 中，将其对应的出现频率加 1；否则，将该 k-分子添加到 hash_map 中，并设置出现频率为 1；

3) 根据 k-分子间的相邻信息，生成图中的邻边信息 E_s^1，最终得到双向一步图 $G_{S_i^*}=(V_{S_i^*},E_{S_i^*})$。

通过对 read 信息的处理，得到双向 De Bruijn 图中一系列顶点的集合 $V_s=\mathbb{S}(s,k)$；根据双向一步边的定义，可知，对于相邻的两个顶点，其连接方式存在着四种类型，那么其连接边便存在着四种不同的连接方式。

以顶点 α，β 为例，来说明具体实现中边的连接方式。其中，flag 用来标记生成的顶点是否是第一个顶点，若 flag=0，表示该顶点没有前驱节点，本身即为第一个顶点，若 flag=1，表示该顶点存在前驱节点。对于双向 De Bruijn 图中的任意两个相邻顶点，其生成图中的邻边的过程描述如下:

步骤 1: 取序列片段集中一条序列 s_i（$0\le i\le n-1$），若 flag=0，那么 PreNodeID=CurrNodeID，PrePrimeNodeID=PrimeNodeID，flag=1，继续步骤 1；若 flag=1，转至步骤 2；

步骤 2：取当前顶点 α 的二进制编码 CurrNodeID, 进行判断，若 CurrNodeID=PrimeNodeID，那么 α 方向记为“+”；取当前顶点 α 的前驱节点 β 的二进制编码进行判断，若 PreNodeID=PrePrimeNodeID，那么 β 方向记为“+”，那么添加邻边 $e^1_{\alpha^+\beta^+}=\{\alpha^+,\beta^+,+,+,C^1_{\alpha^+\beta^+}\}, e^1_{\beta^-\alpha^-}=\{\beta^-,\alpha^-,-,-,C^1_{\beta^-\alpha^-}\}$，否则，转至步骤 3；

步骤 3：若对于当前顶点 α， CurrNodeID= PrimeNodeID，那么 α 方向记为“+”，对于前驱节点节点 β，若 PreNodeID=PreTwinNodeID，那么 β 方向记为“-”，那么添加邻边 $e^1_{\alpha^+\beta^-}=\{\alpha^+,\beta^-,+,-,C^1_{\alpha^+\beta^-}\}, e^1_{\beta^+\alpha^-}=\{\beta^+,\alpha^-,+,-,C^1_{\beta^+\alpha^-}\}$，否则，转至步骤 4；

步骤 4：若对于当前顶点 α， CurrNodeID= TwinNodeID，那么 α 方向记为“-”，对于前驱节点节点 β，若 PreNodeID=PrePrimeNodeID，那么 β 方向记为“+”，那么添加邻边 $e^1_{\alpha^-\beta^+}=\{\alpha^-,\beta^+,-,+,C^1_{\alpha^-\beta^+}\}, e^1_{\beta^-\alpha^+}=\{\beta^-,\alpha^+,-,+,C^1_{\beta^-\alpha^+}\}$，否则，转至步骤 5；

步骤 5：若对于当前顶点 α， CurrNodeID= TwinNodeID，那么 α 方向记为“-”，对于前驱节点节点 β，若 PreNodeID=PreTwinNodeID，那么 β 方向记为“-”，那么添加邻边 $e^1_{\alpha^-\beta^-}=\{\alpha^-,\beta^-,-,-,C^1_{\alpha^-\beta^-}\}, e^1_{\beta^+\alpha^+}=\{\beta^+,\alpha^+,+,+,C^1_{\beta^+\alpha^+}\}$，转至步骤 1。

通过上述五步操作，从而实现了双向一步边的添加, 进而完成了双向一步图的构建

算法最后本文从计算复杂度，通讯复杂度，进程通讯量三个方面对以上算法做复杂度分析。首先，假设 r 表示 DNA 数据中所有序列片段的数量，l 表示 read 的长度，n 表示序列片段中的碱基总数， k-mer 长度是k， p 表示所有参与计算的进程数量。所以双向一步图构建的算法复杂度为$O(n/p)$，进程间通讯次数为$O(1)$，进程间通讯量为$O(1)$。

4.2.3 双向一步图的图过滤

这一步本文尽可能的将错误的 k 分子从双向一步图中删除掉。本文这里假设测序错误是随机的均匀的分布于测序序列片段，而且这个测序错误率在二代测序设备中是介于1%～3%之间的。已有研究发现1%的测序错误可以导致双向一步图中60%～80%的k-mer是错误的，而且这些错误的 k-mer 的频率相比于正确的会比较低。而且通常这种情况下这种错误 k-mer 的频率是正确 k-mer 频率的 3%~10%之间。 因此后面的文章中本文都设定这个相对的阀值为 5%，当然本文也提供参数可以让那个用户自行调节这个阀值。 最终本文将删除 60%～80%k 分子，从而将双向一步图的规模大大压缩。 因此这一步本文可以在线性时间内将计算任务完成，所以其计算复杂度为 O(n/p)。

4.2.3 双向一步图的边合并

以上小节，完成了双向一步图的分布式构建和过滤，对于双向一步图中的顶点，当存在顶点β，使得$e^x_{\alpha\beta}\neq 0, e^y_{\beta\gamma}\neq 0$，$e^x_{\alpha\beta}\cdot d_\beta = e^y_{\beta\gamma}\cdot d_\beta$，$\deg ree(\beta)=2$，$z=x+y$，那么$e^z_{\alpha\gamma}=e^x_{\alpha\beta}\oplus e^y_{\beta\gamma}$，即$\alpha$，$\beta$，$\gamma$满足邻边合并的条件。本小节将重点阐述在 SWAP 异步计算模型下的邻边合并算法的实现，其实现原理如下：

1). 取图中顶点集合$V_s=\mathbb{S}(s,k)$中的任意顶点β，计算其邻接顶点α，λ，查询邻接顶点是否存在，若存在，执行步骤 2，若不存在，转至步骤 1；

2). 对顶点β，若$e^x_{\alpha\beta}\neq 0$，$e^y_{\beta\gamma}\neq 0$，$e^x_{\alpha\beta}\cdot d_\beta = e^y_{\beta\gamma}\cdot d_\beta$，$\deg ree(\beta)=2$，那满足合并条件，转至步骤 3 进行边合并；否则，转至步骤 1；

3). 如果$\alpha\beta\gamma$ 满足邻边融合条件，则去除顶点β，执行$e^z_{\alpha\gamma}=e^x_{\alpha\beta}\oplus e^y_{\beta\gamma}$的操作；

4). 重复上述步骤，对顶点集合$V_s=\mathbb{S}(s,k)$每个顶点进行遍历，直至图中所有顶点均不能进行边合并为止。

这里本文通过对 SWAP 异步并行计算模型的用户自定义函数的重定义来实现对上述原理进行实现，图 4-10 和图 4-11 列出了获取小世界和执行边合并操作的函数伪代码。

Algorithm 3: Pseudocode of GetSmallWorld(R(a,b))

Input: $R(e^{u}_{\beta\alpha}, e^{v}_{\alpha\gamma})$
Output: small world $[e^{u}_{\beta\alpha}, e^{v}_{\alpha\gamma}]$
begin
  $[e^{u}_{\beta\alpha}, e^{v}_{\alpha\gamma}] \leftarrow \{R(e^{u}_{\beta\alpha}, e^{v}_{\alpha\gamma}), e^{u}_{\beta\alpha}, e^{v}_{\alpha\gamma}\}$;
  return $[e^{u}_{\beta\alpha}, e^{v}_{\alpha\gamma}]$;
end

图 4.8 获取 R(a,b)小世界的用户自定义函数

Algorithm 4: Pseudocode of Operation($e^{u}_{\beta\alpha}, e^{v}_{\alpha\gamma}$)

Input: $e^{u}_{\beta\alpha}, e^{v}_{\alpha\gamma}$
Output: $(e^{u}_{\beta\alpha}, e^{v}_{\gamma'\alpha'})^{*}$
begin
  $e^{u+v}_{\beta\gamma} \leftarrow e^{u}_{\beta\alpha} \otimes e^{v}_{\alpha\gamma}$;
  $e^{u+v}_{\gamma'\beta'} \leftarrow e^{v}_{\gamma'\alpha'} \otimes e^{u}_{\alpha'\beta'}$;
  $e^{u}_{\beta\alpha} \leftarrow e^{u+v}_{\beta\gamma}$;
  $e^{v}_{\gamma'\alpha'} \leftarrow e^{u+v}_{\gamma'\beta'}$;
  return $(e^{u}_{\beta\alpha}, e^{v}_{\gamma'\alpha'})$;
end
* α' is the reverse complement of k-mer α.

图 4.9 用户自定义的操作函数

给定一个长度为 g 的基因组参考序列 w, 如果测序错误，测序空隙，基因组重复片段服从均匀分布，那么根据定理 4 本文有如下推论：

推论 1. 给定一个长度为 g 的基因组参考序列 w，如果在测序错误，测序空隙，基因组重复片段中的 k 分子的百分比是 q, 那么基因组参考序列会被切割为 qw+1 段子序列或者 contig。而且其中最常的子序列或者 contig 满足 $\Pr ob\{C_{\max} \geq 3c \cdot q\log(g)\} \leq \frac{1}{g^{c}}$，这里 c>2。

推论 1 表明最长的子序列或者 contig 的长度的上限很大概率是 $c \cdot q\log(g)$。 这里 c 是一个和基因组本身相关的常数。

图 G(σ,A)是由一系列的无分叉路径组成的，这里每个无分叉路径都可以被合并为一个 contig。而这里这些合并得到的子序列或者 contig 的长度最长只能是图 G(σ,A)的直径，而且直径的上限是 $c \cdot q\log(g)$。 根据公式 4-8，当处理单元的个数接近或者大约操作的个数时，边合并操作的最大通讯轮数是

$$\text{CommRound} = b \cdot \log(3c \cdot q\log(g)) \qquad \text{（公式 4-10）}$$

而总的通讯数据量实际上与参与的处理单元的个数无关，而与通讯轮数有关。在图 4-4 中每个双向一步边都带有一个长度为 1 的字符串标注。而当双向一步边合并为双向二步边时这个标注的长度会加倍。这个标准字符串的长度在每次合并后几乎都会加倍，而该字符串标注是本文在数据传输中的主要信息，所以这一步的总的数据通讯量为

$$F=\sum_{i=0}^{CommRound}2^i\times\frac{n}{2^i}=b\cdot\log(3c\cdot q\log(g))\times n \qquad \text{（公式 4-11）}$$

根据图 4-11 中的算法 4，用户自定义的边合并操作函数的计算复杂度为 O(1)。 所以合并公式 4-7，公式 4-10 和公式 4-11， SWAP-Assembler 在双向多步边合并这一步中的计算复杂度为：

$$RunTime=(bnL\cdot\log(3c\cdot q\log(g))+rnS+n)/p \qquad \text{（公式 4-12）}$$

最后通过对双向多步图中所有半扩展顶点进行邻边合并，最终生成了多条长度不等的 contig 结构集合。同时本文从公式 4-10, 公式 4-11, 公式 4-12 中可以得出，在双向多步边合并操作的计算复杂度为 O(n/p), 通讯数据量为 O(nlog(log(g))/p), 通讯轮数为 O(log(log(g))).

4.2.4contig 扩展

对于双向多步图中的顶点，根据图中顶点度的数量，可以将图中的顶点分为四类描述，如图 3-4 所示：

图 4.10 路径信息分类图

由图4.10所示，图中用圆表示双向多步图中的顶点，双向边表示顶点之间的相邻关系，为了说明图中的邻接关系，在此，默认图的扩展方向向右。图中的深色圆表示目标顶点$\hat{\alpha}$。图 4.10 中， 子图(a)、子图(b)、子图(c)均存在着多条路径。子图(a)中，经过目标顶点的可能路径存在A_3^3种情况；子图(b)中经过目标顶点路径有两条；子图(c)中目标顶点的情况与(b)相似，经过目标顶点路径有两条；图(d)中经过目标顶点的路径只有一条，即$\deg ree(\hat{\alpha})=2$ 。

造成多路径出现的原因有很多，如新一代测序数据高覆盖度的特点，碱基的替换、插入、删除，以及基因序列碱基序列污染、错误序列识别等等，需要依靠特定的生物信息知识才能具体确定。如果对上述子图(a)、 (b)、 (c)中的目标顶点$\hat{\alpha}$去除，并将其左右相邻的两条边进行合并，在没有更多信息的情况下势必造成左右两边路径的错配，从

而得到错误的 contigs。图中顶点 90%均满足第四种情况，因此可以对子图(d)中的目标顶点进行边合并操作。 而在合并完成后的图中的顶点主要是(a), (b), (c)这三种情况，为进一步扩展 contig 的长度，本文需要从收缩后的图中尽可能的寻找有用信息扩展 contig。

由于标准基因组装问题可以在共享内存的服务器上以 O(n)的计算复杂度完成，同时综上所述的序列组装的五个步骤及其算法复杂度分析， 本文的并行序列组装软件 SWAP-Assembler 的计算复杂度为 O(n/p), 通讯复杂度为 O(nlog(log(g))/p), 通讯轮数为 O(log(log(g)))。 SWAP-Assembler 相对于串行算法的加速比为：

$$Speedup = \frac{n}{RunTime} = \frac{p}{bL \cdot \log(3c \cdot q \log(g)) + r1S + 1} \qquad \text{（公式 4-13）}$$

公式 4-13 表明，对于一个给定长度为 g 的未知基因组，SWAP-Assembler 的加速比将随着处理单元的个数增加而线性增加，当处理单元个数固定时， 其加速比将和 log(log(g))成反比减小的趋势。

4.3 本章小结

为优化半群操作的计算并保证其在高性能平台上的扩展性和高效率，由此本文设计了 SWAP 异步计算机制，该模型通过使用通过“加锁-计算-解锁”的机制实现子集同步全局异步的计算模型，最大化的提高半群系统上边合并操作的并行度。接着本文阐述了该异步并行计算模型的具体实现算法，并对其计算通讯复杂度抽象和分析，最后本文从理论上证明该异步计算模型具有高扩展性。该机制提供了高效计算通讯模式，负载均衡策略，以及支撑图算法的半群代数数学模型和表达。相比于大同步计算模型，SWAP 异步计算机制可提供数据一致性保证。该机制可以由底层异步通讯实现，并使用类似 802.11 协议的自动碰撞规避算法以最大化挖掘上层图算法的并行性。接着基于该计算模型本文逐一阐述了并行序列组装软件 SWAP-Assembler 的实现算法，具体实现中 SWAP-Assembler 包括五个步骤：并行 I/O， 双向一步图构建， 图过滤，图收缩，contig 扩展。最后本文通过理论证明该软件总的计算复杂度为 O(n/p), 通讯复杂度为 O(nlog(log(g))/p), 通讯轮数为 O(log(log(g)))。

第五章 高可扩展序列组装软件 SWAP2 性能深度优化技术

本章对 SWAP-Assembler 中最耗时的几个模块进行分析和优化，使得该软件最终可以运行在十万核以上，同时可以处理 TB 到 PB 级的基因组数据。首先本文对 SWAP-Assembler 记录五个模块的运行时间并绘制百分比耗时图，根据分析本文发现 SWAP-Assembler 中最耗时的三个模块是并行化 I/O 模块， 双向一步图构建模块，双向多步图收缩模块（边合并）。因此本文对这三个模块进行深度分析并对每个模块提出性能优化方案以大幅提高序列组装软件的扩展性和并行效率。 最终本文优化后的软件（命名为 SWAP2），在处理 4TB 的千人基因组数据时可以扩展到 131，072 核并达到 40%的并行效率。 同时本文使用 300GB 的人类基因组数据做性能对比试验，相比于现有性能最高的并行序列组装软件 HipMer[41]，SWAP2 的处理时间缩减到了原来的 1/3， 扩展性提高了 4 倍达到 65,536 核。同时 SWAP2 在炎黄基因组数据上的性能比优化前的 SWAP-Assembler 的处理数据提高了 45 倍。

5.1 SWAP-Assembler 性能瓶颈分析

上一章基于双向多步图的半群系统和 SWAP 异步并行计算模型设计实现了 SWAP-Assembler 并行序列组装软件。这里本文将 SWAP-Assembler 中并行实现的四个模块并行化 I/O 模块， 双向一步图构建模块，图过滤， 双向多步图收缩模块（边合并）进行性能测试和分析。图 5-1 收集了 SWAP-Assembler 在处理 4TB 的千人基因组时这四个模块的时间消耗。

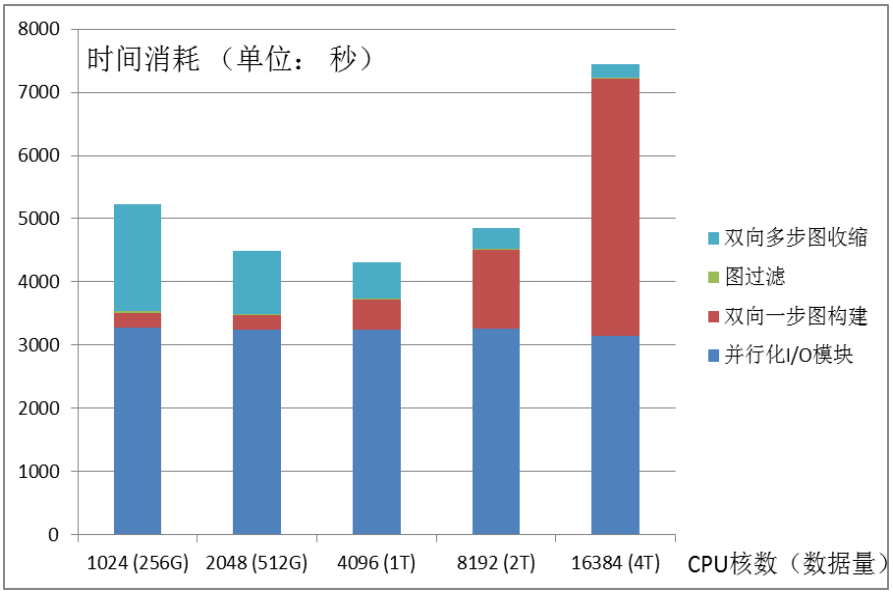

图 5.1 SWAP-Assembler 四个模块在组装 4TB 的千人基因组时的运行时间

从图中可以发现 SWAP-Assembler 的时间消耗主要集中在并行化 I/O 模块， 双向一

步图构建模块和双向多步图收缩模块（边合并）。具体而言，并行化 I/O 模块消耗了近一半的时间， 双向一步图构建模块的时间消耗随着计算核的增加而剧烈增加，双向多步图收缩模块的时间消耗基本持平或少量减少。

为了提高 SWAP-Assembler 的并行化效率并尽可能的保持每个模块的耗时时间占比恒定，接下来本文对这三个模块进行了深度的分析和性能优化。

5.2 并行 I/O 性能优化

将 TB 甚至 PB 级的基因数据载入内存非常耗时， SWAP-Assembler 使用了和 Ray，YAGA 类似的并行化策略简单的将基因数据分成若干文件块分配给各个进程读取。 该策略的计算复杂度上限是 O(n/p), 这里 n 是基因数据文件的大小或者总的碱基数量。然而这里有两个因素会影响这一步的性能， 第一是基因数据都是以 FASTA 或者 FASTQ 格式存储的，而粗暴的分割会导致部分基因片段被分割在不同的文件块中。第二这里也没有一些可调节的参数将并行文件系统的 I/O 性能提高的极限。

SWAP-Assembler 使用的数据分割策略可能会把以 FASTA 或者 FASTQ 存储的基因序列片段中间割断，并分配给不同的进程。 为了解决这个问题，SWAP-Assembler 使用了一个寻找序列片段起始点的函数来寻找文件块的开始位置。但是这个函数会带来额外的 I/O 开销。

为了克服这个缺点，本文提出了一种文件块动态调整算法(FAA)来替代现有的起始点寻找函数，见图 5.2 的算法 1。这里每个进程都是以大块的数据块的形式读取数据同时自动调整记录分给每个进程的文件块的开始和结束位置信息使得文件块不会从中间割断序列片段。一个进程 ID 为 ProcID 的进程将它的文件块的起始位置定位于任何一个序列片段的开始位置，同时将这个位置信息发送给它的前一个进程 ProcID-1。那么进程 ProcID-1 就会将这个位置信息作为其文件块的结束位置信息。 在这个操作之后，实际上每个序列片段都被分配且唯一的分配给了某一个进程的文件块，而不会存在某一个文件块被切断分给两个进程文件块的情况。除此以外，每个进程在读取数据块时都会以 data block 的形式进行块读取，而这样的每个 data block 的大小被设定为一个可以调整的参数用来优化具体的文件系统的并行 I/O 的系统性能，并使其达到系统性能最优值。

下面本文来评估这个 FAA 算法以及 data block 数据块的大小所带来的 I/O 性能的提升。这里本文用千人基因组序列构造了一个用于弱扩展性能测试的输入数据集，随着处理核数从 1024 核增加到 16,384 核，这个输入数据集的数据大小从 256GB 增加到 4TB。同时本文使用阿贡国家实验室的 IBM Blue Gene Q (Mira) [114-117]作为本文的性能测试集群。图 5.3 展示了这个性能测试的时间对比数据。本文可以看出相比于 SWAP-Assembler， FAA 算法节省了大约 60%的并行 I/O 数据读取时间。在数据块的大小从 4MB 增加到 64MB 的时候，并行 I/O 的时间消耗随着数据块大小的提高而不断减小。试验结果表明更大的数据块大小有助于减少并行文件系统的 I/O 操作数并有效的提高了

数据 streaming 效果。但在兼顾性能和内存开销的情况下，本文无法不断的提高数据块的大小，所以在接下来的测试中本文将数据块的大小设定为 64MB。当使用 16，384 核心时，本文最终在并行 I/O 模块获得了 16 倍的性能提升。

图 5.5 中列出了并行 I/O 模块并行 I/O 效率与底层并行文件系统的理论峰值的比值结果。 Mira 中每个 I/O drawer 的 I/O 带宽为 32GB/s， 那么每个机柜的 I/O（包含一个 I/Odrawer）性能为 32GB/s [114-116]。图 5-5 中可以看出使用 16,384 个核心时，SWAP2 可以达到大约 16.5%的并行 I/O 的系统理论峰值。

Algorithm 1: Fragment Adjustment Algorithm.

Input: Dataset S in FASTA or FASTQ format, the rank of local process $procID$ and the total number of processes p.
Output: Virtual fragments $S_1, S_2, \ldots, S_p$.
begin
  $size$ = the file size of dataset S;
  $step = size/p$;
  $start = procID * step$;
  $end = (procID + 1) * step$;
  $end = end < size \ ? \ end : size$;
  $readBuf$ = Read one *data block** starting from $start$;
  $i = 0$;
  while $readBuf[i] \neq$ '>' **do**
    i++;
  $sendAdjustDelta = i$;
  if $procID \neq 0$ **then**
    Send $sendAdjustDelta$ to process $procID - 1$;
  if $procID \neq p - 1$ **then**
    receive $recvAdjustDelta$ from process $procID + 1$;
  $start$ += $sendAdjustDelta$;
  end += $recvAdjustDelta$;
  $S_{procID} = (start, end)$;

* Here the data block size will be larger than the length of reads to ensure that every data block contains at least one start symbol of the read '>'.

图 5.2 SWAP2 中并行 I/O 模块的 FAA 算法

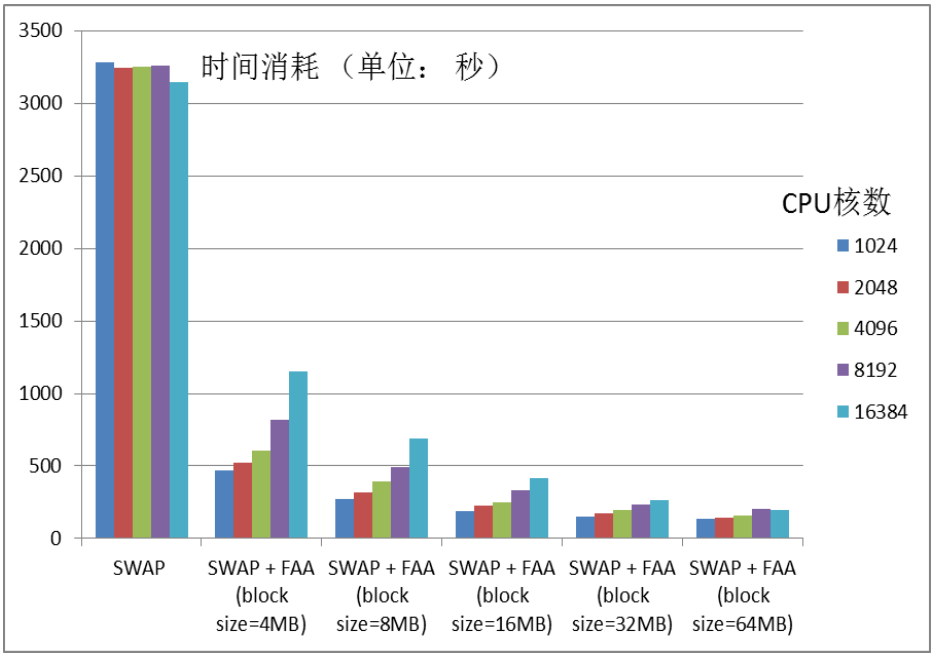

图 5.3 FAA 算法及 data block 数据块的大小所带来的并行 I/O 性能提升

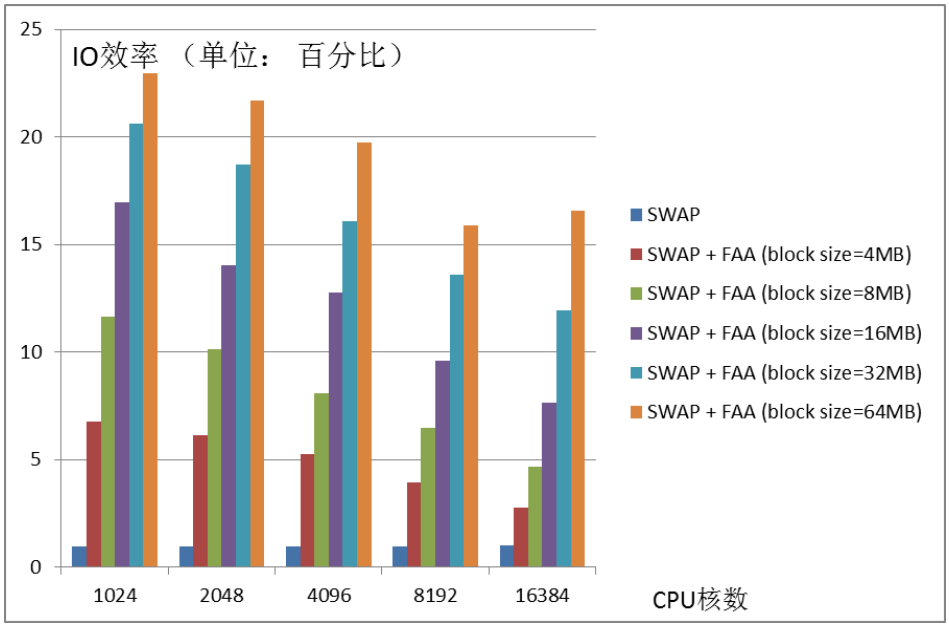

图 5.4FAA 算法以及 data block 数据块的大小匹配值所对应并行 I/O 效率

5.3 双向一步图性能优化

在这个模块中，SWAP-Assembler 将构造一个由 k 分子组成的双向一步图。这一步涉及三个阶段：k 分子生成， k 分子分发，k 分子存储。 在这一个阶段输入序列会把基因序列用一个长度为 k 的窗口切割为连续重叠的 k 分子，在第二个阶段，这些 k 分子会按照一个卷积哈希函数分发到对应的进程去，由于每个基因序列都会产生序列长度减 k 个 k 分子，如果输入数据达到 TB 甚至 PB 级别，那么将产生万亿甚至千万亿个 k 分子用于分发（通讯）。在第三个阶段，这些分发的 k 分子会被直接聚合并存储在本地。

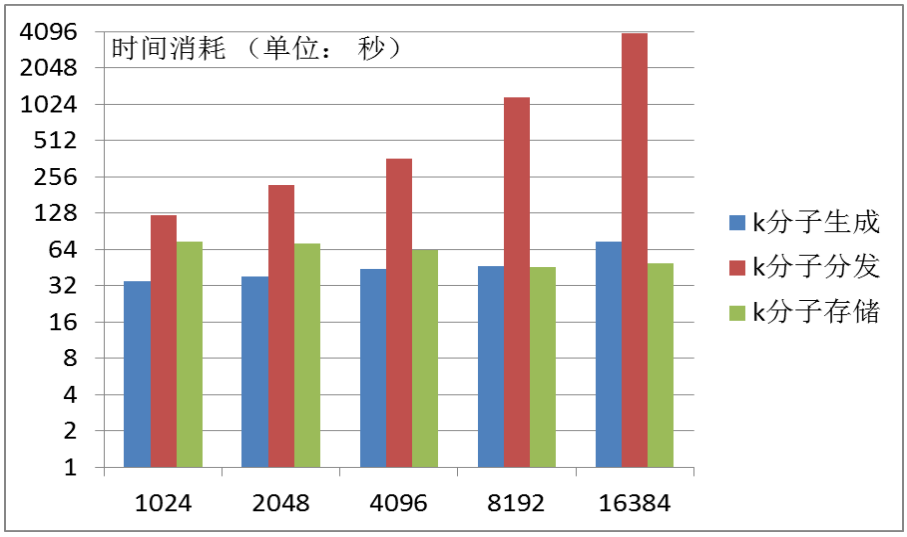

图 5.5双向一步图构建的三个阶段耗时对比图

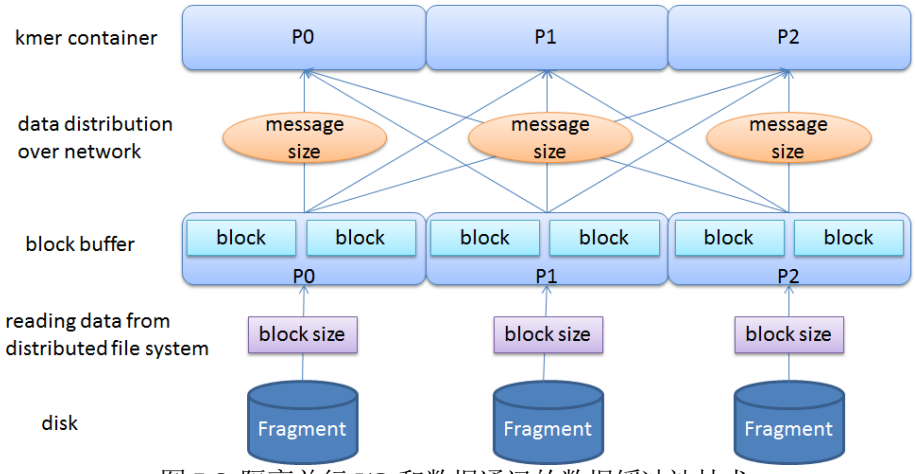

图 5.6 隔离并行 I/O 和数据通讯的数据缓冲池技术

图 5.5 收集了这 SWAP-Assember 在处理 4TB 的千人基因组时三个阶段的时间消耗情况。图中结果说明这个模块的主要性能瓶颈在于 k 分子的分发，当处理核数从 1024 核增长到 16，384 核时，这个阶段的时间占总时间的比例从 53%快速增长到了 97%。由于这个阶段主要的工作负载就是分发 k 分子，具体来说每个进程将每个 k 分子按照一个卷积哈希函数计算出其目标进程，并将该 k 分子发送到目标进程，同时该进程还需要接受所有发往本进程的所有 k 分子。 在 SWAP-Assember 中，这个总的数据通讯量在理论上是固定不变，所以这个时间消耗应该也是恒定不变的，但当参与计算的进程加倍的时候，那么这个任意两个进程间通讯消息的载荷就会减半，而过小的通讯载荷将使得通讯效率急剧下降，并会直接影响到这一模块的性能

为了提高通讯效率并防止通讯消息携带过下的通讯载荷，本文使用如下 3 个优化策略来优化本模块的通讯效率：

数据压缩 在 k 分子生产的阶段，本文将两个共享一个 k 分子的双向边压缩为一个边，这样可以将数据的通讯量降低一半。

消息载荷调优 为了防止通讯消息过小的通讯载荷，本文将通讯消息的大小固定并使之与参与通讯的进程数无关。在一轮通讯迭代中，本文将每个进程处理的碱基个数固定为 L, 因此每个进程将最多产生 L 个 k 分子，同时这个进程没轮通讯中处理的碱基个数随着进程的增加等比例的增加。 这样每次通讯过程中任意两个进程间通讯的载荷(k 分子的个数)将保持恒定并约等于 $L \times B_k / p_0$，这里 B_k 是每个 k 分子的数据结构大小，p_0 是起始进程数，这里 p_0=1024。这样在每轮通讯过程中任意两个进程间通讯的载荷是基本恒定的。

本文可以通过调整初始的碱基数量 L 来调优网络的通讯效率，同时本文也可以通过调整并行 I/O 数据块的大小来调整并行 I/O 的系统效率。然而任意的调整这两个参数可能会导致互相间的参数干扰，使得参数化调优策略无效。例如当本文的并行 I/O 数据块的大小设置为 1MB，每轮进程处理的碱基数量为 1024, 那么当参与通讯的进程数大于 1024 时，本文需要至少 1 百万的 k 分子来分配以保持通讯消息的载荷恒定，然而这里下层的并行 I/O 模块却无法提供足够的数据用来产生足够的 k 分子，这种情况下通讯消息的载荷将无法保持恒定并开始减小。

并行 I/O 和通讯隔离 为了解决上面的问题，本文设计了一个数据池（data pool）用来缓存并行 I/O 模块和双向一步图构建模块间的基因序列数据，并将两个模块隔离开来独立运作。在图 5.6 中，为了隔离这两个模块间的 I/O 和通讯部分，本文让这两个部分共享一个数据池(data pool)，这个数据池在这里扮演的是一个基因序列片段的阻塞队列。有了这个数据池之后，图构建模块的通讯部分可以不停的从数据池中取出序列分割为 k 分子并进行分发通讯，而并行 I/O 的数据 I/O 部分可以不停的向这个数据池中存入基因序列。只有当这个数据池满或者空时，并行 I/O 部分或者通讯部分才会阻塞，其他情况下这两个部分都可以并发的运行而且互相不干扰。这里只要将这个数据池设置的足够大，就可以保证通讯消息的载荷恒定，同时用户可以自由的调整 I/O 数据块的大小和通讯载荷 L 的大小以逼近 I/O 和通讯系统的性能极限。

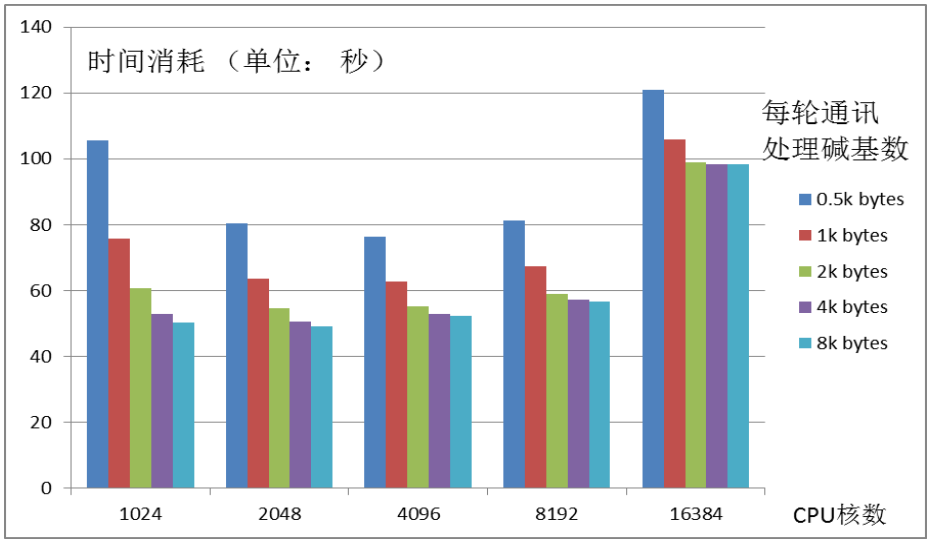

图 5.7 k 分子分发阶段数据通讯所消耗的时间统计

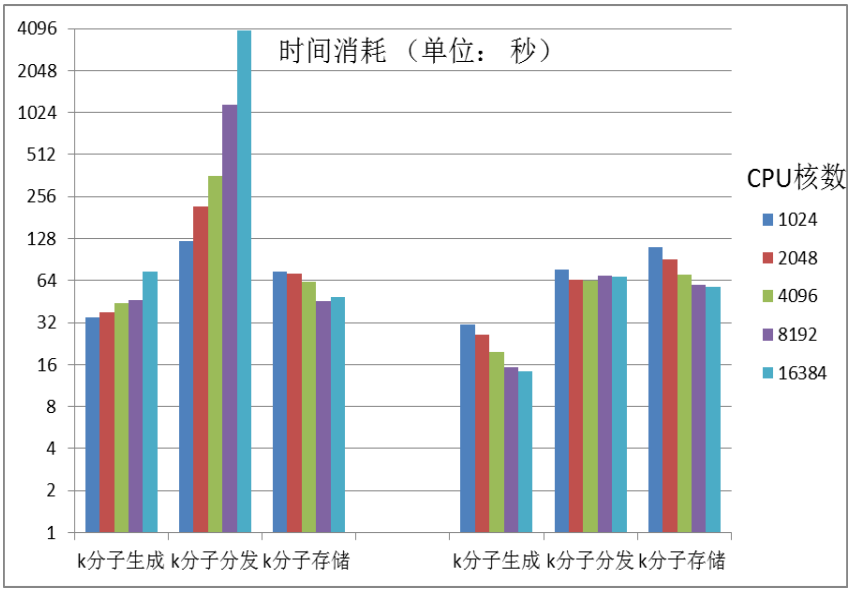

图 5.8图构建的三个阶段在优化前后的耗时对比分析

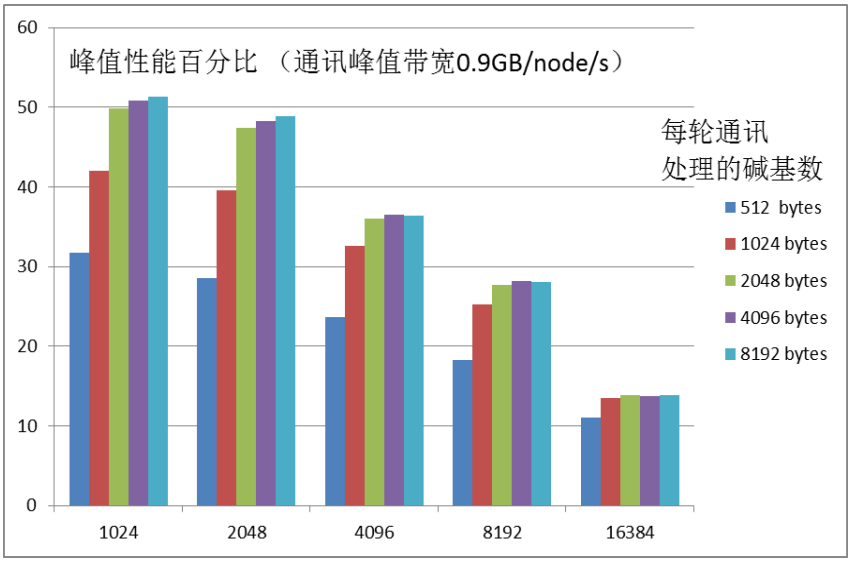

图 5.9 图构建过程中数据通讯带宽占通讯网络理论峰值的百分比

为了寻找最佳的通讯载荷，本文设计了一个若扩展性能测试以找到碱基数量 L 的值使得通讯性能最优，这里碱基数量 L 从 512 字节增加到 8192 字节。为了收集到细致的通讯时间消耗本文在 MPI 通讯开始和结束的代码中插入一些时间戳(Timer Tag)，这些数据被绘制在图 5.7 中。从图中可以发现，对已一个固定长度的碱基数量 L， 通讯时间随着参与通讯的核心的数量的增加而增加；当本文提高碱基数量 L 时，通讯时间逐渐减小，但是这种趋势随着核数的增加同时在减弱。为获取较高的通讯效率，本文在后续讨论中将 L 设置为 8192。

双向一步图构建的三个阶段的耗时数据被绘制在图 5.8 中。在第一个阶段中，k 分子生产所消耗的时间比之前大幅降低，并达到了 5.2 倍的加速比。在第二个阶段，随着处理核数的增加，k 分子分发通讯所消耗的时间基本维持不变，达到较好的弱扩展性能，当处理核数达到 16，384 核时，优化策略取得了 64 倍的加速比。因为没有对最后一个阶

段进行特别优化，所以前后的运行时间的趋势相同。

最后本文对本模块的通讯效率做了实验对比分析。 这里 Mira 使用的网络拓扑结果是 5D-torus 网络，所以其网络带宽的理论极限与其路由维数中最长的一条有关，这里本文使用了 4096 个计算节点，所以最长的一维路由条数是 16，那么对于每个计算节点其网络的峰值通讯速率是 0.9GB/s [118, 119]。 那么本文以此为网络的峰值带宽在图 5-9 中绘制出了图构建模块中网络通讯效率百分比。图中的结果显示当每轮通讯中处理的碱基数 L 提高时系统通讯效率就随之增加，但是当处理核数规模增大时从 1024 核（1/4 个机柜）增长到 16，384 核（4 个机柜）时，系统通讯效率从 50%降低到了 15%。

5.4 双向一步图收缩性能优化

在图收缩模块中，本文通过重定义了 SWAP 异步并行计算模型中的两个用户自定义函数，将无分叉路径上的边合并为 contig。 前一章中本文具体描述了 SWAP 模型的加锁-计算-解锁的机制，同时本文采用了 SWAP 线程和服务线程来协作实现以上工作机制。在 SWAP 线程中，图中每个顶点需要发送一个 Lock 信息给它的邻居顶点，当且仅当这个顶点在接受到所有邻居顶点的加锁成功 ACK 消息后，这个顶点才能转换为确认状态。否则这个顶点将发送 unlock 信息去解锁那些已经加锁成果的顶点。 在确认状态中，每个顶点将发送本地的数据和计算指令给其邻居顶点，去合并在这些数据上做相应的计算任务。 在服务线程中，本文设计了一个 while 循环来侦听指令消息和数据消息，在接受到指令消息后服务进程会按照指令进行运算或者消息回复。

图 5.10 中用三个顶点举例说了这个 SWAP 模型中的通讯协议工作机制。 顶点 0 发送了一个 Lock 消息给顶点 1，在收到加锁成功的 ACK 消息后，顶点 0 接着发送 Lock 消息给顶点 2，并等待接受 ACK 消息。 这里本文需要两个通讯循环才能将顶点 1 和顶点 2 加锁。 在接收到两个加锁成功的 ACK 消息后，顶点 0 会在接下来的两个通讯循环里分别发送一个指令消息并捎带上计算相关的本地数据到顶点 1 和顶点 2。 综上分析，算法 2 总共需要 4 个通讯循环才能完成 SWAP 模型的工作机制。影响 SWAP 模型的底层通讯效率的一个关键因素是发送一个消息给目标顶点后等待一个 ACK 回复消息的一个来回时间(Round-trip latency)。在算法 2 中服务线程只有当接受到一个指令消息后才能开始计算，对于像 Mira 这样的拥有百万核规模的超级计算机，更多的参与计算的核心意味着更大规模的网络，这将会带来更长的通讯延时，图 5-11 中已经有性能测试数据也表明这个通讯延迟随着处理核数的增加而不同增长。

为了将这个通讯延迟所带来的时间消耗缩小，本文通过减少通讯轮数和共享通讯等待的延迟来提高 SWAP 模型的通讯效率。 这里本文介绍一种优化的通讯机制，并绘制在在图 5.12 中，这里每次顶点 0 都会一次同时发送两个 Lock 消息给它的两个邻居顶点，并同时等待这两个邻居顶点的 ACK 回复消息。当接受到两个加锁成功的 ACK 消息后，顶点 0 同时会发送两个指令消息及捎带数据到顶点 1 和顶点 2 用于半群操作的计算。这

样这个优化的通讯机制只需要 2 个通讯循环，并在内部通讯过程中共享了通讯延迟以提高通讯效率并缩短整个工作机制的运行时间。

最后本文用千人基因组的若扩展数据集来测试以上优化的通讯机制的实际性能效果。在图 5.13 中左边是优化前的运行时间，右边是优化后的运行时间。其中左边（优化前）的运行时间表明当运行核数增加到 32，768 核心时，通讯等待时间占总时间的 85%。而右边（优化后的通讯机制）的运行时间表明随着处理核数的增加，通讯等待时间基本保持在总运行时间的 40%左右。

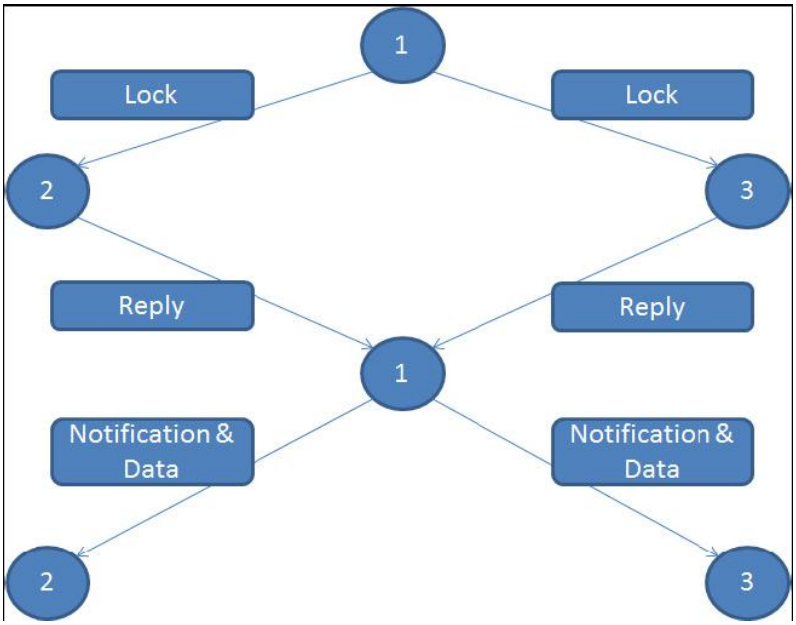

图 5.10SWAP 模型中的消息通讯工作机制

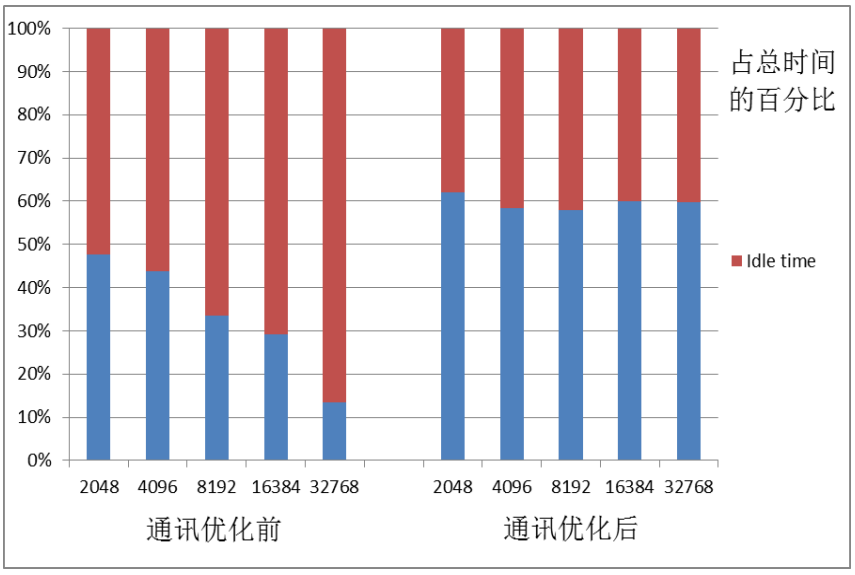

图 5.11 SWAP 异步并行计算模型的消息通讯机制实际性能优化效果

Algorithm 2: Communication protocol for lock-computing-unlock schedule in SWAP. Here the protocol is divided into two routines for the SWAP thread and service thread [10].

```
begin
    Routine in SWAP thread;
    Lock Stage:
    Post MPI_Isend(compReq);
    Post MPI_Irecv(compReq + 1);
    Reply Call RecvProc(2, compReq);
    Notify Stage:
    Post MPI_Isend(compReq);
    Post MPI_Isend(compReq + 1);
    Call RecvProc(2, compReq);

    Routine in service thread;
    while true do
        Post MPI_Testall(2, compReq, &flag);
        if flag then
            break;
        Post MPI_Test(&globalReq, &flag);
        if flag == 0 then
            continue;
        Doing computation work here ...;
        Post MPI_Irecv(&globalReq);
```

图 5.12 SWAP 异步并行计算模型中消息通讯机制的 SWAP 线程伪代码

Algorithm 3: Optimized communication protocol for lock-computing-unlock schedule. Here the calls to the compute routine on two vertices have been integrated into one routine.

```
begin
    Routine in SWAP thread;
    Lock Stage:
    Post MPI_Isend(compReq) ;
    Post MPI_Irecv(compReq+1) ;
    Post MPI_Isend(compReq+2) ;
    Post MPI_Irecv(compReq+3) ;
    Call RecvProc(4,compReq) ;
    Notify Stage:
    Post MPI_Isend(compReq);
    Post MPI_Irecv(compReq+1);
    Post MPI_Isend(compReq+2);
    Post MPI_Irecv(compReq+3);
    Call RecvProc(4, compReq);

    Routine in service thread;
    while true do
        Post MPI_Testall(2, compReq, &flag);
        if flag then
            break;
        Post MPI_Test(&globalReq, &flag);
        if flag == 0 then
            continue;
        Doing computation work here ...;
        Post MPI_Irecv(&globalReq);
```

图 5.13SWAP 异步并行计算模型中优化后的消息通讯机制的 SWAP 线程伪代码

5.5 本章小结

本文使用自适应的自动优化策略最大的逼近底层高性能计算平台及其文件系统的理论极限。在优化的过程中尽量使用体系结构无关的参数空间性能自动策略以达到优化策略的通用性。为了使计算、通讯、I/O 同步优化避免干扰，本文使用数据缓冲池进行速率缓冲匹配，为各步骤提供参数空间进行性能自动优化挖掘底层超算平台软硬件性能极限，同时支持计算通讯 I/O 的叠加(Overlap)技术。 除此以外，本文为深度优化图算法底层库，使用并行图数据 streaming I/O 技术,内存数据压缩技术，消息通讯聚合压缩技术以提升现有 SWAP-Assembler 的问题规模，系统效率和扩展性。

对于不同的(并行)文件系统及其其支撑硬件平台，其达到最大吞吐性能的IO进程数和读写文件块的大小是不同的，本文通过动态搜索其参数空间来搜索其达到最佳性能的参数值，并使得系统稳定在这个参数值上。对于在多样的通讯网络上（例如胖树，5D torus, Cray Aries等）运算的多样的图算法，其通讯模式也是不一样的。为了使得给定的图算法在给定的通讯网络上达到最佳的通讯性能，本文通过在算法中调整通讯进程数，通讯消息的大小，消息压缩的算法来搜索整个参数空间，降低通讯等待时间以最大化通讯带宽。

最后在做以上多重优化的过程中，为使得计算，通讯，IO 在优化过程中互相不干扰，而且同时达到最优，本文提出了图5.6中的通讯和IO的隔离技术。这里本文使用一个数据块缓存池来缓存 IO 数据库和通讯数据块，这样底层库 IO 模块可以探寻最优 IO 性能，同时上层的通讯模块可以探寻其最优通讯性能。 此外该隔离技术使得本文可以深度使用重叠技术来重叠计算，通讯和 IO 最终使得整个底层库在支撑上层图算法时达到最佳的系统性能，使得整个系统的性能得以最大的释放。

第六章 性能分析与评价

6.1 序列组装开发环境与测试实验平台

SWAP-Assember 序列组装软件基于中国科学院深圳先进技术研究院曙光 5000 高性能集群系统开发完成。曙光 5000 是一个以消息传递的松散耦合大规模并行计算机系统，可以提供科学计算服务，同时，也是面向信息服务的超级服务器。系统共有 2 台文件存储系统服务器，72 个计算节点。系统计算节点采用 4 路 4 核 AMD shanghai 处理器，主频 2.2GHz，每个节点内存为 32GB。整个系统共有 1152 颗 CPU 与 2304GB 内存资源供使用。所有节点采用延迟小于 1.3us 的 Infiniband 高速网络进行互联。曙光 5000 集群采用 LUSTRE 并行文件系统，每个计算节点统一为 SuSe Linux 企业版 10.0 操作系统。

在 SWAP-Assember 序列组装软件的精度和质量测试中，本文使用国家超级计算中心-天津中心的 Tianhe 1A 作为本文测试的高性能集群[120, 121, 122]。天河 1A 的高性能集群系统包含 7168 个计算结点（或 14336 核）， 226TB 内存和 2PB 存储。每个计算结点包含 2 路 6 核英特尔 X5670 CPU，一块英伟达 M2050 GPU（1.15GHz、14 核/448 个 CUDA 核）和 24G 内存，总计 14336 个计算 Intel X86 核心和 172TB 的内存。天河 1A 通信网络采用自主设计的高阶路由芯片 NRC 和高速网络接口芯片 NIC，实现光电混合的胖树结构高阶路由网络，链路双向带宽 160Gbps，延迟 1.57us。分布式存储系统主要采用 Lustre 全局分布共享并行 I/O 结构，具有 6 个元数据管理结点，128 个对象存储结点，总容量 2PB。每个计算节点上都安装有 64 位麒麟 Linux。

考虑到测试 SWAP2 序列组装软件的强扩展性和弱扩展性，本文 SWAP2 序列组装软件测试实验基于美国阿贡国家实验室顶级计算设施(ALCF)下属的超级计算机 IBM Blue Gene Q – Mira 系统测试完成[114-119]。该超级计算机于 2012 年由 IBM 建造完成，现居全球 Top500 排名第 5 名[123]， Graph500 排名第 3 名[51]。系统由 48 组机柜组成，共有 49152 个计算节点，每个计算节点由 16 颗 IBM PowerPC 架构、主频为 1.6GHz 的 IBM Power A2 处理器组成。每个计算节点内存为 16GB，最大访存带宽达到 43GB/s。网络拓扑架构使用 5D-torus 网络，计算节点点对点的网络峰值带宽为 40GB/s，全局通讯网络峰值带宽为 0.9GB/s。IBM Blue Gene Q – Mira 系统采用 IBM GPFS 分布式文件系统， 总存储空间达到 70PB，共有 786432 CPU 个内核。 每个机柜（或 1024 个计算节点）装备一个 I/O drawer， 每个 I/O drawer 的 I/O 峰值带宽是 32GB/s，全系统聚合 I/O 带宽是 470GB/s。 该计算机主要使用水冷技术制冷，单机柜 Linpack 峰值性能是 209.7Tflop，能耗大约为 100kW。IBM Blue Gene Q-Mira 超级计算机组装结构如图 6-1 所示。

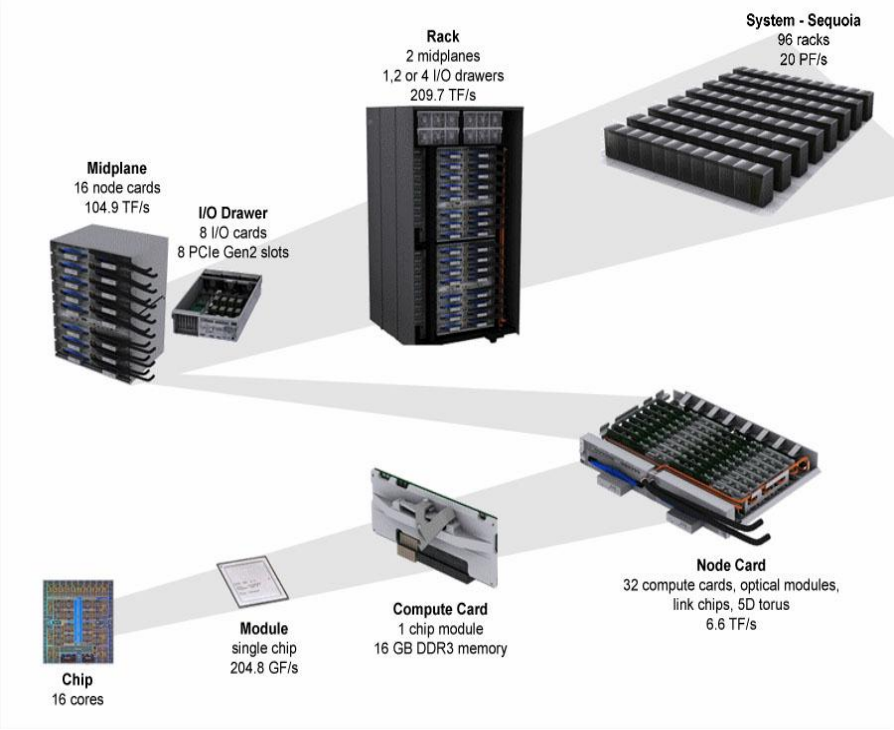

图 6.1 IBM Blue Gene Q-Mira 超级计算机的组装系统结构

6.2 序列组装实验测试数据

基因测序所产生的数据主要采用 FASTA 和 FASTQ 格式[124]，本文中所使用的数据均是 FASTA 格式。FASTA 是一种基于文本用于表示核苷酸序列或氨基酸序列的格式，在 FASTA 格式中，碱基对或氨基酸用单个字母来编码，且允许在序列前添加序列名及注释，该格式如图 6-2 所示。在 FASTA 文件中，每条序列片段的信息包括了说明信息以及序列片段的序列信息。其中每一条 read 的说明信息是以‘>’为信息开始标识，该信息包含了该序列片段的是由那种测序仪测序产生，它所属的文库 ID，该序列片段在物种中染色体可能的位置以及其它生物信息。从第二行开始为序列本身，只允许使用既定的核苷酸或氨基酸编码符号。

```
>read_1_gi|330443391|ref|NC_001133.9| Saccharomyces cerevisiae S288c chromosome
I, complete sequence_-_81401
AAAAGTATTAAAAATGAAAGAATGTGGGACAAAATA
>read_2_gi|330443391|ref|NC_001133.9| Saccharomyces cerevisiae S288c chromosome
I, complete sequence_+_175921
ATAATGCGCCAAGCCTTTATAAGGAACTCAAAATAA
>read_3_gi|330443391|ref|NC_001133.9| Saccharomyces cerevisiae S288c chromosome
I, complete sequence_+_211639
AAAGGAACAGTATACTTTTTTGATAAAAAAATCTTG
>read_4_gi|330443391|ref|NC_001133.9| Saccharomyces cerevisiae S288c chromosome
I, complete sequence_+_140349
TTGATAGATCTGTTTGGTTCCTTACCGTTGAAGTAG
>read_5_gi|330443391|ref|NC_001133.9| Saccharomyces cerevisiae S288c chromosome
I, complete sequence_+_84539
TTGAACTTCTTTCATCAACTTTTTAGCTGGGGAAAA
>read_6_gi|330443391|ref|NC_001133.9| Saccharomyces cerevisiae S288c chromosome
I, complete sequence_+_115712
AAGTGGCGACGACTACACAGAATCCACGCCGCAGAT
>read_7_gi|330443391|ref|NC_001133.9| Saccharomyces cerevisiae S288c chromosome
I, complete sequence_-_49968
```

图 6.2基因测序数据的 FASTA 文件格式

当前基因组测序数据具有高通量、短序列、高覆盖度等特点。本章使用的数据主要是由 Solexa 测序仪产生的 FASTA 格式数据，具体来说本文选取 S.aureus，R.sphaeroides，human chromosome（Hg14），Fish，Yanhuang 五种规模大小不等基因组数据进行序列组装质量测试与组装软件性能测试。这些基因组数据中，S.aureus，R.sphaeroides，human chromosome 14(Hg14)从 GAGE 项目中下载[104]，Fish 测序数据从 Assemblathon 2[125]竞赛项目中下载， Yanhuang 基因组数据由华大基因提供。测试数据的相关信息参见表 6.1 所示。除此以外本文还使用由华大基因千人基因组官方数据库提供的千人基因组数据。

表 6.1 参与测试的物种基因数据信息

物种	S.aureus	R.sphaeroides	Hg14	Fish	Yanhuang
数据大小(GB)	0.684	0.906	14.2	425	495
序列长度(bp)	37,101	101	101	101	80-120
序列数（百万条）	4.8	4.1	62	1910	1859
覆盖度(倍)	90X	90X	70X	192X	57X
k-mer 数量（个）	31	31	31	31	31
数据源规模（MB）	2.90	4.60	88.6	1000	3000

为了对 SWAP-Assembler 序列组装软件的组装结果进行评估， 本文使用上述表 6-1 中的 5 个大小不同的基因组数据。其中，这些数据的主要特点如下：

（1）S.aureus：含有 480 万条 Illumina 测序仪产生的序列， 这个数据包括两个文库，一个文库的序列长度是 101bp，pair-end 的 insert 长度是 180bp；另一个文库的序列长度是 37bp， pair-end 的 insert 长度是 3.5kbp 。该数据也可在 NCBI 网站 SRA 数据库[14]中下载，下载编号是 no. SRS004751, no. SRS004752。

（2）R.sphaeroides：包含 410 万条 Illumina 测序仪产生的长度为 101bp 的序列， 该序列数据也包括两个文库，两个文库的 insert 长度分别为 180bp 和 3.5kbp。该数据也可以在 NCBI 网站的 SRA 数据库中下载，下载编号为 no. SRX033397。

（3）Hg14：人类基因组的第 14 条染色体测序数据，该数据共有 2650 万条长度为 101bp 的 Illumina 测序仪产生的序列，这些序列分布于三个文库，这三个文库的 insert 长度分别为 155bp, 2.8kbp, 35kbp。该数据在 NCBI 网站的 SRA 数据库中的下载编号为 no. SRP003680。

（4）Fish：M.zebra 一种可食用的石斑鱼，通过使用 Illumina 测序仪测序产生 19 亿条双端序列，该数据包括 17 个 insert 长度不等的文库。该数据由 Assemblathon 2 网站提供下载。

（5）Yanhuang： 炎黄基因组是由华大基因使用 Illumina 测序仪测序产生的 18.6 亿条序列。该数据可在华大基因网站下载，同时也可以在 EBI 官网 SRA 数据库下载，下载编号为 EMBL:ERP001652。

6.3 序列组装结果质量分析

6.3.1 序列组装软件介绍

本文选取 Velvet、 SOAPdenovo、ABySS、Ray 四个主流序列组装软件与 SWAP-Assember 序列组装软件从序列组装质量、精度以及软件拼接速度、内存消耗、可扩展性、负载均衡等角度进行测试分析与说明。其中，Velvet、SOAPdenovo 是单机共享内存的序列组装软件，ABySS、Ray 是可运行于高性能集群的并行序列组装软件。其中，软件版本相关介绍见表 6.2。

表 6.2 参与组装结果质量对比的四个主流序列组装软件

软件名称	版本号
Velvet	1.1.04
SOAPdenovo	1.05
ABySS	1.3.3
Ray	2.2.0

6.3.2 序列组装结果质量分析

为了使本文中的实验结果可与 GAGE 比赛的组装结果直接对比，本实验中使用了和 GAGE 相同的序列纠错流程，即采用 ALLPATH-LG 和 Quake 两个序列纠错工具。序列纠错可以将全基因组测序产生的各种测序错误后尽可能的剔除。

现有的序列纠错算法的基本原理首先使用 k-mer 频谱分析，对数据测序深度 Coverage 和 k-mer 剔除阀值进行估计，并将出现频率小于剔除阀值的 k-mer 进行剔除。通常，质量更高的序列数据可以组装出更长的 contig，例如，本文中 SWAP-Assembler 在使用了 GAGE 的序列纠错流程后，可以将组装 S.aureus 的序列数据产生的 contig 的 N50 长度从 11.6kbp 提高到 30.2kbp，将组装 R.sphaeroides 的序列数据产生的 contig 的 N50 长度从 2.1kbp 提高到 7.3kbp。因此，本文同样使用 ALLPATH-LG 和 Quake 两个序列纠错工具，并将这两个序列纠错工具纠正过的序列数据作为表 6.2 的四个组装软件和 SWAP-Assembler 的输入数据，同时，本文使用这两种纠正过的数据来进行组装结果对比与分析。

表 6.3 S.aureus 和 R.sphaeriodes 数据集的组装结果质量分析表

dataset	software	contigs			
		Number	N50(kb)	Error	N50[1](kb)

S.aureus	Velvet	162	48.4	28	41.5
	SOAP[2]	**107**	**288.2**	48	**62.7**
	ABySS	302	29.2	14	24.8
	Ray	221	36.6	15	35.6
	SWAP[3]	313	30.2	**7**	30.0
R.sph	Velvet	583	15.7	35	**14.5**
	SOAP[2]	**204**	**131.7**	414	14.3
	ABySS	1915	5.9	55	4.2
	Ray	752	11.5	17	11.2
	SWAP[3]	1054	7.3	**12**	7.3

[1] 这里的 N50 代表的是经过纠正后的 contig 的 N50 长度。

[2] SOAP 这里是 SOAPdenovo 的简写。

[3] SWAP 这里是 SWAP-Assembler 的简写。

在表 6.3 中， 本文先列出这五个参与比较的序列组装软件组装结果。表 6.3 中主要是在 contig 数量， contig N50 大小， 错误 contig 数量，以及纠正后的 contig 的 N50 大小。

本文，N50 的计算方法如下描述，首先将组装产生的 contig 按长短排序，然后当 50%的参考基因组序列累积被排序后的某个 contig 覆盖，这个 contig 的长度即为 **N50** 的长度，同理这里还有 N25, N75 分表示的是 25%或 75%的参考序列被覆盖时的 contig 长度。 当一个 contig 中错配、删除或插入的碱基数超过 5bp 时即可被断定为**错误的 contig**，这时通常可以在每个错配、删除或插入碱基数超过 5bp 的位置将 contig 切开为多个更短的 contig 后，通过重新计算这些修正过的 contig 的 N50 来得到纠正后的 contig 的 N50。

根据表 6.3 结果，本文可以看到 SOAPdenovo 所产生的 contig 的数量最少，N50 长度最长，但是它纠错后的 contig N50 长度下滑严重，这说明 SOAPdenovo 组装的 contig 中组装错误的 contig 比较多。最终对于 S.aureus 数据的 SOAPdenovo 纠错后的 contig N50 还是排名第一，但是在组装 R.sphaeriodes 数据时，组装结果不如 Velvet。相比于 Velvet、SOAP、ABySS、Ray 组装软件，SWAP-Assembler 在这五组数据中产生的错误 contig 最少，因此它在纠正前后的 contig N50 长度基本保持一致。在并行组装软件中（ABySS, Ray, SWAP-Assembler），Ray 的 N50 长度以及纠正后的 N50 长度始终保持第一，接着就是 SWAP-Assembler， ABySS 排在最后。

6.3.3 序列组装结果精度分析

组装结果的质量需要通过把所有的 contig 与基因组参考序列进行一对一的对比分析。因为只有 S.aureus 和 R.sphaeriodes 两个基因组有基因组参考序列，所以在表 6.4 和表 6-5 中对这两个基因组的对比分析的结果做了详细的统计。在表 6.4 中，本文选取了下面这五个精度评估的参数来分析组装结果，参数意义表述如下：

参考序列覆盖比例(Assembly size): 所有 contig 比对到基因组参考序列上所覆盖的碱基比例。

短 contig 覆盖比例(chaff size): 所有长度过短的 contig(<1kbp)比对到基因组参考序列上所覆盖的碱基比例。

参考序列未覆盖比例(unaligned ref size): 没有被 contig 比对上的参考序列中碱基比例。

contig 上的无效碱基比例（unaligned asm size）: 没有比对到参考序列的 contig 上的碱基比例。

contig 上重复比对的碱基比例（duplicated ref bases）: contigs 上重复比对到参考序列上的碱基占 contig 上的总的碱基的比例。

重复序列压缩比例（compress ref bases）: contigs 上重复比对到参考序列上的重复区的压缩比例。

在表 6.4 中， 由于 SWAP-Assembler 并没有激进的去解耦基因组本身的重复区域，而是使用了一些比较保守的解耦策略，例如错误纠正，错误过滤，tip 去除，分叉边解耦，虚分叉点解耦技术。因此，SWAP-Assembler 在 S.aureus 和 R.sphaeriodes 数据集上都保持着最小的参考序列的覆盖百分比（Assembly size），contig 上的无效碱基比例（unaligned asm size）以及 contig 上重复比对的碱基比例（duplicated ref bases）。 但是这个策略会导致产生很多碎小的 chaff contigs（即长度<1000bp 的 contig），因此也会使得相当大一部分的基因组参考序列无法被覆盖。与此相反，SOAPdenovo 的策略却非常激进，它会尝试去解耦重复区来扩展 contig 并覆盖尽可能多的基因组参考序列，但这样会带来更多的重复 contig 以及较高的**重复序列压缩比例。**

接着本文在表 6.5 中统计了组装结果的另外三个精度评估标准：单核苷酸多态性（SNPs），插入删除(indels)，错配(Misjoins)。这里本文用 contig 中存在的错误数量来评估组装软件的激进度，SOAPdenovo 因其尽全力去最大化 contig 的长度并容忍可能带来的错误，所以被认为是这五个组装软件中最为激进的序列组装软件，而 SWAP-Assember 恰恰相反，它在尽可能的保持 contig 的正确性的前提下去扩展 contig 的长度，因此含有最少的 SNPs，Indels 以及 Misjoins，可以被认为是这五个软件中最为保守的组装软件。在三个并行组装软件中，Ray 可以被认为是最为激进并行组装软件，而 SWAP-Assembler 排名第二位，最后 ABySS 虽然以较多的 SNPs, Indels 以及 Misjoins 为代价也未能赶超

Ray 的 contig N50，取得更长的 contig 结果。

根据以上的组装结果的质量和精度评估结果，本文建议如果用户需要获取尽可能长的 contig，那么建议这类用户使用 SOAPdenovo， 而对于那些更加注重精准的 contig，同时也希望 contig 比较长的用户，可以使用 SWAP-Assembler。

表 6.4 contig 中无法比对或重复比对到参考序列中碱基的统计情况（单位：百分比%）

software	Assembly size	Chaff Size	Unalign ref bases	Unalign asm bases	Duplicate ref bases	Compress ref bases
S.aureus(2.87Mb)						
Velvet	99.2	0.45	0.79	0.03	0.10	1.28
SOAP	101.3	0.35	**0.38**	0.01	1.44	1.41
ABySS	127.0	66.00	1.37	**<0.01**	23.30	0.99
Ray	98.4	**0.10**	0.88	0.04	0.08	1.26
SWAP	**97.3**	0.76	2.60	**<0.01**	**<0.01**	**0.33**
R.sphaeroides (4.60Mb)						
Velvet	97.8	0.54	1.60	0.01	0.29	0.92
SOAP	99.9	0.45	**0.88**	0.02	1.07	0.51
ABySS	108.0	1.65	3.01	0.15	10.04	**0.04**
Ray	99.0	**0.13**	1.03	**<0.01**	0.27	0.73
SWAP	**97.4**	1.01	2.17	**<0.01**	**0.03**	0.69

表 6.5 对 S.aureus 和 R.sphaeroides 数据组装结果中 Insertions, deletions, misassembler 等错误个数的统计结果

software	SNPs	contigs		contigs		
		<5bp	≥5bp	Misjoins	Inversion	Relocation
S.aureus (2.87Mb)						
Velvet	217	6	14	14	5	9
SOAP	246	25	31	17	**1**	16
ABySS	258	20	9	5	3	**2**
Ray	160	10	6	9	4	5
SWAP	**12**	**6**	**2**	**5**	2	3
R.sphaeroides (4.60Mb)						
Velvet	413	148	27	8	0	8
SOAP	527	155	406	8	0	8
ABySS	692	288	34	21	2	19

Ray	316	141	14	3	1	2
SWAP	**152**	**127**	**11**	**1**	**0**	**1**

表 6.6 对大型基因组 Hg14, Fish 和 Yanhuang 数据集的组装结果质量分析表

Dataset	Software	Contigs			
		Num	N50(bp)	Max(bp)	BaseinContig(Mbp)
Hg14 (88.3Mb)	Velvet	90,784	1688	25,729	**83.3**
	SOAPdenovo	200,153	836	21,144	96.4
	ABySS	190,693	1914	**26,697**	107.4
	Ray	**76,950**	964	14,399	68.4
	SWAP	88,609	**2036**	21,246	96.4
Fish (1 Gb)	Velvet	-	-	-	-
	SOAPdenovo	3291,290	378	7181	1,134.40
	ABySS	-	-	-	-
	Ray	-	-	-	-
	SWAP	**2881,443**	**1309**	**35,962**	**1,097.90**
Yanhuang (3 Gb)	Velvet	-	-	-	-
	SOAPdenovo	8,584,515	841	23,782	**3396.2**
	ABySS	9,218,967	1059	24,428	3691.8
	Ray	3,755,103	266	6,765	1620.1
	SWAP	**2,379,151**	**1368**	**31,152**	2434.3

接着本文对 Hg14, Fish 和 Yanhuang 三个大型基因组数据集进行组装，并将组装后的 contig 质量信息记录与表 6.6 中。由于 GAGE 的自动化评价脚本需要参考序列，而这三个数据没有标准的原始参考序列可以提供，所以这里本文写了一个自动化脚本用来自动统计组装软件输出的 contig 的数量，N50 大小， 最长的 contig 长度， 以及在所有 contig 中所包含的碱基总数。 这里本文还是使用了上面的五个组装软件，并且设置 k-mer 的长度为 31。

表 6.6 证实，在所有的三个无参考序列的基因数据的组装结果中，SWAP-Assembler 所产生的 contig N50 的长度最长。其中在 Fish 和炎黄基因组中，SWAP-Assembler 在 contig 的数量和最长的 contig 长度上也排名第一。然而在 Hg14 数据集中，contig 的数量在这五个组装软件中排名第二，contig 的最大长度排名第三，contig 中的碱基数量排名第二。但是 SWAP-Assembler 在所有的这三个数据集中的 N50 长度都是最长。这主要是因为

SWAP-Assembler 的图过滤步骤中有使用了高效的错误 k-mer 过滤策略，而其他的组装只是依靠其他测序错误纠正软件来过滤错误，而这些纠错软件可能会额外的将正确的碱基修改错误。此外 SWAP-Assembler 还具有 contig 扩展步骤，而这个步骤不仅可以将传统的 tips，鼓泡(bubble)，分叉顶点(cross node)进行解耦处理，同时还可以将虚分叉顶点(virtual cross node)进行解耦处理。

综上所述，SWAP-Assembler 组装软件对规模较小的两个基因组(S.aureus 和 R.sphaeriodes)的组装结果的 N50 长度在五个基因组组装软件中位于前列，组装精度最高，而对大规模的基因组数据（Hg14, Fish 和 Yanhuang）进行组装的结果在所有五个组装软件中，其 N50 长度最长， 组装结果最好。

6.4 序列组装软件性能测试

6.4.1SWAP-Assember 扩展性和系统效率测试

本小节将对 SWAP-Assembler 的扩展性和系统效率做性能测试。首先本文在天津国家超级计算中心的天河 1A 上对 Ray, ABySS, SWAP-Assembler 做千核规模的扩展性测试。

首先对 S.aureus，R.sphaeroides，human chromosome（Hg14），Fish，Yanhuang 基因数据使用并行组装软件进行强扩展性性能测试，测试结果见表 6.7。 从表 6.7 中可以看到，在天河 1A 超级计算机上使用 1024 核心时，SWAP-Assembler 在 S.aureus 数据集上比 ABySS 快 119 倍，比 Ray 快 73 倍。 在组装 R.sphaeroides 数据集时， ABySS 和 Ray 在超过 128 核后无法通过核数的提升实现性能提高，而 SWAP-Assembler 在达到 512 核时依然有性能提升。最终 ABySS,Ray 和 SWAP-Assembler 在组装 R.sphaeroides 数据集时分别可以扩展到 128，256，1024 核心。

以上的两个小数据集测试在使用更多的计算核心时因为额外的通讯时间增长等因素无法获得更高的性能提升。因此在表 6-7 中本文接着使用了三个更大的数据集合 Hg14, Fish 和炎黄基因组数据。在使用 1024 核心组装 Hg14 基因组数据时，SWAP-Assembler 的处理速度是 ABySS 的 280 倍和 Ray 的 38 倍。在这套数据上三个基因组维持着和 R.sphaeroides 数据集一样的扩展性，即 128 核心，256 核心，1024 核心。Fish 的数据集由于数据量大测序深度太深而使得 Ray 和 ABySS 无法在 48 小时内完成组装， 但 SWAP-Assembler 使用 1024 核心时在 53 分钟内将 Fish 基因组测序数据集组装完成。 当本文处理这组数据中最大的数据集炎黄基因组数据时，Ray 和 ABySS 仍然无法在 48 小时内完成组装，而 SWAP-Assembler 使用 1024 核心时在 29 分钟内将该基因组组装完毕。为了将 SWAP-Assembler 与其他单机的基因组组装软件进行比较，根据文献中提供的性能数据，SOAPdenovo1[30]和 SOAPdenovo2[8]分别可以在 48 小时和 72 小时完成炎黄基因组的组装。 因此在处理最大的两个基因组数据时（Fish 和 Yanhuang 数据集）， 在不

断的提高计算核心数时，SWAP-Assembler 在天河 1A 上扩展到 1024 核心。

表 6.7 SWAP-Assembler 在天河 1A 上强扩展性测试结果

Software	64	128	256	512	1024
S.aureus (2.87Mb)					
ABySS	248	269	334	554	831
Ray	244	198	202	266	510
SWAP	23	15	8	5	7
R.sphaeroides (4.60Mb)					
ABySS	492	454	522	718	1312
Ray	287	183	181	190	285
SWAP	43	29	15	7	7
Hg14 (88.29Mb)					
ABySS	6472	5299	6935	9045	16530
Ray	2926	1746	1288	1517	2266
SWAP	585	428	203	128	59
Fish (1Gb)					
ABySS	-	-	-	-	-
Ray	-	-	-	-	-
SWAP	59214	13941	8622	3263	3194
Yanhuang					
ABySS	153,060	150,685	150,747	153,030	161,581
Ray	-	-	-	-	-
SWAP	-	11243	5761	4021	1783

- 代表该组装软件在处理对应的基因组数据时内存溢出或者超时，在本测试中最长运行时间为 48 小时。

6.4.2 SWAP-Assembler 序列组装软件极限测试

接着为了测试 SWAP-Assembler 的扩展性和系统效率的极限，本文在更大规模的高性能集群上测试该软件优化版本 SWAP2，但这里由于机时限制，没有测试其他两个并行组装软件 Ray 和 ABySS。

首先本文使用千人基因组数据的一部分来对 SWAP-Assembler 及其优化版本做若扩展性测试。在本次测试中，千人基因组数据被随机的抽取出来做成若扩展性能测试数据集，随着计算核数从 1024 增加到 16，384 核，该数据集合的大小从 256G 增加到 4TB。图 6.3 和图 6.4 中可以看到 SWAP2 在扩展性上相比于 SWAP-Assembler 如下三个性能提

升：

扩展性：SWAP2 可以扩展到 16，384 核心，然而 SWAP-Assembler 在 Mira 上最多只能扩展到 4096 核心。图 6-4 显示出开图收缩的时间开销，SWAP2 在前面三步的时间开销随着核数的增加知识缓缓增加了一点。同时表 6-8 中可以确认这三个步骤的时间消耗的百分比基本保持稳定。因为千人基因组的基因组参考序列的大小是固定的只有 30 亿 bp 大小，所以在图过滤这一步后，这个双向多步图中的顶点规模基本在 30 亿个左右。所以图收缩这一步实际上是在问题规模确定的情况下，计算核心数在不断的加倍的一种强扩展测试，而时间统计结果显示这一步的时间消耗随着计算核心数的加倍而每次减半，这个结果也接近了强扩展测试的理想性能加速比。

表 6.8 SWAP2 使用 4TB 的千人基因组数据集做强扩展测试时各个步骤运行时间统计表[126]。这里每个计算节点最多分配 4 个进程，ppn=4。 时间是以秒为单位。

表 6.8 SWAP2 强扩展性测试运行时间统计表

计算进程数	并行 I/O	图构建	图过滤	图收缩	总时间
1,024	681	880.54	14.85	148.85	1725.24
2,048	1372.33	1268.96	61.33	991.24	3721.01
4,096	691.88	633.04	53.35	516.96	1906.96
8,192	346.23	328.91	26.61	264.11	972.55
16,384	207.15	184.86	13.2	132.76	541.46
32,768	114.26	107.16	6.6	66.63	297.38
65,536	56.2	70.76	3.28	31.34	165.01
131,072	51.53	64.71	1.63	15.84	138.39

图 6.3 SWAP-Assembler 在 Mira 超级计算机上使用千人基因组弱扩展测试数据集的性能测试结果图。这里 Mira 的每个计算节点被分配 4 个进程，即 ppn=4。

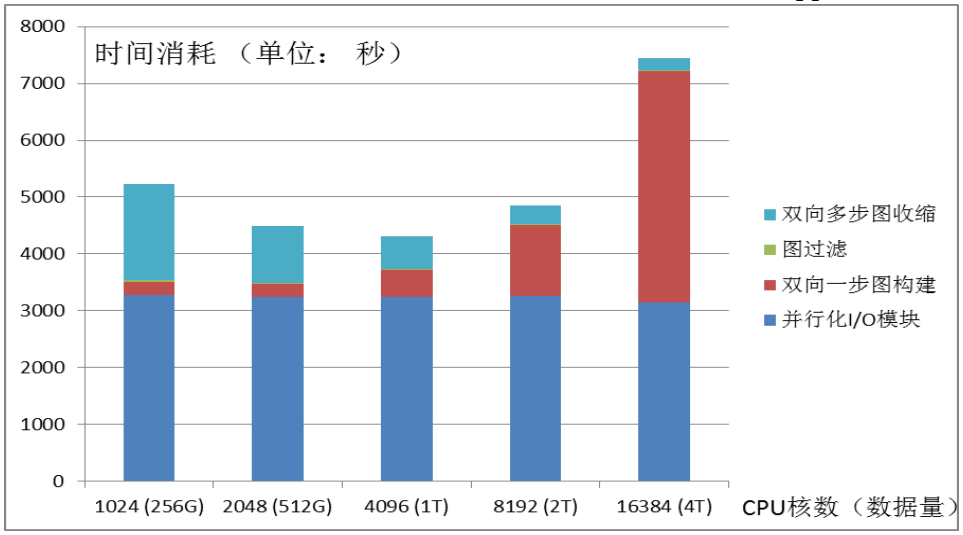

图 6.3 SWAP-Assembler 弱扩展性测试结果图

图 6.4 SWAP2 在 Mira 超级计算机上使用千人基因组弱扩展测试数据集的性能测试结果图。这里 Mira 的每个计算节点被分配 4 个进程，即 ppn=4。

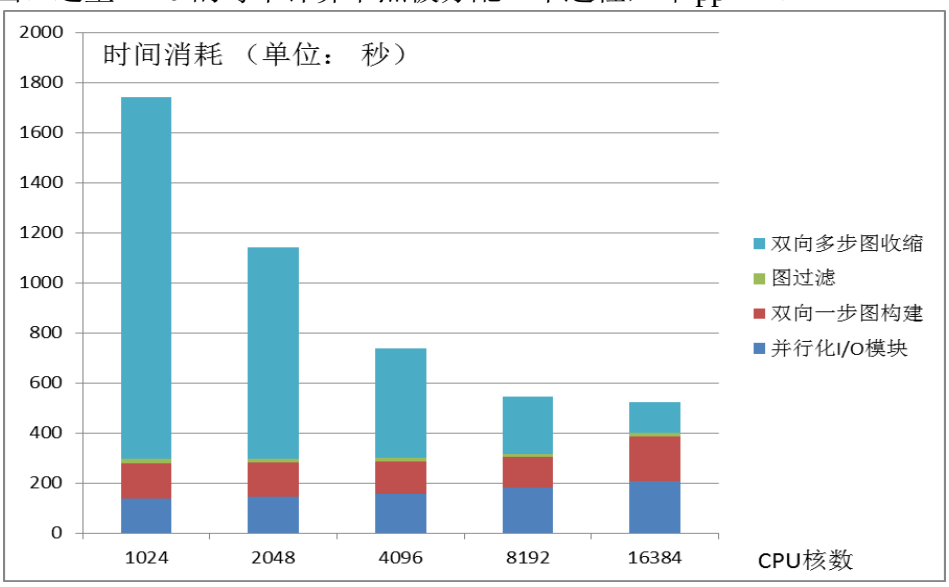

图6.4SWAP2弱扩展性能测试结果图

图 6.5 SWAP2 I/O 速度，通讯带宽，内存使用与系统理论峰值的百分比。这里 Mira 的峰值 I/O 速度是 32GB/rack/s，峰值通讯速度是 0.9GB/node/s[118, 119]， 单机内存最大是 16GB。因为每个计算节点被分配了 4 个进程，所以每个进程最大可使用内存是 4GB。

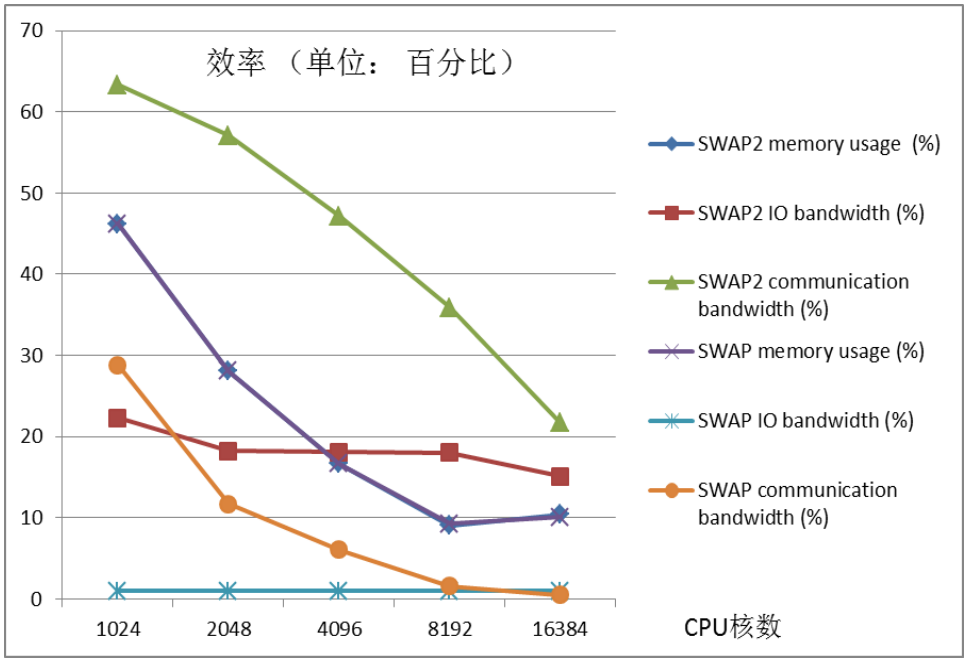

图 6 .5 SWAP2 I/O 速度，通讯带宽，内存使用与系统理论峰值的效果图

加速比：SWAP2 在时间消耗上面比优化前的 SWAP-Assembler 要少 10 多倍。具体来讲其中数据分块算法 FAA 和数据块大小调优策略使得 SWAP2 在并行 I/O 模块比 SWAP-Assembler 快 15 倍；在双向多步图构建模块中，通过采用固定消息载荷技术和数据缓冲池技术来将并行 I/O 和图构建模块通讯进行隔离，进而解决了通讯效率退化问题

并实现了 23 倍的加速。在图收缩模块中，通过消息压缩，通讯协议的优化等措施使得该模块比优化前快 1.75 倍。最后总的性能相比优化前加速了 14 倍。

系统效率：为了探索系统的持续优化的空间，本文计算了 SWAP2 的并行 I/O 速率，通讯带宽，内存使用率与系统峰值的百分比的比值。其中在 MIRA 中的一个机柜或者 4096 核规模时，SWAP2 的 I/O 带宽从优化前的 1%提高到 18%， 通讯带宽从优化前的 5%提高到 47%。 而内存利用率基本保持不变，但这一项后续工作需要持续优化以提高内存效率。

在对 SWAP2 进行强扩展性测试的实验中，本文使用了 4TB 的千人基因组数据，而参与计算的核数从 1024 核（或 256 计算节点）一直增长到 131，072 核心（或 16384 计算节点）。性能测试的运行结果收集于表 6.8 中，并将各步骤的耗时百分比和加速比绘制于图 6.6。在图 6.6 中本文可以看到当计算核数等比例增加时，SWAP2 的各个步骤的耗时百分比基本保持不变，所有的四个步骤都基本保持这同样的并行加速比，其中每个模块的具体耗时时间可参见表 6.8。 SWAP2 在使用 131，072 核心时需要大约 2 分钟处理完 4TB 炎黄基因组数据，而这套数据现今最大的用于组装的数据集。 最后 SWAP2 的加速比增长稳定，并在 131，072 核时达到了 8.8 的加速比，同时系统并行效率保持在 40%左右。

本文使用炎黄基因组数据(300GB)测试 SWAP2 的强扩展性，将性能结果收集于表 6.9 中，并与现今扩展度最高的基因组组装软件 HipMer 做性能比较分析。 在使用 16，384 核时，SWAP2 可以在 163 秒内在 Mira 上将炎黄基因组组装完毕。 因为 HipMer 还没有将源代码开源，所以本文直接将其已发表的论文中的数据拿出来和 SWAP2 的性能结果做比较，这里需要注意的是 HipMer 使用的高性能集群是美国国家能源研究科学计算机中心 NERSC 的 Cray Edison[127]。与 IBM Blue Gene Q-Mira 相比，Edison 使用的 Dragonfly 网络的速度是 Mira 的 7.8 倍，CPU 的频率比 Mira 高 1.5 倍，综合性能大约是 IBM Blue Gene Q-Mira 的 8 倍[128,129,130]。HipMer 在 Edison 上使用 15，000 核时，可以在 8 分钟内将人类基因组数据(290G)组装完毕。使用相同规模的核数时， SWAP2 组装同等规模的炎黄基因组数据(300G)仅需 163 秒，运行速度大约是 HipMer 的 3 倍数。不仅如此，SWAP2 可以继续扩展到 65，536 核并可以在 64 秒内将炎黄基因组组装完，这时 SWAP2 的系统并行效率达到 55%。 根据 HipMer 的优化原理[41]，HipMer 的扩展性主要还是依赖于 oracle 提供的并行 De Bruijn 图的分割算法及其传统 De Bruijn 图的组装策略实现，然而 SWAP2 实际上是采用了双向多步图策略的全并行组装算法，其中已经消除了计算过程中的数据依赖问题，因此 SWAP2 可以在处理大规模的基因组数据时保持更高的扩展性和系统效率。

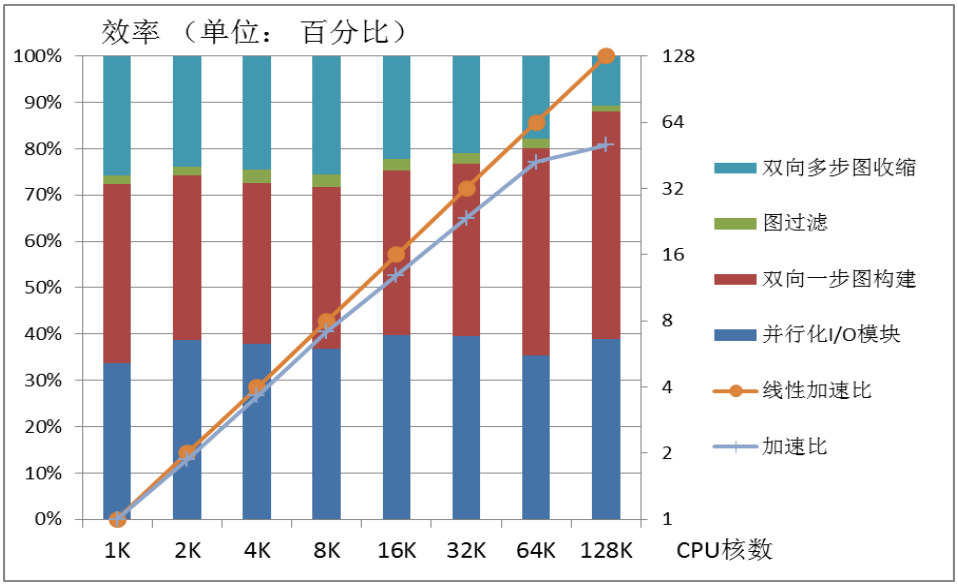

图 6.6 SWAP2 在组装 4TB 的千人基因组数据时各步骤的耗时百分比和加速比

表 6.9 SWAP2 使用 300GB 炎黄数据集做强扩展测试时各个步骤运行时间统计表。这里每个计算节点最多分配 4 个进程，ppn=4。 时间是以秒为单位。

表 6.9 SWAP2 强扩展性测试时间统计表

计算进程数	并行 I/O	图构建	图过滤	图收缩	总时间
1,024	117.55	281.44	19.08	1815.37	2266.72
2,048	59.81	140.56	9.63	930.74	1157.72
4,096	20.81	71.77	6.1	476.01	583.51
8,192	13.46	39.29	3.15	246.41	307.06
16,384	8.35	23.9	1.55	126.83	163.36
32,768	5.08	18.99	0.88	69.48	96.51
65,536	6.5	20.85	0.67	30.66	64.55

6.4 本章总结

基于 SWAP-Assembler 的组装结果的质量精度测试，以及扩展性加速比分析，实验结果表明，SWAP-Assembler 在组装现有最大的基因组数据-千人基因组数据时，可以扩展到 131，072 核心，并保持 40%的系统并行效率。 除此以外，SWAP-Assembler 的在组装炎黄基因组时，运行速度大约是现有最快的基因组装软件 HipMer 的 3 倍数，而且在使用 65，536 核心时，可以在 1 分钟内完成组装。在基因组组装质量上，相比 SOAPdenovo 组装软件的激进扩展 contig 长度的策略， SWAP-Assembler 的组装策略更偏向保守，所以其组装结果在首先保证 contig 精准度的基础上，再考虑扩展 contig 的长度。

最后，虽然 SWAP-Assembler 及其优化版本 SWAP2 仅在 Tianhe 1A 和 Mira 上做过

性能测试，而该系列软件实际上通过调优相关参数（数据块大小，通讯消息大小等）也可以用于在其他高性能集群上逼近系统峰值性能。SWAP-Assembler 和 SWAP2 可以在 Sourceforge 上免费下载使用：http://sourceforge.net/projects/swapassembler [131]。

第七章 结束语

7.1 本文工作总结

针对 TB 级大规模的高通量基因测序数据， 本文对大型基因组的高效可扩展的组装应用方面做了如下三个方面的研究：

1). 基因组高可扩展的并行组装的数学模型

本文结合基因组组装问题的传统基于 De Bruijn 图的策略，提出了用改进的双向多步图的数学模型来抽象基因组组装问题，并将基因组装问题等价的转化为一个半群系统上的一个可并发乱序执行的边合并操作。最终得到所有的全扩展双向多步边集合 E_S^* 中双向多步边上的标注序列集 L_S^* 得到最后的 contig。双向多步图的两个性质，直接证实上述该数学模型的与原始问题的等价性和正确性。其次该数学模型已经将边合并操作的计算依赖解耦，所以该数学模型相比于基于 De Bruijn 图的策略，更适合用于并行可扩展的基因组组装。

2). 高效并行的基因组装计算模型

为了高效处理双向多步图构造的半群系统上的边合并操作，本文提出了异步并行计算模型 SWAP，该模型可以最大化的提高半群系统上边合并操作的并行度。接着本文对该异步并行计算模型的具体实现算法，对其计算通讯复杂度抽象和分析，最后本文从理论上证明该异步计算模型具有高扩展性。基于该计算模型本文逐一阐述了并行序列组装软件 SWAP-Assembler 的每一步实现算法，同时最后通过理论证明该软件总的计算复杂度为 $O(g/p)$。而整个通讯过程的通讯轮数是 $O(\log(\log g($ ，进程间通讯量为 $O(g\log(\log g($ $)p/$。

3). 并行基因组装的深度系统优化

本文使用自适应的自动优化策略最大的逼近底层高性能计算平台及其文件系统的理论极限。在优化的过程中尽量使用体系结构无关的参数空间性能自动策略以达到优化策略的通用性。为了使计算、通讯、I/O 同步优化避免干扰，本文使用数据缓冲池进行速率缓冲匹配，为各步骤提供参数空间进行性能自动优化挖掘底层超算平台软硬件性能极限，同时支持计算通讯 I/O 的叠加(Overlap)技术。除此以外，本文为深度优化图算法底层库，使用并行图数据 streaming I/O 技术，内存数据压缩技术，消息通讯聚合压缩技术以提升现有 SWAP-Assembler 的问题规模，系统效率和扩展性。

最后基于 SWAP-Assembler 的组装结果的质量精度测试，以及扩展性加速比分析，实验结果表明 SWAP-Assembler 在组装现有最大的基因组数据-千人基因组数据时，可以扩展到 131，072 核心，并保持 40%的系统并行效率。 除此以外，SWAP-Assembler 的

在组装炎黄基因组时，运行速度大约是现有最快的基因组装软件 HipMer 的 3 倍数，而且在使用 65，536 核心时，可以在 1 分钟内完成组装。在基因组组装质量上，相比 SOAPdenovo 组装软件的激进扩展 contig 长度的策略， SWAP-Assembler 的组装策略更偏向保守，所以其组装结果在首先保证 contig 精准度的基础上，再考虑扩展 contig 的长度。

7.2 下一步研究方向

在本文研究工作的基础上，本文未来接着对下面几个问题开展进一步的研究：

1. TB 级大规模二三代混合基因数据的组装研究

随着第三代基因测序仪的出现，第三代测序仪产生的基因 read 更长，错误率更高，如何在更短时间内得到更高质量的基因组装结果对基因组装算法提出了更高的挑战。本文将利用第二代测序仪产生数据错误率低的特点，结合第二代和第三代测序技术开发新的基因组装算法。二三代测序数据混合组装是针对大型复杂基因组的参考序列获取问题，综合利用高通量测序技术中二代 reads 短小而准确并相对廉价和三代 reads 长而错误率较高并成本较高的特点，对二三代数据进行混合组装以获得更高质量的基因组参考序列。

本文结合二代测序相对低成本的特点和三代测序覆盖基因组更完整的特点，对目标基因组分别进行二三代测序，对其测序数据进行混合组装。混合组装策略主要是对两类数据之间的信息关系进行整合利用。本课题将使用成熟的混合组装策略，即从测序与基因组组装的历史得到启示，先用二代数据得到一个基因组框架草图，然后用二代数据的组装结果对三代数据进行纠错或者对三代组装结果进行局部抛光，最后再利用所有的三代数据对该草图二次组装并进行查漏补缺。同时本文充分考虑两种类型数据内部及其之间的相互关系，在组装算法的各步骤优化对不同数据的利用。例如，二代数据中 insert size 较大的 mate pair reads 与三代数据的长 reads 可以相互印证并对 contig 进行进一步的长度扩展进而生成高质量的组装结果。

2. 基于半群代数和 SWAP 计算模型的图算法应用支持

为了使得本文算法优化和可扩展性研究工作具有普适性，同时在系统层面达到高扩展和高效率的预期，区别于以往工作仅仅对个别算法，个别应用，在特地硬件系统上的定向优化，本文使用了基于半群代数系统的通用并行数学抽象模型以及基于该数学模型设计了图算法的表达方法。 为此本文用半群理论对常用图分析算法进行数学抽象如下：

给定一个顶点集合为 V, 边集合为 E 的有向图 G={V, E}，半群 SG(E, ×)被定义为一个运算符（算子）×在集合 E 上的半群，其中每个顶点 vi 都被一个运算符 xi 所关联，而顶点 vi 及其所有的邻居被组合在一起，并定义为运算符 xi 的小世界[xi]。当且仅当 xi 的小世界的所有的值都传送到 vi 时，xi 才能被计算。

本文通过重定义半群的算子来表达常用图分析算法，同时在底层实现上本文也提供

用户自定义的半群算子 API 来对用户自定义图算法进行数学表达和系统支持。为了便于将常用图分析算法进行数学抽象（包括主流图计算系统支持的图算法），表 7.1 列举了遍历类，查询类，迭代类，拓扑变化类 4 类图分析算法 4 算子表达方法。

表 7.1 图算法的分类和半群算子表达

类别	算法	应用领域	半群算子/算法表达
遍历类计算	BFS	Graph 500，图基本结构分析	标记扩散
	SSSP	图基本结构分析	距离扩散
查询类计算	图半径	图基本结构分析	距离扩散
	入/出度分布	图基本结构分析	统计扩散
	强/弱连通分量	图基本结构分析	可达性扩散
迭代类计算	Page Rank	搜索引擎	扩散迭代
	Personal PR	社交网络	扩散迭代
	SimRank	推荐系统	RandWalker
拓扑变化类	图收缩	基因组装	边收缩
	神经元网络	脑科学，认知网络	结构进化

参考文献

[1]. MD Graf, DF Needham, N Teed, T Brown, Genetic testing insurance coverage trends: a review of publicly available policies from the largest US payers. Personalized Medicine, 2013, 10(3): 235-243.

[2]. http://health.takungpao.com/q/2013/0826/1854216.html

[3]. JT Meng, BQ Wang, SZ Feng, P Balaji, Yanjie Wei, SWAP-Assembler: Scalable and Efficient Genome Assembly towards Thousands of Cores, BMC Bioinformatics, 2014，15(s9)

[4]. JT Meng, BQ Wang, YJ Wei, SZ Feng, P Balaji. SWAP-Assembler: Scalable and Efficient Genome Assembly towards Thousands of Cores, in 4th annual RECOMB satellite workshop on massively parallel sequencing (RECOMB-seq 2014), April, 2014.

[5]. J Gans, M Wolinsky, J Dunbar, Computational improvements reveal great bacterial diversity and high metal toxicity in soil. Science 2005, 309(5739): 1387-139.

[6]. JI Prosser, Dispersing misconceptions and identifying opportunities for the use of 'omics' in soil microbial ecology, Nature Reviews Microbiology, 2015, 13, 439–446

[7]. JA Gilbert, F Meyer, J Jansson, J Gordon, N Pace, J Tiedje, R Ley, N Fierer, D Field, N Kyrpides, FO Gl?ckner, HP Klenk, KE Wommack, E Glass, K Docherty,R Gallery, R Stevens, R Knight. The Earth microbiome project: Meeting report of the ' EMP meeting on sample selection and acquisition' at Argonne National Laboratory October 6 2010. Stand Genomic Sci, 2010, 3: 249-253.

[8]. R Luo, B Liu, Y Xie, Z Li, W Huang, J Yuan, G He, Y Chen, Q Pan, Y Liu, J Tang, G Wu, H Zhang, Y Shi, Y Liu, C Yu, B Wang, Y Lu, C Han, DW Cheung, SM Yiu, S Peng, Z Xiaoqian, G Liu, X Liao, Y Li, H Yang, J Wang, TW Lam, J Wang, SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler, Gigascience. 2012;1(1):18 [8] Margulies M, Egholm M, Altman W E, et al. Genome sequencing in microfabricated high-density picolitre reactors[J]. Nature, 2005, 437(7057): 376-380.

[9]. Compeau P E C, Pevzner P A, Tesler G. How to apply De Bruijn graphs to genome assembly[J]. Nature biotechnology, 2011, 29(11): 987-991.

[10]. Pop M, Phillippy A, Delcher A L, et al. Comparative genome assembly[J]. Briefings in bioinformatics, 2004, 5(3): 237-248.

[11]. Miller J R, Koren S, Sutton G. Assembly algorithms for next-generation sequencing data[J]. Genomics, 2010, 95(6): 315-327.

[12]. Caporaso J G, Lauber C L, Walters W A, et al. Ultra-high-throughput microbial community analysis on the Illumina HiSeq and MiSeq platforms[J]. The ISME journal, 2012, 6(8): 1621-1624.

[13]. European bioinformatics institute (EBI), http://www.ebi.ac.uk

[14]. National Center for Biotechnology Information(NCBI), http://www.ncbi.nlm.nih.gov

[15]. Coordinators N R. Database resources of the national center for biotechnology information[J]. Nucleic acids research, 2013, 41(Database issue): D8.

[16]. Joint Genome Institute(JGI), http://jgi.doe.gov

[17]. Pevzner PA, Tang H, Waterman MS. An eulerian approach to DNA fragment assembly[J]. Proc Natl Acad Aci USA 2001, 98(17): 9748-9753.

[18]. Mihai Pop, Steven L.Salzberg, Martin Sbumeay. Genome sequence assembly:algorithms and issues[J]. Genome Res, 2002,35(7):47-54.

[19]. Wenyu Zhang, Jiajia Chen, Yang Yang, et al. A practical comparison of de novo genome assembly software tools for next-generation sequencing technologes[J]. Plos One, 2011,6(3): e17915.

[20]. Ewing B and Green P. Base-calling of automated sequencer traces using Phred.II. error probabilities[J]. Genome Research, 1998, 8: 186-194.

[21]. Sutton GG, White O, Adams MD, et al. TIGR assembler: a new tool for assembling large shotgun sequencing projects[J]. Genome Sci Technol, 1995, 1:9-19.

[22]. Xiaoqiu Huang , Anup Madan. CAP3: A DNA sequence assembly program[J]. Genome Res, 1999, 9: 868-877.

[23]. Myers EW, Sutton GG, Delche AL,et al. A whole-genome assembly of Drosophila[J]. Science, 2000, 287: 2196-2204.

[24]. Batzoglou S, Jaffe OB, Stanley K, et al. ARACHNE: A whole-genome shotgun assembler[J]. Genome Res, 2002, 12: 177-189.

[25]. Jaffe DB, Butler J, Gnerre S, et al. Whole-genome sequence assembly for mammalian genomes: ARACHNE 2[J]. Genome Res, 2003, 13: 91-96.

[26]. James C, Zemin Ning. The Phusion assembler [J]. Genome Res, 2003, 13: 81-90.

[27]. Warren RL, Sutton GG, Jones SJM, et al. Assembling millions of short DNA sequences using SSAKE[J]. Bioinformatics, 2007, 23(4): 500-501.

[28]. Jeck WR, Reinhardt JA, Baltrus DA, et al. Extending assembly of short DNA sequences to handle error[J]. Bioinformatics, 2007, 23(21): 2942-2944.

[29]. Daniel R, Zerbino, Ewan Birney. Velvet: Algorithms for de novo short read assembly using De Bruijn graphs[J]. Genome Res, 2008, 18: 821-829.

[30]. Li R, Zhu H, Ruan J, et al. De novo assembly of human genomes with massively parallel short read sequencing[J]. Genome Res, 2010, 20(2): 265-272.

[31]. Yu Peng, Henry Leung, S.M.Yiu. IDBA-A practical iterative De Bruijn graph de novo assmbler[C]. Lecture Notes in Computer Science, 2010, 426-440.

[32]. Simpson JT, Wong K, Jackman SD, et al. ABYSS: A parallel assembler for short read

sequence data[J]. Genome Res, 2009, 19(6): 1117-1123.

[33]. Yongchao Liu, Bertil Schmidt, Douglas Maskell. Parallelized short read assembly of large genomes using De Bruijn graphs [J]. BMC Bioinformatics, 2011, 12: 354.

[34]. Marx, Vivien, Next-generation sequencing: The genome jigsaw, Nature 501 (7466): 263–268. doi:10.1038/501261a.

[35]. S. Boisvert, F. Raymond, E. Godzaridis, F. Laviolette, J. Corbeil et al.,Ray Meta: scalable de novo metagenome assembly and profiling,Genome Biol, vol. 13, no. 12, p. R122, 2012.

[36]. E. Godzaridis, S. Boisvert, F. Xia, M. Kandel, S. Behling, B. Long, C. P. Sosa, F. Laviolette, and J. Corbeil, Human analysts at superhuman scales.

[37]. B. G. Jackson and S. Aluru, Parallel construction of bidirected string graphs for genome assembly, in Parallel Processing, 2008. ICPP'08. 37th International Conference on. IEEE, 2008, pp. 346–353.

[38]. B. G. Jackson, P. S. Schnable, and S. Aluru, Parallel short sequence assembly of transcriptomes, BMC bioinformatics, vol. 10, no. Suppl 1, p. S14, 2009.

[39]. B. G. Jackson, M. Regennitter, X. Yang, P. S. Schnable, and S. Aluru, Parallel de novo assembly of large genomes from high-throughput short reads, in IEEE International Symposium on Parallel and Distributed Processing. IEEE, 2010, pp. 1–10.

[40]. E. Georganas, A. Buluc, J. Chapman, L. Oliker, D. Rokhsar, and K. Yelick, Parallel De Bruijn graph construction and traversal for de novo genome assembly, in International Conference for High Performance Computing, Networking, Storage and Analysis, SC14. IEEE, 2014, pp. 437–448.

[41]. E. Georganas, A. Buluc, J. Chapman, S. Hofmeyr, C. Aluru, R. Egan,L. Oliker, D. Rokhsar, and K. Yelick, HipMer: an extreme-scale de novo genome assembler,in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. ACM, 2015, p. 14.

[42]. D. Gregor, A. Lumsdaine, The Parallel BGL: A Generic Library for Distributed Graph Computations, in Proc. of Parallel Object-Oriented Scientific Computing (POOSC), July 2005.

[43]. D. Gregor, A. Lumsdaine, Lifting Sequential Graph Algorithms for Distributed-Memory Parallel Computation. In Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA'05), October 2005, pp. 423-437.

[44]. G. Malewicz, M.H. Austern, Aart J. C. Bik, J.C. Dehnert, I. Horn, N. Leiser, G. Czajkowski, Pregel: a system for large-scale graph processing, In SIGMOD'10 Proceedings of the 2010 international conference on Management of data, 2010, pp. 135-146, New York.

[45]. Apache giraph project. http://giraph.apache.org

[46]. Apache Hama. http://incubator.apache.org/hama/

[47]. S. Salihoglu. GPS: Graph processing system. http://infolab.stanford.edu/gps.

[48]. P. Pevzner, Computational molecular biology: an algorithmic approach. MIT press, 2000.

[49]. N. Siva, "1000 Genomes project," Nature biotechnology, vol. 26, no. 3, pp. 256–256, 2008.

[50]. Data provided by the 1000 genomes project. [Online]. Available: ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/data

[51]. Brief introduction graph 500. [Online]. Available: www.graph500.org

[52]. Y. Peng, H. C. Leung, S.-M. Yiu, and F. Y. Chin, "Idba–a practical iterative De Bruijn graph de novo assembler," in Research in Computational Molecular Biology. Springer, 2010, pp. 426–440.

[53]. J. Meng, J. Yuan, J. Cheng, Y. Wei, and S. Feng, "Small world asynchronous parallel model for genome assembly," in Network and Parallel Computing. Springer, 2012, pp. 145–155.

[54]. van Dijk E L, Auger H, Jaszczyszyn Y, et al. Ten years of next-generation sequencing technology[J]. Trends in genetics, 2014, 30(9): 418-426.

[55]. McCarthy A. Third generation DNA sequencing: pacific biosciences' single molecule real time technology[J]. Chemistry & biology, 2010, 17(7): 675-676.

[56]. Staden R. A strategy of DNA sequencing employing computer programs[J]. Nucleic acids research, 1979, 6(7): 2601-2610.

[57]. Lander E S, Waterman M S. Genomic mapping by fingerprinting random clones: a mathematical analysis[J]. Genomics, 1988, 2(3): 231-239.

[58]. Maier D. The complexity of some problems on subsequences and supersequences[J]. Journal of the ACM (JACM), 1978, 25(2): 322-336.

[59]. Kececioglu J D, Myers E W. Combinatorial algorithms for DNA sequence assembly[J]. Algorithmica, 1995, 13(1-2): 7-51.

[60]. Medvedev P, Georgiou K, Myers G, et al. Computability of models for sequence assembly[M]//Algorithms in Bioinformatics. Springer Berlin Heidelberg, 2007: 289-301.

[61]. Green P. Against a whole-genome shotgun[J]. Genome Research, 1997, 7(5): 410-417.

[62]. Myers E W, Sutton G G, Delcher A L, et al. A whole-genome assembly of Drosophila[J]. Science, 2000, 287(5461): 2196-2204.

[63]. Nagarajan N, Pop M. Parametric complexity of sequence assembly: theory and applications to next generation sequencing[J]. Journal of computational biology, 2009, 16(7): 897-908.

[64]. Kingsford C, Schatz M C, Pop M. Assembly complexity of prokaryotic genomes using

short reads[J]. BMC bioinformatics, 2010, 11(1): 1.

[65]. Bastide M, McCombie W R. Assembling genomic DNA sequences with PHRAP[J]. Current Protocols in Bioinformatics, 2007: 11.4. 1-11.4. 15.

[66]. Sutton G G, White O, Adams M D, et al. TIGR Assembler: A new tool for assembling large shotgun sequencing projects[J]. Genome Science and Technology, 1995, 1(1): 9-19.

[67]. Huang X, Madan A. CAP3: A DNA sequence assembly program[J]. Genome research, 1999, 9(9): 868-877.

[68]. Jeck W R, Reinhardt J A, Baltrus D A, et al. Extending assembly of short DNA sequences to handle error[J]. Bioinformatics, 2007, 23(21): 2942-2944.

[69]. Warren R L, Sutton G G, Jones S J M, et al. Assembling millions of short DNA sequences using SSAKE[J]. Bioinformatics, 2007, 23(4): 500-501.

[70]. Myers E W. Toward simplifying and accurately formulating fragment assembly[J]. Journal of Computational Biology, 1995, 2(2): 275-290.

[71]. Myers E W. The fragment assembly string graph[J]. Bioinformatics, 2005, 21(suppl 2): ii79-ii85.

[72]. Myers E W, Sutton G G, Delcher A L, et al. A whole-genome assembly of Drosophila[J]. Science, 2000, 287(5461): 2196-2204.

[73]. Ning Z, Cox A J, Mullikin J C. SSAHA: a fast search method for large DNA databases[J]. Genome research, 2001, 11(10): 1725-1729.

[74]. Voskoboynik A, Neff N F, Sahoo D, et al. The genome sequence of the colonial chordate, Botryllus schlosseri[J]. Elife, 2013, 2: e00569.

[75]. Pevzner P A. 1-Tuple DNA sequencing: computer analysis[J]. Journal of Biomolecular structure and dynamics, 1989, 7(1): 63-73.

[76]. Idury R M, Waterman M S. A new algorithm for DNA sequence assembly[J]. Journal of computational biology, 1995, 2(2): 291-306.

[77]. Pevzner P A, Tang H, Waterman M S. An Eulerian path approach to DNA fragment assembly[J]. Proceedings of the National Academy of Sciences, 2001, 98(17): 9748-9753.

[78]. Pevzner P A, Tang H. Fragment assembly with double-barreled data[J]. Bioinformatics, 2001, 17(suppl 1): S225-S233.

[79]. Kingsford C, Schatz M C, Pop M. Assembly complexity of prokaryotic genomes using short reads[J]. BMC bioinformatics, 2010, 11(1): 1.

[80]. Chaisson M J, Pevzner P A. Short read fragment assembly of bacterial genomes[J]. Genome research, 2008, 18(2): 324-330.

[81]. Conway T C, Bromage A J. Succinct data structures for assembling large genomes[J]. Bioinformatics, 2011, 27(4): 479-486.

[82]. Okanohara D, Sadakane K. Practical entropy-compressed rank/select

dictionary[C]//Proceedings of the Meeting on Algorithm Engineering & Expermiments. Society for Industrial and Applied Mathematics, 2007: 60-70.

[83]. Bloom B H. Space/time trade-offs in hash coding with allowable errors[J]. Communications of the ACM, 1970, 13(7): 422-426.

[84]. Melsted P, Pritchard J K. Efficient counting of k-mers in DNA sequences using a bloom filter[J]. BMC bioinformatics, 2011, 12(1): 1.

[85]. Pell J, Hintze A, Canino-Koning R, et al. Scaling metagenome sequence assembly with probabilistic De Bruijn graphs[J]. Proceedings of the National Academy of Sciences, 2012, 109(33): 13272-13277.

[86]. Chikhi R, Rizk G. Space-efficient and exact De Bruijn graph representation based on a Bloom filter[J]. Algorithms for Molecular Biology, 2013, 8(1): 1.

[87]. Ferragina P, Manzini G. Opportunistic data structures with applications[C]//Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on. IEEE, 2000: 390-398.

[88]. Langmead B, Trapnell C, Pop M, et al. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome[J]. Genome biol, 2009, 10(3): R25.

[89]. Li H, Durbin R. Fast and accurate short read alignment with Burrows–Wheeler transform[J]. Bioinformatics, 2009, 25(14): 1754-1760.

[90]. Li R, Yu C, Li Y, et al. SOAP2: an improved ultrafast tool for short read alignment[J]. Bioinformatics, 2009, 25(15): 1966-1967.

[91]. Lippert R A. Space-efficient whole genome comparisons with Burrows-Wheeler transforms[J]. Journal of computational biology, 2005, 12(4): 407-415.

[92]. Simpson J T. Exploring genome characteristics and sequence quality without a reference[J]. Bioinformatics, 2014, 30(9): 1228-1235.

[93]. Chikhi R, Rizk G. Space-efficient and exact De Bruijn graph representation based on a Bloom filter[J]. Algorithms for Molecular Biology, 2013, 8(1): 1.

[94]. Rødland E A. Compact representation of k-mer De Bruijn graphs for genome read assembly[J]. BMC bioinformatics, 2013, 14(1): 313.

[95]. Ye C, Ma Z S, Cannon C H, et al. Exploiting sparseness in de novo genome assembly[J]. BMC bioinformatics, 2012, 13(6): 1.

[96]. Myers E W. The fragment assembly string graph[J]. Bioinformatics, 2005, 21(suppl 2): ii79-ii85.

[97]. [98] Hernandez D, François P, Farinelli L, et al. De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer[J]. Genome research, 2008, 18(5): 802-809.

[98]. [99] Simpson J T, Durbin R. Efficient construction of an assembly string graph using the FM-index[J]. Bioinformatics, 2010, 26(12): i367-i373.

[99]. Ben-Bassat I, Chor B. String graph construction using incremental hashing[J]. Bioinformatics, 2014, 30(24): 3515-3523.

[100]. Dinh H, Rajasekaran S. A memory-efficient data structure representing exact-match overlap graphs with application for next-generation DNA assembly[J]. Bioinformatics, 2011, 27(14): 1901-1907.

[101]. Gonnella G, Kurtz S. Readjoiner: a fast and memory efficient string graph-based sequence assembler[J]. BMC bioinformatics, 2012, 13(1): 82.

[102]. Boisvert S, Laviolette F, Corbeil J. Ray: simultaneous assembly of reads from a mix of high-throughput sequencing technologies[J]. Journal of Computational Biology, 2010, 17(11): 1519-1533.

[103]. Draper J M, Culler D E, Yelick K, et al. Introduction to UPC and language specification[M]. Center for Computing Sciences, Institute for Defense Analyses, 1999.

[104]. Salzberg S L, Phillippy A M, Zimin A, et al. GAGE: A critical evaluation of genome assemblies and assembly algorithms[J]. Genome research, 2012, 22(3): 557-567.

[105]. Graph 500, www.graph500.org

[106]. http://www.rfc-archive.org/getrfc.php?rfc=5418

[107]. Garg P, Doshi R, Greene R. Using IEEE 802.11e MAC for QoS over wireless[J]. International Performance, Computing, and Communications Conference, 2003.

[108]. Ziouva E, Antonakopoulos T. CSMA/CA performance under high traffic conditions: throughput and delay analysis[J]. Computer Communications, 2002, 25(0): 313-321.

[109]. Gvaliant L. A Bridging Model for Multi-core Computing[J]. European Symposium on Algorithms, 2008.

[110]. Krizanc D, Saarimaki A. Bulk synchronous parallel: practical experience with a model for parallel computing[J]. Parallel Computing, 1999, 25(2): 159-181.

[111]. culler D, Mkarp R, Apatterson D. LogP: towards a realistic model of parallel computation[J]., 1993, 28(7): 1-12.

[112]. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Operating Systems Design and Implementation, 2004.

[113]. Karloff H, Suri S, Vassilvitskii S. A model of computation for MapReduce[J]., 2010.

[114]. K. Kumaran, “Introduction to Mira,” in Code for Q Workshop.

[115]. J. Milano, P. Lembke et al., IBM system Blue Gene solution: Blue Gene/Q hardware overview and installation planning. IBM Redbooks, 2013.

[116]. Mira - IBM BG/Q supercomputer machine overview. [Online]. Available: https://www.alcf.anl.gov/user-guides/machine-overview

[117]. Blue gene/q overview and update. [Online]. Available: https://www. alcf.anl.gov/files/IBMn BGQn Architecturen 0.pdf

[118]. D. Chen, N. A. Eisley, P. Heidelberger, R. M. Senger, Y. Sugawara, S. Kumar, V. Salapura, D. L. Satterfield, B. Steinmacher-Burow, and J. J. Parker, “The IBM Blue Gene/Q interconnection network and message unit,” in High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for. IEEE, 2011, pp. 1–10.

[119]. D. Chen, N. Eisley, P. Heidelberger, S. Kumar, A. Mamidala, F. Petrini, R. Senger, Y. Sugawara, R. Walkup, B. Steinmacher-Burow et al., “Looking under the hood of the IBM Blue Gene/Q network,” in Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. IEEE Computer Society Press, 2012, p. 69.

[120]. Xue-Jun Yang, Senior Member, CCF, Member, ACM, IEEE, Xiang-Ke Liao, Senior Member CCF, Member, ACM, Kai Lu .The TianHe-1A Supercomputer: Its Hardware and Software[J] Journal of Computer Science and Technology, 2011,V26(3): 344-351

[121]. Xiang-Ke Liao, Zheng-Bin Pang, Ke-Fei Wang, Yu-Tong Lu, Min Xie, Jun Xia, De-Zun Dong, Guang Suo.High Performance Interconnect Network for Tianhe System[J] Journal of Computer Science and Technology, 2015,V30(2): 259-272

[122]. Xiang-Ke Liao, Can-Qun Yang, Tao Tang Hui-Zhan Yi, Feng Wang, Qiang Wu, and Jingling Xue.OpenMC：Towards Simplifying Programming for TianHe Supercomputers[J] Journal of Computer Science and Technology, 2014,V29(3): 532-546

[123]. Top 500 supercomputers: http://www.top500.org

[124]. Text Format of the Genome sequencing data, http://hannonlab.cshl.edu/fastx_toolkit/

[125]. Bradnam K, Fass J, Valexandrov A. Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species [J]. Gigascience, 2013, 2(1).

[126]. Dataset provided by the 1000 Genomes project. [Online]. Available: ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/data

[127]. B. Alverson, E. Froese, L. Kaplan, and D. Roweth, “Cray XC series network,” Cray Inc., White Paper WP-Aries01-1112, 2012.

[128]. M. J. Cordery, B. Austin, H. Wassermann, C. S. Daley, N. J. Wright, S. D. Hammond, and D. Doerfler, “Analysis of Cray XC30 performance using Trinity-NERSC-8 benchmarks and comparison with Cray XE6 and IBM BG/Q,” in High Performance Computing Systems. Performance Modeling, Benchmarking and Simulation. Springer, 2013, pp. 52–72.

[129]. Sivalingam K., Lister G., and Lawrence B., "Performance analysis and Optimisation of the Met Unified Model on a Cray XC30." (2015).

[130]. Pgaudin W, Cmallinson A, Jperks O. Optimising Hydrodynamics applications for the Cray XC30 with the application tool suite[J]., 2014

[131]. SWAP2 download site. [Online]. Available: http://sourceforge.net/ projects/swapassembler

附录 1 攻读博士学位期间取得的学术成果

国际期刊会议

[1]. **Jintao Meng**, Sangmin Seo, Pavan Balaji, Yanjie Wei, Bingqiang Wang, Shengzhong Feng, SWAP-Assembler 2: Optimization of De Novo Genome Assembler at Extreme Scale, in Proceeding of the 45th International Conference on Parallel Processing (**ICPP 2016**), Philadelphia, PA. (Accept rate: 18%)

[2]. **Jintao Meng**, Sangmin Seo, Pavan Balaji, Yanjie Wei, Bingqiang Wang, Shengzhong Feng, SWAP-Assembler 2: Scalable Genome Assembler towards Millions of Cores - Practice and Experience, in ATIP workshop in Supercomputing 2015 (**Supercomputing 2015**), Austin.

[3]. **Jintao Meng**, Sangmin Seo, Pavan Balaji, Yanjie Wei, Bingqiang Wang, Shengzhong Feng, SWAP-Assembler 2: Scalable Genome Assembler towards Millions of Cores - Practice and Experience, in 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (**CCGrid'15**)-Doctoral Symposium, May, 2015.

[4]. **Jintao Meng**, Guixin Guo, Bingqiang Wang, Yanjie Wei, Jiefeng Cheng, Shenzhong Feng, An Ultra-fast Memory Efficient Parallel List Ranking Algorithm for BSP Based Graph Processing Systems, **HPC-China 2014**, Guangzhou.

[5]. **Jintao Meng**, Bingqiang Wang, Yanjie Wei, Shengzhong Feng, Pavan Balaji. SWAP-Assembler: Scalable and Efficient Genome Assembly towards Thousands of Cores, **BMC Bioinformatics,** Vol. 15 Supplement 9. (SCI, IF=2.8)

[6]. **Jintao Meng**, Bingqiang Wang, Yanjie Wei, Shengzhong Feng, Pavan Balaji. SWAP-Assembler: Scalable and Efficient Genome Assembly towards Thousands of Cores, in 4th annual RECOMB satellite workshop on massively parallel sequencing (**RECOMB-seq 2014**), April, 2014.

[7]. Li Zeng, Jiefeng Cheng, **Jintao Meng,** Bingqiang Wang, Shengzhong Feng. Improved Parallel Processing of Massive De Bruijn Graph for Genome Assembly, in 15th Asia-Pacific web conference (**APWeb 2013**), April, 2013, Sydney, Australia.

[8]. **Jintao Meng**, Jianrui Yuan, Shengzhong Feng, Yanjie Wei. An Energy Efficient Clustering Scheme for Data Aggregation in Wireless Sensor Networks, **Journal of Computer Science and Technology**, May 2013,V28(3): 564-573 (SCI, IF = 0.656)

[9]. **Jintao Meng**, Jianrui Yuan, Yanjie Wei, Jiefeng Cheng, Shengzhong Feng, Small World Asynchronous Parallel Model for Genome Assembly, in 9th IFIP International Conference on Network and Parallel Computing (**NPC 2012**), Sep.6, Gwangju, Korea (EI)

[10]. **Jintao Meng**, Jianrui Yuan, Yanjie Wei, Jiefeng Cheng, Shengzhong Feng, DGraph: Algorithms for Shortgun Reads Assembly Using De Bruijn Graph, in 9th IFIP International Conference on Network and Parallel Computing (**NPC 2012**), Sep.6, Gwangju, Korea (EI).

[11]. **Jintao Meng**, Jianrui Yuan, Shengzhong Feng, Liansheng Tan, Power Adjusting Algorithm: A New Cross-Layer Power Saving Mechanism for Mobile Ad-Hoc Networks. **Journal of Computer Science and Technology**, Jan 2013,V28(1): 42-53 (SCI, IF = 0.656)

[12]. Jianrui Yuan, **Jintao Meng**. A Power Adjusting Algorithm on Mobility Control in Mobile Ad Hoc Networks. In 8th IFIP International Conference on Network and Parallel Computing (**NPC 2011**), pp. 214-231, Changsha, China, Oct. 2011 (Best Paper candidate, EI)

[13]. Liansheng Tan, **Jintao Meng**, Jie Li and Han-Chieh Chao. PH-MAC: A Periodically Hybrid MAC Protocol for Wireless sensor networks, **Journal of Internet Technology**, Taiwan, Nov 2007. (SCI, IF = 0.508)

[14]. Jian Zhao, Danni Song, Qiang Zhang, **Jintao Meng**. A New Distributed Optimal Utility Max-Min Fair Resource Allocation. IEEE International Conference on Networking, Sensing and Control (**ICNSC 2008**), 2008.

国内期刊

[15]. **Jintao Meng,** Yanjie Wei, Gene and Disease Association Study with Metagenomics Assembling Tool: SWAP-Meta, e-Science Technology & Application, 2014, V5(3):30-37 (in chinese)

[16]. **Jintao Meng**, Jianrui Yuan, Yanjie Wei, Shengzhong Feng, The Analysis of De Novo Genome Assembly Software Based on De Bruijn Graph, e-Science Technology & Application, 2013, V4(5): 58-70.(in chinese)

[17]. **Jintao Meng**, Shuang Wang, Shengzhong Feng, An Introducton on Erroneous Kmers Filtering in Sequencing data, Bulletin of Advanced Technology Research, 2011, V1: 29-31. (in chinese)

附录 2 攻读博士学位期间参加的主要科研项目

图计算相关项目：

[1]. 国自然青年基金，大规模图分析算法优化与可扩展性研究， 2017/01-2019/12， 正在申请，负责人。

[2]. 科技部重点研发计划, E 级高性能应用软件编程框架研制及应用示范-JGraph 子项目（大规模图并行计算框架研发），2015/06-2018/06，正在申请，核心参与

[3]. 科技部 863 重大专项, 2014AA01A302, 医学大数据-子项目（大规模图并行算法研究），2015/06-2018/06，在研，核心参与

[4]. 深圳市基础研究，GJHS20120702091434440，并行海量数据挖掘关键技术及其应用，2013/01-2015/12，结题，参与

[5]. 国家自然科学基金， 百万规模不确定图数据的快速查询技术研究，2013/01-2015/12，结题，参与

生物信息相关项目：

[6]. 广东省中国科学院全面战略合作，2009A091100017，高通量基因测序技术及其应用，2009/02-2011/06，结题，参与

[7]. 广东省教育部产学研，2009B090200044，海量数据处理及生物信息系统的开发应用、2009/02-2011/11，结题，参与

[8]. 国家发展改革委员会， 基于 PB 级物基因数据处理的国民健康服务平台， 2014 年 2 月-2017 年 2 月，在研，参与

高性能计算相关项目：

[9]. 科技部 863 重大专项, 2015AA020109, 高性能计算环境应用服务优化关键技术研究-子项目（新药社区和动漫社区），2015/06-2018/06，在研，核心参与

[10]. 中科院先导专项， 鲲鹏大数据系统-子项目（图并行算法研究）， 2012 年 6 月~2016 年 6 月，在研，核心参与

[11]. 中科院先导专项，海云大数据系统研究及系统研发，2014 年-2017 年，在研，参与

附录 3 攻读博士学位期间获奖情况

申请的专利:

[1]. 孟金涛，张慧琳，彭丰斌，魏彦杰，冯圣中，双向多步 De Bruijn 图的错误双向边识别与去除方法，中国，CN201310672170.X。

[2]. 孟金涛，张慧琳，彭丰斌，魏彦杰，冯圣中，基于多步双向 De Bruijn 图的变长 kmer 查询的顶点扩展方法，中国，CN201310670752.4。

[3]. 孟金涛，张慧琳，彭丰斌，魏彦杰，冯圣中，基于多步双向 De Bruijn 图的变长 kmer 查询的双向边扩展方法，中国，CN201310670740.1。

[4]. 孟金涛，张慧琳，彭丰斌，魏彦杰，冯圣中，双向多步 De Bruijn 图的自环双向边识别与去除方法，中国，CN201310672187.5。

[5]. 孟金涛，张慧琳，彭丰斌，魏彦杰，冯圣中，双向多步 De Bruijn 图的突出端识别与去除方法，中国，CN201310672168.2。

[6]. 孟金涛，张慧琳，彭丰斌，魏彦杰，冯圣中，双向多步 De Bruijn 图的重复双向边识别与去除方法，中国，CN201310670440.3。

[7]. 孟金涛，魏延杰，成杰峰，冯圣中，基因测序数据读取方法及系统，中国，CN201210592061.2。

[8]. 孟金涛，魏彦杰，成杰峰，冯圣中，双向多步 deBruijn 图的压缩存储和构造方法，中国，CN201210587059.6。

[9]. 孟金涛，魏彦杰，曾理，成杰峰，冯圣中，短序列组装中序列片段的过滤方法及系统，中国，CN201210575726.9。

主要奖励:

[1]. 2015 年被评为中国科学院深圳先进技术研究院“优秀员工”

[2]. 2013 年被评为中国科学院深圳先进技术研究院“优秀员工”

[3]. 2012 年被评为中国科学院深圳先进技术研究院“优秀员工”

[4]. 2010 年被评为中国科学院深圳先进技术研究院“优秀员工”

[5]. 盛司潼，冯圣中，邵志峰，金虹，石宝晨，刘涛，朱定局，孟金涛，李花，冯玮樱，张雪琴，刘晋，李文波，冯细华，深圳市科学技术研究成果登记证书，中国,2012/10/19

致　谢

值此论文完成之际，我首先要感谢我的导师冯圣中研究员。 他给我创造了一个宽松的科研环境，并给了我非常大的研究自由度和学术发展空间。 冯老师从读博开始就给我设置了较高的研究目标，支持并指导我在华大基因研究院，香港大学计算机系，阿贡国家实验室，华为诺亚方舟实验室等机构的实习和学术交流。冯老师快速的学术问题定位，活跃的学术思想，宽广的学术胸襟都给我树立了良好的榜样。在我博士求学期间，他对学生的谆谆教导，言传身教，并在工作和生活中给予了很多无私的帮助，能够师从这样的导师是我莫大的荣幸，以后的日子学生将以老师为榜样，平和待人，原则处事，在学术研究上继续前行。

感谢赵晓芳研究员和唐宏伟博士，在博士就读期间对我工作的理解和支持，制度上给予的便利和疏通，以及在学业上的关心和照顾，在此对赵晓芳老师和唐宏伟博士表示深深的感谢。

感谢魏彦杰副研究员，在我五年博士研究工作中对我的指导和关照。他亦师亦友，在研究中治学严谨，对我的研究给予了悉心的指导，在我的课题上投入了大量的时间和精力，并对我的研究工作提供的各种物质和资源的调配，使我可以顺利开展研究工作。此外魏博士也非常注重国际合作交流，并为我找到 Pavan Balaji 研究员对我的工作进行指导和合作。在此对魏博士致以深深的感谢。

感谢王丙强博士，在我于华大基因研究院实习期间给予了全面的生命科学领域的指导。王丙强博士是一位资深计算生物学专家。他在生命信息学方面，特别是生物信息产业链中的关键科学问题方面给我了细致辅导，让我快速的进入了生命科学领域的研究工作，并在后续的生物和计算跨领域研究工作中给予了指导。 在此感谢王丙强博士在我博士研究课题上的支持和生活上的关心。

感谢 Pavan Balaji 研究员和 Sangmin Seo 博士，在阿贡实验室对我研究工作的支持和悉心指导，Pavan 在体系结构和系统性能优化方面有着深厚的基础，并对我的研究工作进行了深入指导，特别是他的研究方法，做学问的要求，高效的执行力，顶级实验室的管理方法以及对领域研究的直觉，对我有着深刻的影响。

感谢成杰峰博士，李振国博士，在华为诺亚实验室对我的提携和指导。成杰峰博士是一位编程论文全能博士，对能实际改变大众生活的研究有着强烈的热情。李振国博士有着极好的人格魅力和学术理想，并时刻关注着 AI 领域前瞻性科技发展。在与两位博士的工作中耳熏目染并对工业科研有着初步的认识。

感谢现在（以及曾经）和我工作在一起的高性能计算中心的同事：文高进博士，黄伟博士，刘涛博士，张涌博士，王伟民博士，李晴岚博士，洪爵博士，李俊杰博士，何耀彬博士，张桂娟博士，张霄宏学姐，赵中英学姐，熊文，贝振东，陈羚，张莎，赵娟

娟，张巍，黄振宇，王周，王治，贺龄慧，张慧玲。感谢你们对我工作的支持和对我生活上的帮助。

此外，感谢钱德培老师，学长阙欣宇博士，以及夏应龙博士对我科研和学术规划上的指导和帮助。

最后，特别感谢我的爱人苑建蕊在我博士期间，给予的精神与物质上的支持，爱人的支持与鼓励使我坚持在学术道路上可以一心一意做研究，完成学术研究。感谢爱人给我们的小家庭带来了可爱的小宝宝孟芊璇。感谢我的岳父岳母帮我分担家务事，照顾宝宝，使我得以有更多的时间进行科研工作。感谢我的父母对我工作的支持。

孟金涛

2016.4.22

作者简介

【基本情况】

孟金涛，男， 湖北宜城人，1982 年 12 月出生。

中国科学院深圳先进技术研究院工程师，

中国科学院计算技术研究所博士研究生。

【教育情况】

2001 年 9 月-2005 年 6 月　　华中师范大学计算机科学与技术系，理学学士

2005 年 9 月-2008 年 6 月　　华中师范大学计算机科学与技术系，工学硕士，硕士专业： 计算机应用

2011 年 9 月-2016 年 6 月　　中国科学院计算技术研究所，工学博士，博士专业：计算机体系结构

【工作经历】

2008 年 7 月-至今， 中国科学院深圳先进技术研究院， 工程师。

【研究兴趣】

高性能计算， 生物信息， 并行图算法

【联系方式】

通讯地址： 深圳市南山区西丽深圳大学城学苑大道 1068 号

邮编： 518055

Email: jt.meng@siat.ac.cn