**密级:**



**硕士学位论文**

**面向基因组数据分析的大数据编程框架**

**作者姓名 王炳琛**

**指导教师 张佩珩 正研级高工**

**中国科学院计算技术研究所**

**学位类别 工学硕士**

**学科专业 计算机系统结构**

**培养单位 中国科学院计算技术研究所**

**2017年5月**

**Programming Framework**

**For Big Genome Data Analysis**

A Thesis Submitted to

**The University of Chinese Academy of Sciences**

in partial fulfillment of the requirement

for the degree of

**Master of Science**

in

**Computer Architecture**

By

**Bingchen Wang**

**Thesis Supervisor: Professor Peiheng Zhang**

Institute of Computing Technology

Chinese Academy of Sciences

May, 2017

**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计算技术研究所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

**关键词：**BWA，序列比对，众核架构，异构平台

**Abstract**

.

**Key words:** m

目录

[摘要 I](#_Toc452106428)

[目录 IV](#_Toc452106429)

[图目录 VI](#_Toc452106430)

[表目录 VIII](#_Toc452106431)

[第一章 引言 1](#_Toc452106432)

[1.1 生物测序与序列比对的快速发展 1](#_Toc452106433)

[1.2 硬件平台的不断革新 2](#_Toc452106434)

[1.2.1 GPU简介 3](#_Toc452106435)

[1.2.2 MIC简介 6](#_Toc452106436)

[1.3 研究动机 7](#_Toc452106437)

[1.4 本文主要贡献 8](#_Toc452106438)

[1.5 本文组织结构 8](#_Toc452106439)

[第二章 背景及相关研究 10](#_Toc452106440)

[2.1 生物序列比对算法概述 10](#_Toc452106441)

[2.2 生物序列比对算法加速相关研究 12](#_Toc452106442)

[2.3 BWT与FM-INDEX 13](#_Toc452106443)

[2.4 BWA简介 14](#_Toc452106444)

[2.5 本章小结 16](#_Toc452106445)

[第三章 基于GPU平台的BWA加速研究 17](#_Toc452106446)

[3.1 算法整体流程分析 17](#_Toc452106447)

[3.2 算法热点测试 19](#_Toc452106448)

[3.3 BWA-MEM的GPU加速设计与实现 21](#_Toc452106449)

[3.3.1 整体设计 21](#_Toc452106450)

[3.3.2 任务划分 24](#_Toc452106451)

[3.3.3 存储设计与优化 25](#_Toc452106452)

[3.3.4 性能测试 27](#_Toc452106453)

[3.4 GPU加速序列比对算法对比分析 28](#_Toc452106454)

[3.4.1 BWA-backtrack与BarraCUDA 28](#_Toc452106455)

[3.4.2 Bowtie2与nvBowtie 29](#_Toc452106456)

[3.4.3 BWA-MEM算法特征及其对GPU加速的影响 30](#_Toc452106457)

[3.5 本章小结 33](#_Toc452106458)

[第四章 基于MIC平台的BWA加速研究 34](#_Toc452106459)

[4.1 整体设计 34](#_Toc452106460)

[4.2 基础版本的移植 35](#_Toc452106461)

[4.2.1 交叉编译环境配置 35](#_Toc452106462)

[4.2.2 依赖库的交叉编译 36](#_Toc452106463)

[4.2.3 文件传输 36](#_Toc452106464)

[4.2.4 通过NFS访问较大的数据 37](#_Toc452106465)

[4.2.5 基础版本性能测试 37](#_Toc452106466)

[4.3 流水线优化 38](#_Toc452106467)

[4.4 扩展性测试 42](#_Toc452106468)

[4.5 指导向量化与数据对齐 43](#_Toc452106469)

[4.6 Smith-Waterman向量化 48](#_Toc452106470)

[4.7 本章小结 53](#_Toc452106471)

[第五章 性能评测 54](#_Toc452106472)

[5.1 实验配置 54](#_Toc452106473)

[5.2 BWA-MEM加速效果测试 55](#_Toc452106474)

[5.2.1 Single-end模式 55](#_Toc452106475)

[5.2.2 Pair-end模式 56](#_Toc452106476)

[5.3 BWA-MEM加速算法微架构分析 56](#_Toc452106477)

[5.3.1 CPI指标 56](#_Toc452106478)

[5.3.2 Cache利用率 57](#_Toc452106479)

[5.3.3 向量化效果 59](#_Toc452106480)

[5.4 本章小结 62](#_Toc452106481)

[第六章 结束语 63](#_Toc452106482)

[6.1 本文工作总结 63](#_Toc452106483)

[6.2 下一步工作 63](#_Toc452106484)

[参考文献 65](#_Toc452106485)

[致谢 i](#_Toc452106486)

图目录

[图 1.1 由基本字符组成的生物序列 （引自[1]） 2](#_Toc452106487)

[图 1.2 NVIDA GPU与Intel CPU的浮点运算性能对比 （引自[4]） 4](#_Toc452106488)

[图 1.3 NVIDA GPU与Intel CPU的访存带宽性能对比 （引自[4]） 4](#_Toc452106489)

[图 1.4 CUDA程序执行过程 （引自[4]） 5](#_Toc452106490)

[图 1.5 MIC架构 （引自[5]） 6](#_Toc452106491)

[图 1.6 MIC平台中每个核心的架构 （引自[5]） 7](#_Toc452106492)

[图 2.1 序列S=GOOGOL的前缀树 （引自[17]） 11](#_Toc452106493)

[图 2.2 BarraCUDA加速BWA效果 （引自[40]） 13](#_Toc452106494)

[图 2.3 FM-index原理图 14](#_Toc452106495)

[图 2.4 BWA-ALN核心算法 （引自[17]） 15](#_Toc452106496)

[图 2.5 BWA-ALN非精确匹配伪代码 （引自[17]） 15](#_Toc452106497)

[图 3.1 BWA-MEM算法流程 18](#_Toc452106498)

[图 3.2 BWA-MEM算法热点分析（100bp） 20](#_Toc452106499)

[图 3.3 BWA-MEM算法热点分析（250bp） 20](#_Toc452106500)

[图 3.4 BWA-MEM算法对于不同长度reads的热点对比 21](#_Toc452106501)

[图 3.5 mem\_chain模块算法流程 22](#_Toc452106502)

[图 3.6 BWA-MEM多kernel并行框架 23](#_Toc452106503)

[图 3.7 BWA-MEM重构前的多线程模式 23](#_Toc452106504)

[图 3.8 BWA-MEM重构后的多线程模式 24](#_Toc452106505)

[图 3.9 nvcc编译移植的BWA-MEM存储器使用情况 25](#_Toc452106506)

[图 3.10 优化前的chain存储格式 26](#_Toc452106507)

[图 3.11 优化后的chain存储格式 26](#_Toc452106508)

[图 3.12 BarraCUDA对BWA-backtrack算法搜索方法的优化 （引自[40]） 28](#_Toc452106509)

[图 3.13 BWA-backtrack seeding模式与BarraCUDA优化方法 29](#_Toc452106510)

[图 3.14 Bowtie2的固定seed模式 （引自[18]） 30](#_Toc452106511)

[图 3.15 寻找超级最长精确匹配（SMEM）伪代码 （引自[46]） 32](#_Toc452106512)

[图 3.16 BWA-MEM采用的前向-后向搜索算法图示 32](#_Toc452106513)

[图 4.1 MIC平台基础版本加速效果 38](#_Toc452106514)

[图 4.2 MIC平台BWA各阶段运行时间对比 39](#_Toc452106515)

[图 4.3 BWA无流水线执行模式 39](#_Toc452106516)

[图 4.4 BWA在MIC平台上三级流水线执行模式 39](#_Toc452106517)

[图 4.5 优化后的三级流水线处理模式 40](#_Toc452106518)

[图 4.6 无流水线时的程序执行时序图 40](#_Toc452106519)

[图 4.7 优化后的三级流水线程序执行时序图 41](#_Toc452106520)

[图 4.8 采用三级流水线优化后性能测试结果 41](#_Toc452106521)

[图 4.9 四级流水线优化设计 42](#_Toc452106522)

[图 4.10 BWA-MEM扩展性测试 43](#_Toc452106523)

[图 4.11 SIMD向量部件示意图 43](#_Toc452106524)

[图 4.12 指导向量化代码实例 44](#_Toc452106525)

[图 4.13 icc编译器向量化报告示例 46](#_Toc452106526)

[图 4.14 数据对齐与指导向量化加速效果 48](#_Toc452106527)

[图 4.15 E、F、H矩阵数据依赖关系 50](#_Toc452106528)

[图 4.16 向量任务划分方式 51](#_Toc452106529)

[图 4.17 矩阵计算伪代码 52](#_Toc452106530)

[图 4.18 ksw\_extend2向量化加速效果 52](#_Toc452106531)

[图 5.1 CPI测试结果 57](#_Toc452106532)

[图 5.2 热点模块向量化程度测试 60](#_Toc452106533)

[图 5.3 L1 COMPUTE / DATA ACCESS测试结果 61](#_Toc452106534)

[图 5.4 L2 COMPUTE / DATA ACCESS测试结果 61](#_Toc452106535)

表目录

[表 3.1 BWA-MEM传输到GPU数据的存储方式 27](#_Toc452106536)

[表 3.2 mem\_collect\_intv模块CPU平台和GPU平台运行时间对比 27](#_Toc452106537)

[表 3.3 generate\_chain模块CPU平台和GPU平台运行时间对比 27](#_Toc452106538)

[表 5.1 实验平台处理器主要参数 54](#_Toc452106539)

[表 5.2 Single-end模式测试结果 55](#_Toc452106540)

[表 5.3 Pair-end模式测试结果 56](#_Toc452106541)

[表 5.4 Intel® Xeon Phi™ 协处理器cache信息 58](#_Toc452106542)

[表 5.5 Cache命中率及未命中延迟测试结果 59](#_Toc452106543)

# 第一章 引言

生物信息学是一门新兴的交叉学科，其主要目的是通过运用数学、生物学和计算机科学的各种工具，来阐明大量数据所包含的生物学意义。自上世纪80年代末人类基因组计划提出并成功实施以来，尤其是伴随着近年大数据处理技术的发展，生物信息学逐渐发展成为当今生命科学和自然科学的前沿领域之一。

生物大分子（如DNA、RNA和蛋白质等）的测序过程由始至终贯穿着整个生物信息学的研究过程。

生物信息学的基本任务是对各种生物分析序列进行分析，研究新的计算方法，从大量的序列信息中获取基因结构、功能和进化等知识。序列比较是生物信息学中最基本、最重要的操作，通过序列比对可以发现生物序列中的功能、结构和进化的信息。在生命科学中，一个普遍的规律是序列决定结构，结构决定功能。序列比较的根本任务是：通过比较生物分子序列，发现它们的相似性，找出序列之间共同的区域，同时辨别序列之间的差异。研究序列相似性的主要目的是通过序列的相似性，判别序列之间的同源性，推测序列之间的进化关系，通过相似的序列得到相似的结构或相似的功能。这里，将序列看成由基本字符组成的字符串，无论核酸序列还是蛋白质序列，都是特殊的字符串。

生物序列比对是计算机领域经典的文本比对问题在生物领域的应用。随着不断涌现的新型分子生物学技术的发展，随之而来的比如基因变异、RNA 表达、蛋白质和基因相互作用等分子生物学研究需要研究人员采用高通量的方法去解释。这对高性能计算提出了新的挑战。与此同时，计算机硬件平台也在近几年飞速发展，不断推陈出新，新得多核平台和众核平台不断出现，2006年TOP500出现了第一台使用加速设备的高性能计算机。众核协处理器加速设备使高性能计算机的性能快速提高，众核异构平台正成为构建高性能计算机的重要方式。这也使得计算机体系结构变得越来越复杂，给计算程序的优化带来了新的难题。BWA作为当前使用最为广泛，性能最好的生物序列比对软件之一，在近几年也不断发展，其最新的BWA-MEM算法对当前的新型测序仪产生的序列片段比对具有非常好的性能，是生物测序领域的基础算法之一。随着高通量测序技术的不断发展，利用众核处理器对BWA序列比对算法进行加速，具有重要的现实意义。本章首先介绍生物测序平台和算法的不断发展、硬件计算平台的发展与变革，然后阐明本文的研究动机，最后列出本文的主要贡献和本文的组织结构。

## 1.1 基因测序技术的发展与应用

序列比较是生物信息学中最基本、最重要的操作，通过序列比对可以发现生物序列中的功能、结构和进化的信息。序列比较的根本任务是：通过比较生物分子序列，发现它们的相似性，找出序列之间共同的区域，同时辨别序列之间的差异[1]。在分子生物学中，DNA或蛋白质的相似性是多方面的，可能是核酸或氨基酸序列的相似，可能是结构的相似，也可能是功能的相似。一个普遍的规律是序列决定结构，结构决定功能。研究序列相似性的目的之一是，通过相似的序列得到相似的结构或相似的功能。这种方法在大多数情况下是成功的，研究序列相似性的另一个目的是通过序列的相似性，判别序列之间的同源性，推测序列之间的进化关系。这里，将序列看成由基本字符组成的字符串，无论核酸序列还是蛋白质序列，都是特殊的字符串。

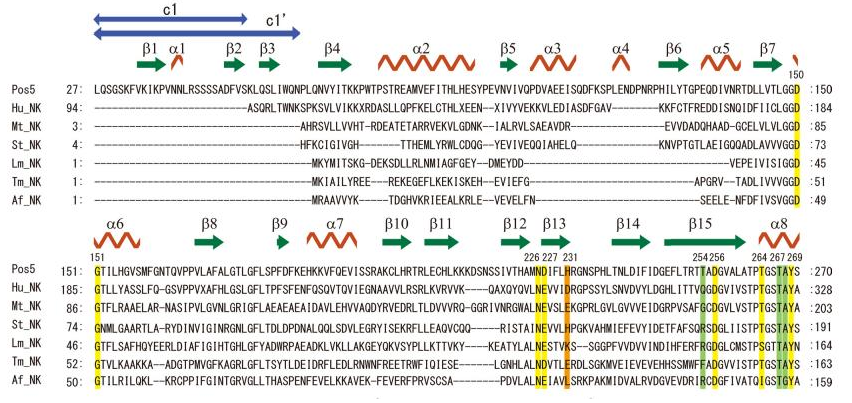


图 1.1 由基本字符组成的生物序列 （引自[1]）

生物序列比对相关算法的研究开始于20 世纪70 年代，基本的理论和算法日趋成熟。早期的序列比对算法由于需要处理的数据量小，因此采用动态规划思想的Smith-Waterman 算法，启发式的BLAST算法就能很好地满足数据分析的需求。下一代测序(Next Generation Sequencing，NGS)技术的发展使得序列分析面临严峻的挑战。下一代测序又称高通量测序，它的一大特点是高通量，其数据规模超出了中小型实验室的数据处理能力。高通量测序时代的挑战不再是数据的产生，而是数据存储,处理和分析。第三代测序技术的发展将进一步加快测序速度，同时产生较长的测序片段，对序列比对技术的发展提出了更高的要求。

## 1.2 大数据处理技术的发展

包括生物序列比对在内，大量的高性能计算需求推动了计算机硬件平台的不断发展。如今，计算机体系结构越来越复杂，多核和众核平台不断发展，根据摩尔定律(Moore’s Law)[3]以每18个月翻一番的速度持续增加芯片上晶体管的数量。在过去几十年中，处理器工艺水平不断增加，体系结构设计也在不断创新，处理器的功能变得越来越强大，超级流水线、超标量等技术逐渐应用到处理其中。尽管如此，未来CPU的快速发展将面临巨大的难题。晶体管的尺寸越来越接近原子大小，这导致越来越严重的漏电问题。此外，单位尺寸上的发电量和能耗变得更多。由于这些原因，处理器的频率和制作工艺很难像之前的那样飞速地提高。与此同时，存储器的存取速度的发展大大落后于处理器性能的提升。访存已经越来越成为高性能计算的瓶颈。针对以上的种种问题，新型的体系结构不断被开发出来。例如众核平台，它们有大量的计算单元，能进行密集计算，充分挖掘应用程序的内在的并行性。此外，它们往往具有优化的访存设计，能提供更高效的数据访问。

从2007年开始，GPGPU的广泛应用标志着众核时代的到来。众核平台在提升计算能力的同时还能降低能耗。2012年Intel推出了MIC（Many Integrated Core）平台，作为高性能加速设备支持并行计算，给高性能计算领域提供了一个可选择的方案。

GPU（Graphic Processing Unit），即图形处理器，以前主要针对图像处理相关的应用。现在也越来越多的应用到高性能领域。从2000年以来，GPU的浮点运算能力不断提高，其发展速度几乎是摩尔定律的3倍，平均以每12个月翻一番。其计算能力甚至超越了同时期的CPU。GPU的概念是NVIDA公司于1999年发布GeForce 256图形处理器时提出的。从那时起，各种复杂的大量的应用需求促使GPU产业不断蓬勃发展。

近年来GPU的浮点运算能力和显存带宽已经远远领先同时期的CPU，如图所示。随着架构的创新和技术的进步，GPU在计算单元、访存部件等方面取得了相当大的改进。不断满足实时处理、高分辨率3D图形等应用的计算需求。得益于其SIMT ( Single Instruction Multiple Threads )众核架构，GPU在很多大规模数据并行的应用中发挥着重大的作用。最开始的GPU只是单独的用在图像处理中，后来其强大的浮点运算能力被推广到图像处理之外的其他应用领域。NVIDIA公司和AMD公司先后提出了GPGPU（General-purpose graphics processing unit）概念。通过改变GPU内部的硬件结构和实现细节，增强了GPU的功能，使得它能够适用于更多的领域。图1.2和1.3给出了NVIDIA GPU与Intel CPU浮点性能以及访存带宽的发展和对比。

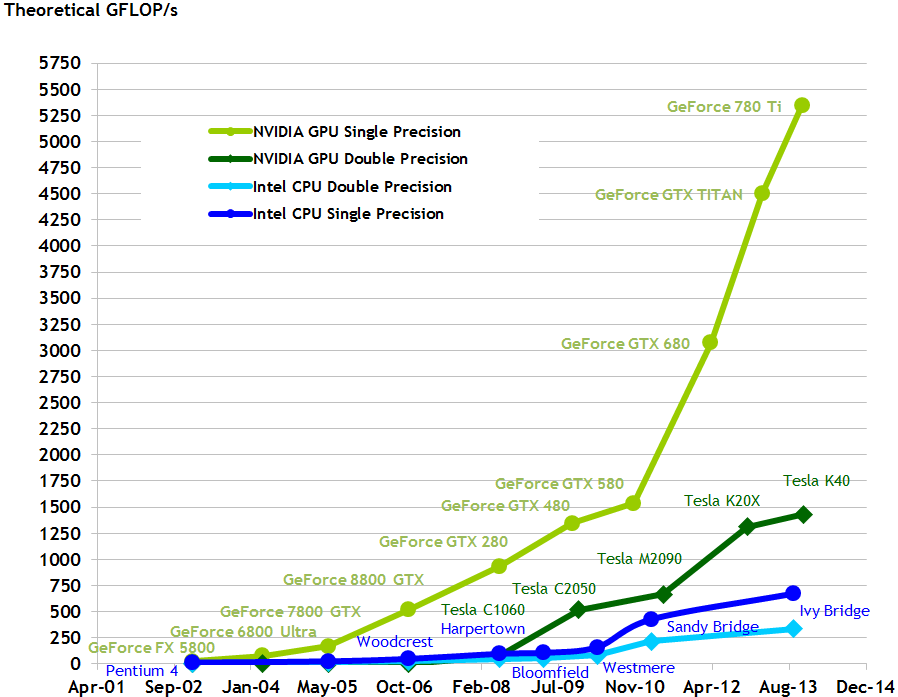


图 1.2 NVIDA GPU与Intel CPU的浮点运算性能对比 （引自[4]）

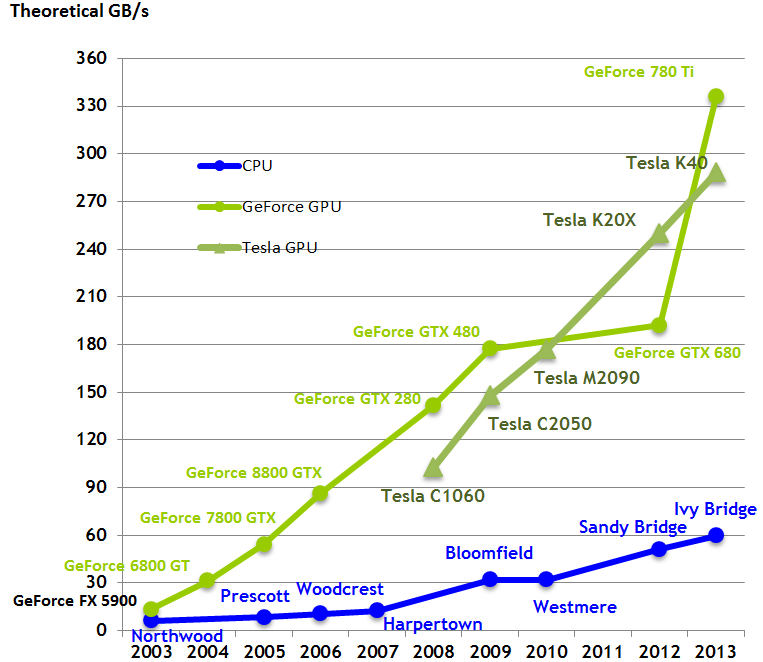


图 1.3 NVIDA GPU与Intel CPU的访存带宽性能对比 （引自[4]）

1999年以前，第一代GPU把部分功能从CPU分离出来，提供硬件加速，只能起到3D图像处理的加速作用，不具备软件编程的特性。1999年到2002年，第二代GPU提供了有限的编程性，2001年NVIDIA和ATI分别推出的GeForce3和Radeon 8500，定义了流处理器概念，出现了顶点级和有限的像素级编程性。2002年以后，第三代GPU普遍提供了比较方便的编程环境，比如CUDA等。这样使得GPU的可编程性不断增强，也使得GPU通用计算编程的复杂性大幅度降低。宣告了GPU面向通用计算的新时代的到来。CUDA编程语言由NVIDIA开发，是一个通用并行计算架构。开发人员可以使用C或C++来编写CUDA程序。CUDA程序的一般执行过程如图1.7所示。CUDA编程模型将硬件平台分为主机端和设备端两个部分，CPU作为主机端，GPU作为计算设备进行辅助计算，他们各自维护自己的 内存空间。在主机端执行的部分一般用于流程控制，以及主机端和设备端的数据传输操作。比较耗时的计算部分一般都放到设备端运行，GPU上的线程首先被划分成多个网格，网格包含多个线程块。网格与SPA（streaming processor array）一一对应。这些对程序员都是透明的。

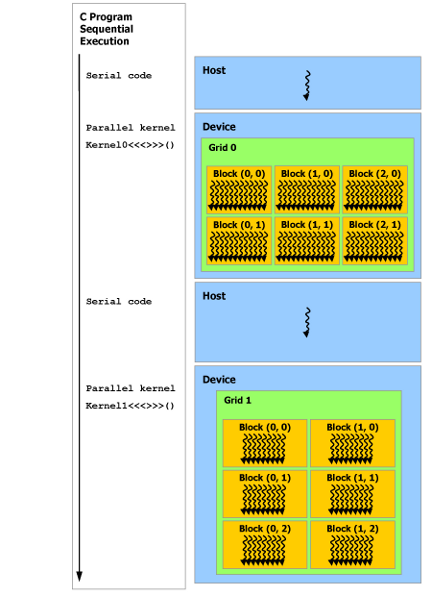


图 1.4 CUDA程序执行过程 （引自[4]）

Intel MIC全称为Intel Many Integrated Core Architecture。是一个多个核心顺序执行的体系结构，其架构如图1.8所示。2012年6月18日Intel在国际超级计算会议上发布了基于MIC架构的Xeon Phi协处理器，其包含多达60个核心，主要面向高性能计算领域。MIC的每个核心具有四个线程，他们共享一个核，线程的切换开销比较小。MIC的所有核心的连接以及核心与卡上的内存都是通过环形总线相互连接，其内存带宽高达352GB/s。MIC具有独立的二级缓存，并且是完全一致的高速缓存结构。每个核心有32KB的L1指令缓存和32KB的数据缓存。同时，MIC架构提供了一个非常方便的编程环境，支持很多现有的编程工具和编程语言。支持OpenMP等多线程编程环境。提供了多种使用方式，包括native和offload模式。

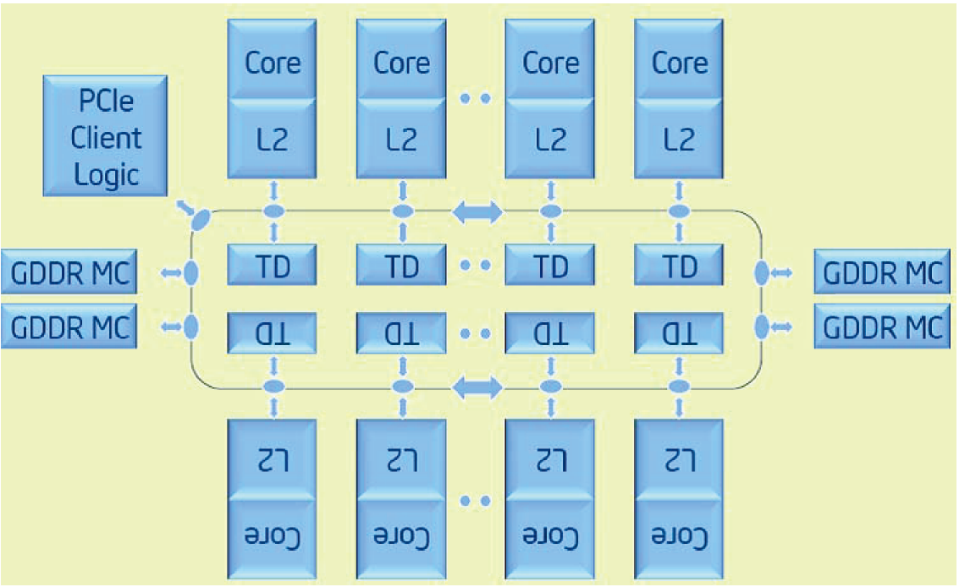


图 1.5 MIC架构 （引自[5]）

MIC每个核心的架构如图1.9所示，每个核心包含标量处理器（ALU）和向量处理器(VPU)，每个向量处理器包含512位的向量寄存器。VPU可以一次处理8个双精度浮点数或者16个单精度浮点数。

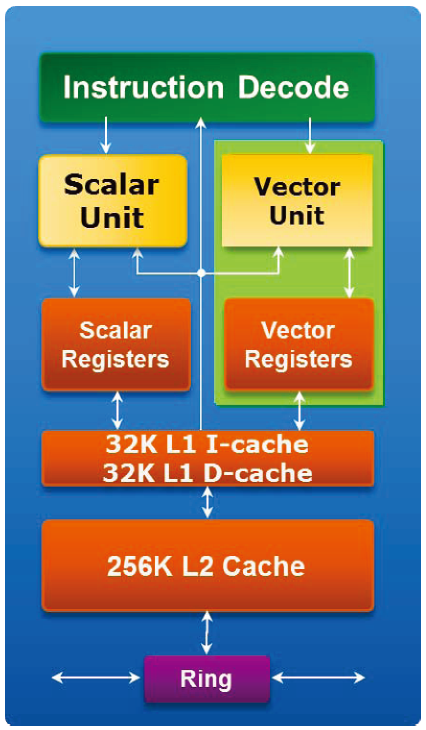


图 1.6 MIC平台中每个核心的架构 （引自[5]）

## 1.3 研究动机

前文提到，高通量测序技术的不断发展，对序列比对技术的发展提出了更高的要求。如今的计算平台和体系结构越来越复杂，而体系结构的不同对算法的性能影响很大。在以前相当长一段时间内，CPU频率持续提升，程序的性能随着处理器的发展及主频的提升而变得更好。然而，随着工艺水平受到物理方面的限制，CPU的主频已经无法持续提高，处理器的发展进入多核时代。如今，众核处理器以其出色的性价比，逐步成为研究人员对序列比对算法进行加速的有力武器。不过，由于众核处理器架构的特殊性，其开发与优化的难度要远远高于通用CPU架构程序，对开发人员提出了更高的要求。总体来说，研究如何加速高通量特征下的序列比对工具具有以下几点重要意义：

1. 生物序列比对作为生物信息学研究的基本手段，在多个研究领域发挥着重要作用。
2. 随着生物序列数据的快速增长，大规模序列比对变得极为耗时，需要庞大的计算能力。
3. 不同的应用方向往往对序列比对工具提出了不同的需求。
4. 伴随着新一代测序技术的发展，生物数据的数据量和数据性质都会发生改变。
5. 面对未来第三代测序技术产生的海量中长串的比对任务，现有的传统比对工具和短串比对工具都有其自身的限制。

BWA作为当前使用最为广泛，性能最好的生物序列比对软件之一，在近几年也不断发展，其最新的BWA-MEM算法对当前的新型测序仪产生的序列片段比对表现出了非常好的性能，是生物测序领域的基础算法之一。但是，对于不断优化的BWA算法，特别是最新出现的BWA-MEM算法，基于众核平台的加速研究目前还很少。因此，本文基于众核处理器平台，对BWA特别是BWA-MEM算法在众核平台的加速方法进行了深入研究，希望能够让BWA算法的特征与众核加速设备的特征紧密结合，充分利用众核平台强大的运算能力，为BWA序列比对算法提供更优异的性能。

## 1.4 本文主要贡献

本课题对BWA序列比对算法在众核平台的加速方法与效果进行了深入分析和研究，分别利用GPU和MIC加速设备对BWA进行加速，结合算法特性对加速效果进行了深入分析。主要贡献有：

1. 深入分析了最新的BWA-MEM算法的整体流程与热点模块，具有一定的普遍意义。
2. 设计并实现了BWA-MEM算法热点模块的GPU移植，结合GPU平台架构和硬件特性对算法进行了优化，比较了与已有的序列比对工具的GPU加速版本诸如BarraCUDA、nvBowtie等在优化过程中存在的差异，分析了影响BWA-MEM算法GPU加速效果的主要因素。
3. 设计并实现了BWA算法在MIC平台的移植，针对MIC平台架构和硬件特性进行了深度优化，在单MIC设备上获得了相当于10-12个CPU核心的比对速度，是目前已知的第一个对包括BWA-MEM算法在内的完整BWA算法的MIC平台移植工作

## 1.5 本文组织结构

本论文总共分为六章，具体组织结构如下：

第一章阐述了研究本课题的研究背景，生物序列比对算法的各种硬件计算平台的发展及现状。引出本可以的研究动机和意义，介绍了本课题的研究内容和主要贡献。

第二章介绍了相关的技术背景和发展现状。包括生物序列比对主要算法介绍，生物序列加速算法研究进展，本课题所需要的BWT与FMD-index相关背景知识，BWA简要介绍，以及本课题使用的硬件平台CUDA和MIC编程模型。

第三章详细介绍了利用GPU平台加速BWA的设计方案与实现方法，对BWA-MEM算法进行了算法流程与热点分析，将其移植到GPU平台，对GPU平台BWA加速效果的测试，分析了BWA-MEM算法在GPU平台加速效果受到限制的主要因素

第四章详细介绍了如何利用MIC平台对BWA进行加速，深入阐述了BWA向MIC平台移植的方法，详细介绍了流水线、数据对齐与指导向量化、手工向量化的实现细节。

第五章对MIC平台加速的BWA进行了全面的性能测试，对BWA-MEM算法的各个分支利用不同的数据集进行性能评价，并与CPU平台、GPU平台的常用比对软件进行性能比较。

第六章总结全文，并展望下一步工作。

# 第二章 背景及相关研究

## 2.1 典型基因数据处理流程概述

关于生物序列比对算法的研究从20 世纪70 年代就开始了，到目前，其相关理论和算法都已经比较成熟。最初的序列比对由于数据量小，因此利用Smith-Waterman 算法、启发式BLAST算法就已经能够满足需求。但是，大规模的数据（百万级甚至上亿级）在reference 上的比对，这些序列比对工具已经不符合序列比对的需求了。因此，我们必须采用启发式算法加快比对速度。目前，我都多数是是通过建立索引的方法加快序列比对的速度。建立索引，是指用辅助的数据结构来存储待比对的序列，目的是凸显序列中有一定规则的子序列。这些数据结构中，哈希表和后缀树（后缀数组）是比较常用的两种。

### 2.1.1 基因序列比对

### 2.1.2 基因数据清理

### 2.1.3 变异位点发现

基于seed的哈希表索引方法，以哈希表的形式建立序列的索引，比较适合数据库搜索以及将reads比对到参考序列上。下面简要介绍一种基于种子的哈希表方法：

1. 建立哈希表。

DNA序列，一般是由A、 C、 T、 G 四种脱氧核苷酸组成, 长度为k 的seed有4k 种可能。将4k个种子存放在哈希表中，并将4个碱基的值转换为二进制数据表示，这样就能够给每个碱基指定一个数字标识，并且内存占用只有2比特。

1. 关联数据库中的序列与哈希表

设一个含有条DNA序列的数据库 , 将数据库中的每条序列分解为连种子，其长度，从种子的头开始每次偏移一位直到序列 结束，那么长度为的序列有个种子。接下来，我们将每一个“种子”所在的序列 和序列中的顺序号 在对应的哈希表 中的seed以二维数组方式记录下来。将序列与哈希表建立关联，

1. 将查询序列Q以k长度进行分割，成为种子，计算该种子对应的标识号，接下来利用哈希表查找其在数据库中的匹配位置。

研究人员长期以来致力于对seed形式和选择的改进，因为seed长度、允许错配和indel数量等因素，都会直接影响比对结果的敏感度、准确性、时间空间花费等重要指标。在采用hash方法的比对软件中，SOAP[9]、SeqMap[10]、MAQ[11]、SHRiMP[12]等都使用不同的种子模型，比较常见的包括空位种子、空位种子组（同时使用多个种子模型）等。此外，SeqMap、MAQ、 RMAP[13]等软件对reads序列建立索引，另而一些软件则将参考序列建立索引。另一种比较常用的优化方法是，根据一个设置好的阈值F，将哈希表中出现频率低于F的seed删去。

后缀树是一种重要的数据结构，在序列比对软件中有重要应用，一些典型的序列比对算法如MUMmer1[14]、MUMmer2[15]都是基于后缀树这种数据结构实现的。后缀树最初常用语比对两个序列的相似性，寻找最大公共子序列。但是在生物序列比对中，将reads比对到参考序列上，是将reads或其子序列按序排列的所有后缀中一一比对找到匹配位置。与后缀树对应，前缀树也是一种常用的数据结构，采用前缀树的典型比对算法包括BWA-SW[16]等，它们将序列的所有前缀用前缀树的结果表示出来，其原理与后缀树类似，如图2.3所示为序列的前缀树。

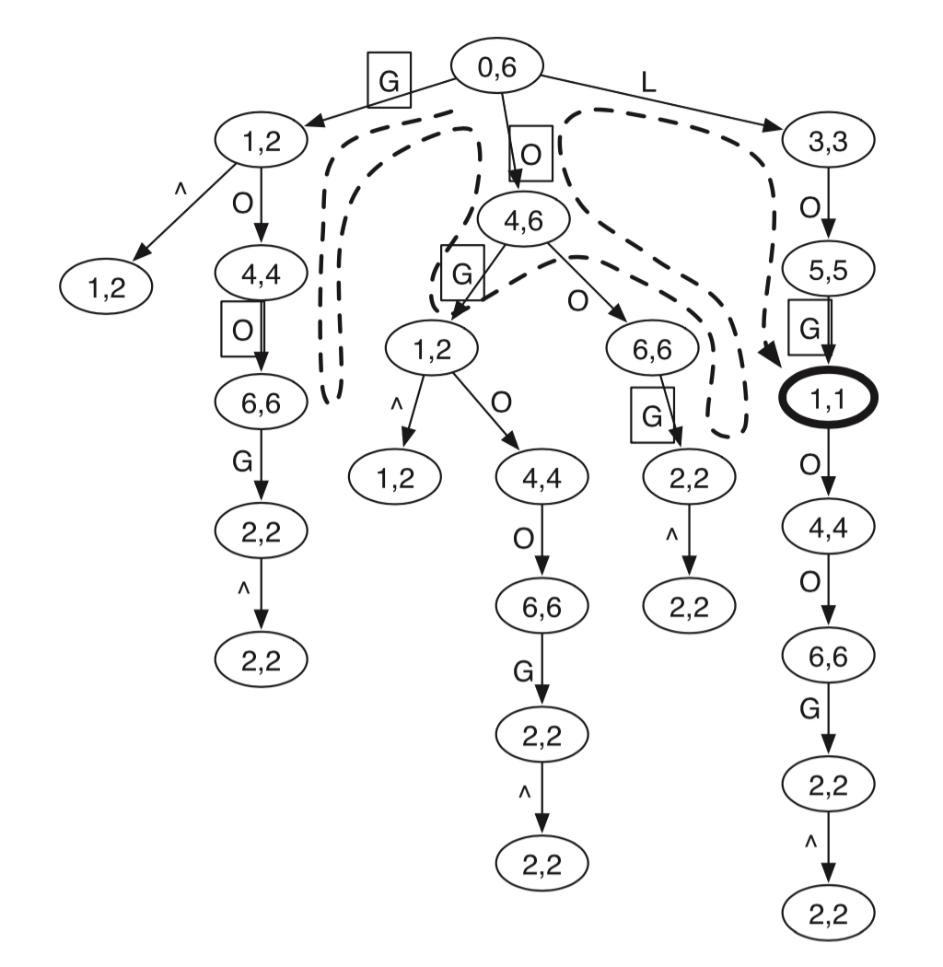


图 2.1 序列S=GOOGOL的前缀树 （引自[17]）

后缀树数据结构存在的一个问题是，对于较长的序列，其空间占用非常大。增强后缀数组[20]是Abouelhoda等对Manber和Myers提出的后缀树存储方法的改进，能够提高空间利用率。Ferragina和Manzini提出了 FM(full-textminute-space)-Index[21]，这是一种基BWT（Burrows-Wheeler Transform)的全文本压缩索引结构[22]，它利用BWT矩阵与后缀数组之间的关系，实现了压缩的后缀数组和索引，大幅减少了内存占用率。

生物序列比对的常用软件BWA-SW、Bowtie2[18]、BWT-SW[19]等都采用了这种方法。在这种方法中应用FM-Index，每隔若干行建立一个索引，可以大幅降低内存占用。比对人类基因组3GB的数据，其内存占用约为1.3GB。

## 2.2 现有并行优化工作

测序技术的飞速发展，使测序仪短时间内产生的测序数据量更大，因此需要更快的序列比对算法，对此，科研人员尝试了大量方法对序列比对进行加速，整体来看，主要通过如下两种方式对生物序列比对算法进行加速。

一是不断优化生物序列比对算法，采用更合适的数据结构，更精巧的比对方法来实现对生物序列比对的速度和准确率的提升，这种方式包括改进已有的算法和研究序列比对的新算法。例如BWA的作者Heng Li先后在其软件中加入了BWA-ALN，BWA-SW，BWA-MEM三种不同的算法，这些算法可以应用于不同特征的序列，同时性能也有一定的提升。

另一种是将已有的生物序列比对算法移植到计算能力更为强大的高性能计算平台上。这种方法根据其计算平台的不同，需要采取不同的优化手段，具体来说，目前主要的加速平台有以下几种。

1. 线程级并行（如OpenMP[23]）或进程级并行（如MPI[24]）

这是一种最传统的加速方法。能够充分利用多核处理器的运算核心，目前较新的比对工具都加入了线程级并行，如最新版的BWA、Bowtie2都支持多线程模式。

进程级是集群环境的主要并行手段，进程级并行一般是将目标数据库或查询序列进行分割，分配到每个进程上去计算。由于这种分割方式独立性更好，因此加速比几乎可以随随处理器数量线性增长。这方面比较有代表性的有MPI BLAST[27]等。

1. 分布式计算

Hadoop是一种可靠、高效、可伸缩的分布式处理软件框架。这个框架包括分布式文件系统HDFS（Hadoop Distributed File System）和分布式并行计算模型 Map Reduce。基于 Hadoop，印第安纳大学 SALSA（Service Aggregated Linked Sequential Activities）项目[28]实现了Hadoop-Blast[29]，利用Hadoop任务分发的高效性和数据的可靠性进行序列比对，能够便利地部署到云计算环境中，服务于生物序列数据分析任务。

1. 异构计算

异构计算的基本思想是利用辅助加速硬件来加速运算速度。目前流行解决方案主要有：可编程门阵列（Field-programmable Gate Array，FPGA）与通用处理器（Central Processing Unit，CPU）混合架构、GPU 与 CPU 混合架构、Intel集成众核（Intel Many Integrated Core，MIC）与CPU混合架构。异构计算性价比高，但是开发难度较大。但是，随着NVIDIA公司推出的CUDA编程模型，在GPU上进行并行程序开发难度被大大降低，因此目前出现了大量的序列比对GPU移植工作。如MUMmer GPU[31]是基于MUMmer 算法的 GPU 加速版本，能够使一些特定序列比对获得 10 倍的加速比。对于常用的Smith-Waterman 算法和 BLAST 算法，也已经有了多个GPU一直版本，如SW-CUDA[32]、GSW[33]和 CUDASW++[34]等， GPU-BLAST[35]。

短串比对工具的GPU移植工作目前还不多。比较有代表性的如SOAP3[38]是 SOAP2[39]的 GPU 移植，采用 SOAP2 的快速比对算法，GPU版本的比对能够对CPU版本的BWA和 Bowtie有，能够分别达到 7.5 倍和 20 倍加速，并且对于至多允许 4 个不匹配的比对，其能够得到更7.5和20倍加速。Barra CUDA[40]是基于BWA 0.6版本开发GPU比对工具，它不仅支持允许空位比对，还可以利用多个GPU提升比对速度，当采用一个GPU时，其处理能力与在6个CPU核相当。此外，CUSHAW[41]基于 Burrows-Wheeler 变换和FM-index，也能够达到较好的加速效果。

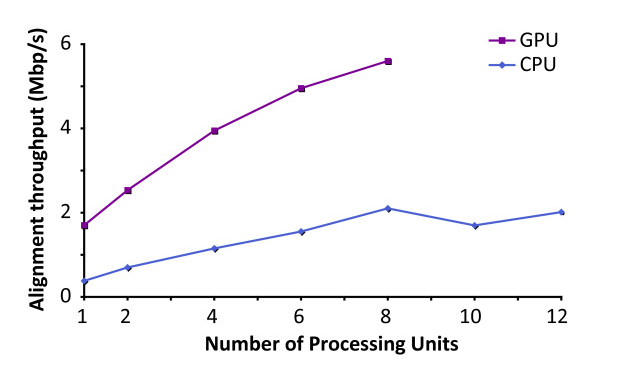


图 2.2 BarraCUDA加速BWA效果 （引自[40]）

BWT变换[22]将原字符串T的内容重新排列，形成变换后的字符串BWT(T)。BWT变换是可逆的，因此可以将变换后的字符串BWT(T)变换回原字符串T。BWT变换最初由David Wheeler在1983年提出，并由Michael Burrows与David Wheeler在1994年共同发表，因此命名为BWT变换。

后缀树数据结构存在的一个问题是，对于较长的序列，其空间占用非常大。Abouelhoda等改进了Manber和Myers提出的后缀树存储方法，这种方法被称为增强后缀数组[20]，能够提高空间利用率。Ferragina 和 Manzini提出了 FM(full-textminute-space)-Index[21]，这是一种基BWT（Burrows-Wheeler Transform)的全文本压缩索引结构，它利用BWT矩阵与后缀数组之间的关系，实现了压缩的后缀数组和索引，大幅减少了内存占用率。这种方法的过程如图2.5所示，可以简述如下：

1. 计算BWT矩阵T。规定序列S以“$”为结束符号，切“$”的字母序最小，通过轮换列出序列S所有的后缀记为数组M, 并把数组M的每行按照字母顺序排序，得到的结果即为BWT矩阵T；
2. 矩阵T中第i行在矩阵M中的行号为数组S[i]，矩阵T中最后一列的结果为BWT数组B[i]。利用UNPERMUTE算法，我们就可以根据这两个数组还原序列S。
3. 数组Occ (i, A)的含义是BWT矩阵T中最后一列第i个A在第几行，例如Occ(3,G)表示矩阵T的最后一列中第3行的G是此列的第几个G。以这两个数组为索引可以很快找到匹配点。

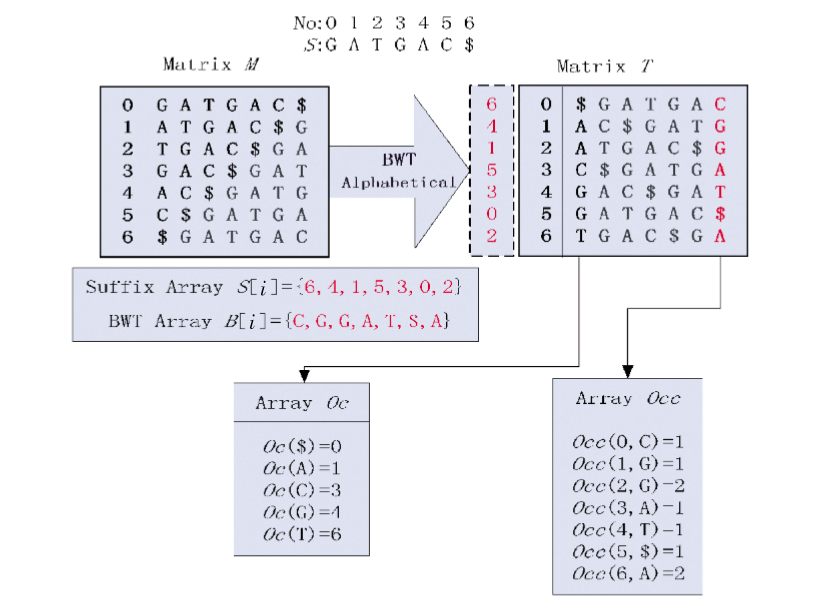


图 2.3 FM-index原理图

生物序列比对的常用软件BWA-SW、Bowtie、BWTSW等都采用了这种方法，大幅降低了内存占用。比对人类基因组3GB的数据，其内存占用约为1.3GB。

BWA是一个把低发散序列比对到一个大型参考基因组（比如说人类基因组）上去的软件包，作者是Heng Li。它由三个算法组成：BWA-backtrack， BWA-SW和BWA-MEM。

BWA-backtrack是为不超过100bp的illumina测序reads设计的。另外两个是为更长的序列，从70bp到1Mbp，而设计的。BWA-MEM和BWA-SW有一些相同的特性，比如支持长reads和剪切比对，但是BWA-MEM，也是最新的算法，是通常被推荐用来做高质量查询的，因为它更快，更准确。在70-100bp的illumina reads上，BWA-MEM也比BWA-backtrack有更好的性能。

BWA-backtrack使用前缀树数据结构保存参考基因组序列，使用backward search的方法来查找reads在参考序列上的位置。对于精确匹配，BWA-backtrack核心算法如图2.6的公式，从后向前不断缩小查询reads对应在参考序列FM-Index上的interval，直到interval为空，说明查询的序列在参考序列中不存在，或完成全部查询序列，即可根据当前的interval找到查询序列在参考序列中出现的所有位置。其中，a为查询序列当前位置，为参考序列中字母序小于的元素个数，为在中出现的次数。

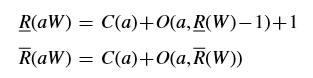


图 2.4 BWA-ALN核心算法 （引自[17]）

对于非精确匹配，采用*bounded search*方法，如图2.7所示。

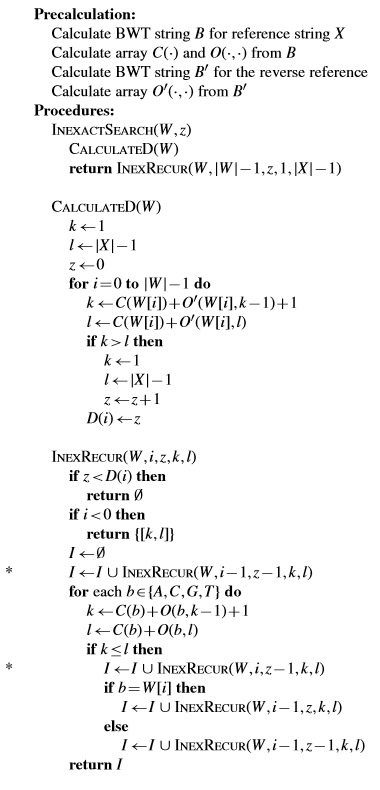


图 2.5 BWA-ALN非精确匹配伪代码 （引自[17]）

## 2.3 本章小结

本章对首先对生物序列比对做了概述，介绍了目前集中主要的生物序列比对方法，总结了当前对生物序列比对算法进行加速的几种主要方式，展示了集中常用生物序列比对软件的加速效果。接下来介绍了本文研究内容中涉及的一些基础研究背景，包括BWT与FMD-Index数据结构的原理，最后介绍了本研究的基础，BWA算法。

# 第三章 常见基因数据处理算法分析

绝大多数短序列比对算法在都是针对下一代基因测序技术生成的短序列而开发的，但是随着DNA测序技术的不断发展，测序得到的读段长度也增长了很多。这时在短序列比对中的一些处理方法就不适用于长序列了。例如，对于36bp长度的读段，可以进行end-to-end方式进行比对，根据编辑距离是否满足一定条件来决定是否返回这个比对结果。能够获得较为准确的结果。但是下一代测序仪产生的更长的读段，为序列比对带来了新的挑战。例如，对于100bp或者更长的读段，就应该在比对过程中允许较长的gap的出现，通过gap penalty来控制可能的比对结果。现有的短串比对算法在应对较长的读段时的表现并不是很好，部分算法能够提供较快的比对速度，但是精度不够，而另外一部分算法则恰好相反。BWA-MEM算法[42]正是应这种需求而出现的。该算法能够高效并精确的比对较长的读段序列。因此，BWA-MEM算法逐渐成为最流行的常用读段比对算法。但是目前还没有对BWA-MEM算法在异构平台下进行加速的研究，本章从对BWA-MEM算法的深入分析入手，尝试使用GPU来加速BWA-MEM算法。

## 3.1 BWA mem算法

随着低成本高通量下一代DNA测序技术的快速发展，目前技术上已经能够通过基因测序对遗传导致的癌症等进行较高准确率的检测。在DNA测序中，DNA被打断成很多小片段，再经由测序仪进行测序，产生数百万条很短的DNA读段。接下来对这些产生的DNA读段与参考基因组数据库进行比对，找出其中可能导致病变的基因片段。比对数百万DNA读段的过程非常耗时，所以，加速与优化DNA分析，对于DNA测序、诊断技术的发展具有非常重要的意义。

读段比对是DNA分析流程中非常核心的一个基础步骤。读段比对就是比对两条读段序列，找出其中相似的部分的过程。在读段比对中，一个短读段（长度通常小于100个碱基）需要与一个包含将近3千万个碱基的人类基因组数据进行比对。

Smith-Waterman算法是序列比对中的一个常用算法，比对两个长度分别为和的序列，其时间复杂度为。这种比对方法的时间复杂度太高，在实际的比对过程中非常耗时。因此，很多新的读段必读算法在近几年逐渐出现。绝大多数短序列比对算法都是针对下一代基因测序技术生成的短序列而开发的，但是随着DNA测序技术的不断发展，测序得到的读段长度也增长了很多。这时在短序列比对中的一些处理方法就不适用于长序列了。BWA-MEM算法正是应这种需求而出现的。该算法能够高效并精确的比对较长的读段序列。因此，BWA-MEM算法逐渐成为最流行的常用读段比对算法。

虽然对BWA-MEM的加速研究还没有出现，但是对其他读段比对算法的加速研究已经很多了。BarraCUDA[40]使用NVIDIA的CUDA平台对BWA中的第一个算法BWA-backtrack[17]进行了加速，但是BWA-backtrack算法不适用于长序列比对。

CUSHAW2[43]的seed生成环节与BWA-MEM算法比较相似，但CUSHAW2是通过计算最大精确匹配而不是超级最长精确匹配来作为seed，并且CUSHAW2使用的BWT变换的index也存在区别，需要扫描两遍才能找到全部的seed，而BWA-MEM算法采用最新的FMD-index，只需要一遍扫描即可找到全部seed。CUSHAW2目前已经有了GPU平台加速版本，名为CUSHAW2-GPU[44]。

如同其他基于前/后缀树的序列比对方法一样，BWA-MEM算法在开始进行读段比对之前，需要先对参考基因组建立索引。虽然建立索引需要较长时间，但是索引相对于同一个参考基因组是固定不变的。对于不同查询序列的比对操作，只需要建立一次索引。因此建立索引这个环节不在本文讨论范围之内，后文中默认索引已经被建立好。

BWA-MEM整体流程如图3.1所示，重点包括以下几个环节。

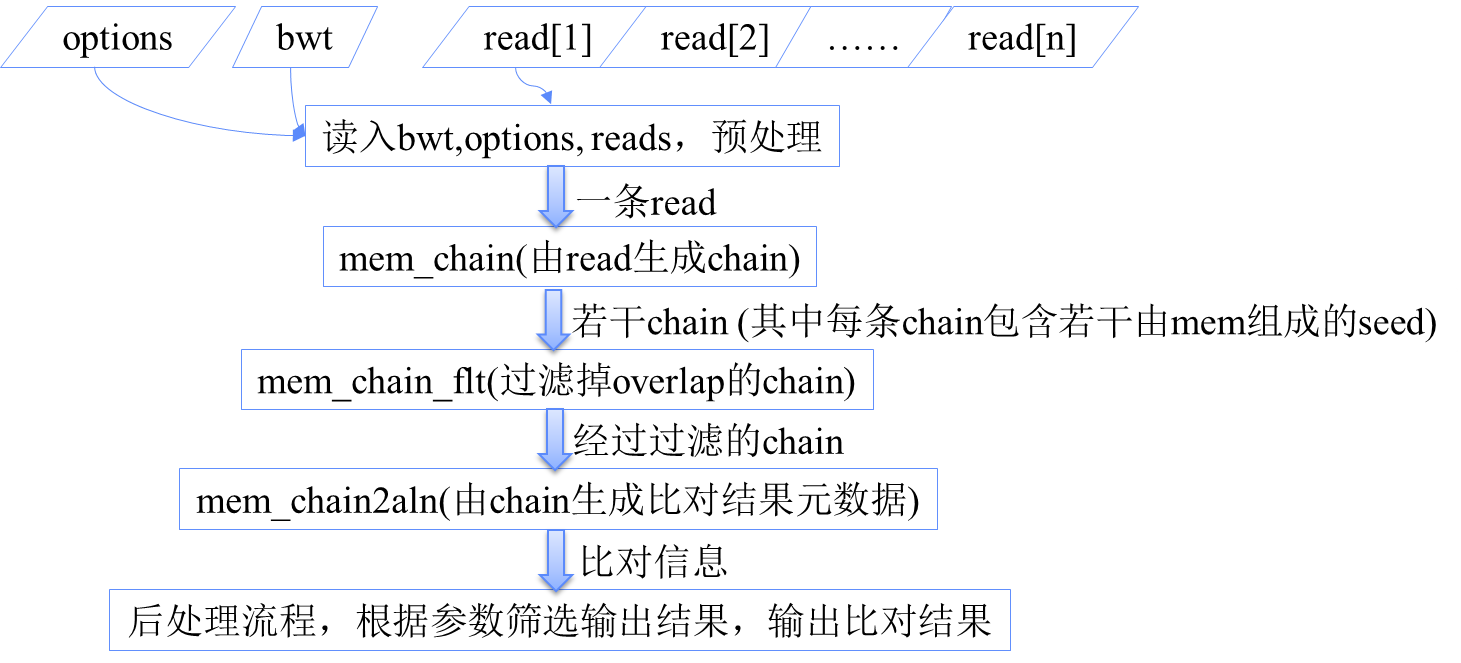


图 3.1 BWA-MEM算法流程

1. 读入索引和reads

在比对开始前需要先对参考序列建立索引，在0.7.12版本的BWA中使用的索引是作者Heng Li在2012年的论文[46]中提出的FMD-index，用以替代需要使用两个FM-index的双向BWT，FMD-index可以在一个索引中同时建立前向和反向序列的索引，可以使查找最大精确匹配（MEM）环节获得80%的加速。使用FMD-index索引的版本与原FM-index是不兼容的，如果之前使用的是早于0.6的版本，需要对参考序列重新建立索引。

BWA对于reads不是一次性全部读入的，而是分块读取。分块大小是一个可由用户指定的参数，默认为100000000个碱基。这样可以避免reads过大导致无法放入内存。BWA完成对当前分块的比对并写回比对信息之后，再开始读入下一个分块。

1. 查找SMEM

BWA-MEM算法采用经典的seed-and-extend范式，所谓seed-and-extend范式即先找到一段比较精确地匹配子段作为seed，然后采用如Smith-Waterman等算法向两侧进行扩展，直到达到比对条件的边界，得到的即为一个候选比对结果。BWA-MEM采用的seed是“超级最长精确匹配（super-maximal exact matches）”，简称“SMEM”，其定义为，读段序列中与参考基因组能够精确匹配的最长的子序列，并且这个序列不能向两边进一步的拓展，同时，这个子序列不能被其他SMEM包含。BWA-MEM所采用的寻找seed的方法是作者在[46]中提出的基于FMD-INDEX的方法，我们将在后续章节中进一步介绍该算法。在某些情况下，最优的比对结果可能并不包含任何一个“SMEM”，针对这种情况，BWA-MEM算法引入了re-seeding环节。如果得到的SMEM在index中出现了次，并且长度过长（默认参数下为28个bp），则进入re-seeding模块，在re-seed的过程中，条件加强为覆盖了上述SMEM的中间碱基，并且在index中出现了次的最SMEM。通过这种方法能够避免由于SMEM过长而导致找不到最优匹配的情况。

1. 生成Chain

在BWA-MEM算法中定义了一个Chain的概念，由部分重叠在一起的seed（不存在覆盖的seed）或距离足够近的seed组成一条Chain。Chain的意义是一方面将部分重叠在一起的seed合并后可以减少搜索空间，提高效率，另一方面可以允许在最终比对中存在少量的错配和gap，提高比对质量。并不是每一个chain在扩展后都能够得到一个比对结果。为了提高算法效率，chain生成后，还会对这些chain进行一次过滤操作。过滤的目的是去除那些在更长的chain中被大量覆盖住的、并且质量得分及长度都远不如更长的chain的那些chain。这里的chain并不要求一定是精确的。

1. 基于Smith-Waterman算法拓展Chain

在上一步中生成的Chain的基础上，为了让整个读段能够在参考序列上进行比配，需要对Chain在两个方向上进行拓展。在BWA-MEM算法中，seed拓展使用的是基于Smith-Waterman的动态规划方法。BWA-MEM算法的seed拓展方法与标准的Smith-Waterman拓展算法主要有两点区别。第一，BWA-MEM加入了启发式剪枝，这个启发式剪枝算法能够提高比对速度。第二，在seed拓展过程中，BWA-MEM会记录每一个reads末端的最优得分，如果这个得分与局部最优得分的差值小于一个阈值，那么即使局部最优得分更高，也不会被选取。BWA-MEM通过这种策略实现在局部最优与全局最优比对的自动选择。

1. 生成并输出比对结果

经过上述步骤后，可以得到比对结果的元数据。接下来BWA会根据这些元数据，并结合用户指定的参数，来生成比对结果信息，包括匹配结果、位置、注释等信息，输出比对结果。

## 3.2 Picard Mark duplicate算法

使用GPU加速BWA-MEM算法，首先需要对算法进行热点分析。BWA-MEM算法适用于70bp以上的reads比对，但是对于不同长度的reads，算法的热点有一定区别。我们将BWA-MEM算法按模块分解，并分别用100bp和250bp的reads数据进行热点测试，得到结果如图3.2和图3.3所示。

|  |  |
| --- | --- |
| **模块** | **时间占比** |
| mem\_chain  collect\_int\_v  generate\_chain  calc\_frac  tree\_resize  tree\_traverse | **64.8%**  34.4%  29.4%  0.3%  0.3%  0.4% |
| mem\_chain\_flt | 4.3% |
| mem\_flt\_chained\_seeds | 0.4% |
| chain2aln | **28.9%** |
| mem\_sort\_dedup\_patch | 0.8% |
| mem\_test\_and\_remove\_exact | 0.7% |

图 3.2 BWA-MEM算法热点分析（100bp）

|  |  |
| --- | --- |
| **模块** | **时间占比** |
| mem\_chain  collect\_int\_v  generate\_chain  calc\_frac  tree\_resize  tree\_traverse | **44.8%**  23.4%  20.4%  0.4%  0.5%  0.3% |
| mem\_chain\_flt | 2.3% |
| mem\_flt\_chained\_seeds | 0.2% |
| chain2aln | **51.9%** |
| mem\_sort\_dedup\_patch | 0.4% |
| mem\_test\_and\_remove\_exact | 0.3% |

图 3.3 BWA-MEM算法热点分析（250bp）

图 3.4 BWA-MEM算法对于不同长度reads的热点对比

由图3.4中我们可以看出，BWA-MEM算法中有两个最主要的热点模块，mem\_chain和chain2aln。其中mem\_chain负责从reads中找出超级最长精确匹配（SMEM），并由这些SMEM组成Chain。而chain2aln的作用是对生成的Chain进行双端拓展，利用改进的Smith-Waterman算法，寻找整个reads在参考序列上的最佳匹配。

随着reads长度的增加，chain2aln模块的时间占比逐渐超过了mem\_chain模块。这是由于chain2aln中使用的Smith-Waterman算法的时间复杂度是，而mem\_chain模块的时间复杂度是，显然随着n的增长，chain2aln模块运行时间的增长速度要高于mem\_chain模块。因此，对于较长的reads，chain2aln模块时间占比更大。

根据测试结果，BWA-MEM算法加速最重要的就是对mem\_chain模块和chain2aln模块进行加速。

## 3.3 GATK Indel Realigner算法

根据热点测试结果，BWA-MEM算法加速最重要的就是对mem\_chain模块和chain2aln模块进行加速。我们尝试利用GPU对BWA-MEM算法进行加速。

mem\_chain和chain2aln这两个模块是整个程序的两个最重要的热点。如前文所述，mem\_chain算法主要完成SMEM的查找和Chain的生成，而chain2aln模块则负责将Chain的两端利用Smith-Waterman算法进行拓展，找到最佳比对位置。Smith-Waterman算法的GPU加速已经有科研人员进行了深入的研究，取得了很好的加速效果。而SMEM查找和Chain的生成目前未见有GPU加速相关研究出现。因此我们重点关注mem\_chain模块的GPU加速。

mem\_chain模块的整体流程如图3.5所示。

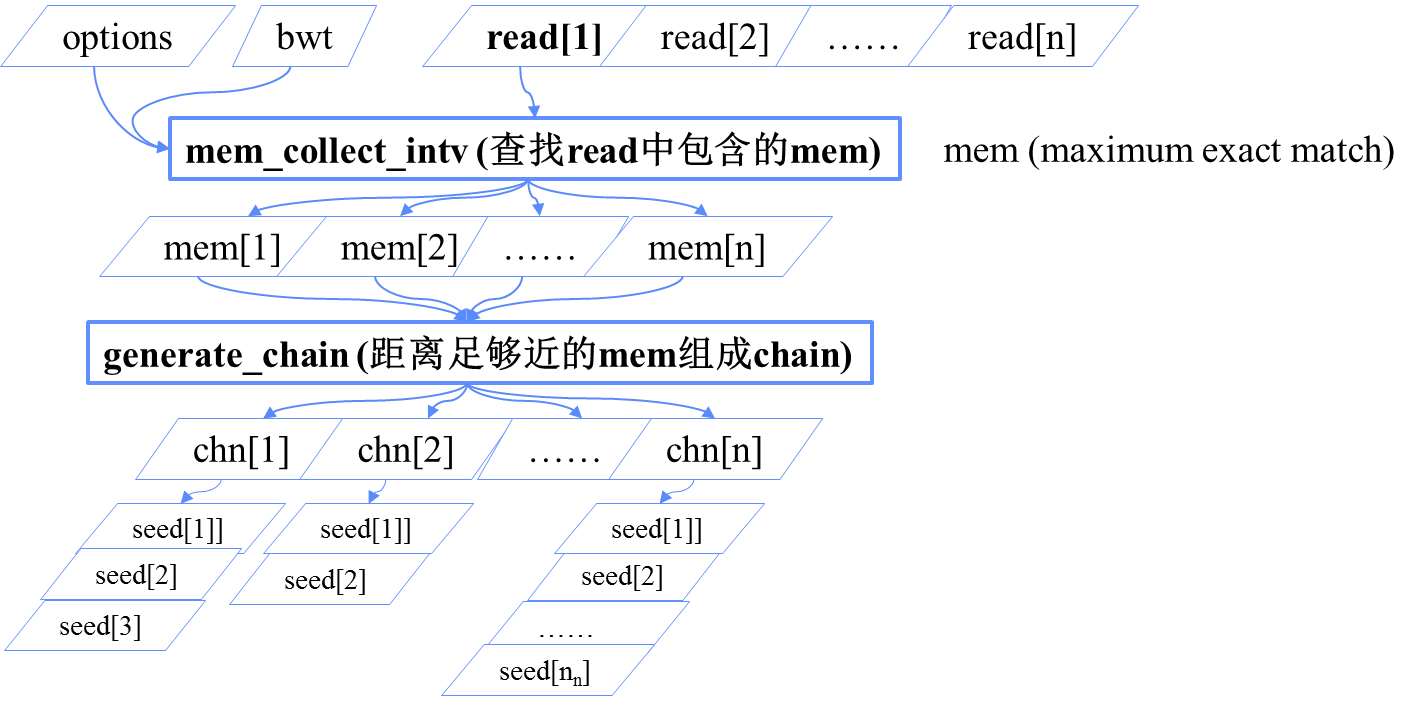


图 3.5 mem\_chain模块算法流程

mem\_chain模块中包含两个重要模块：mem\_collect\_intv和generate\_chain。其中mem\_collect\_intv模块从read中查找最大精确匹配（MEM），而generate\_chain模块将距离符合条件的mem构成chain。

由于整个算法流程要经过多个模块，为了能够使程序在GPU上有更好的性能，我们设计了图3.6所示的多kernel并行框架。GPU加速程序主要由三个kernel组成，分别对应前文中提到的三个热点模块。其中，基于Smith-Waterman的GPU加速算法已经有很多相关研究，Ernst Joachim Houtgast在[47]中完成了对chain拓展环节的GPU加速，取得了3倍的加速效果。在本文中，我们主要测是GPU对生成MEM和生成Chain模块的加速效果。

为了完成多kernel模式的移植，我们需要对原程序中的多线程模式进行重构。如图3.7所示，在原程序中，多线程分支从程序完成数据读入后就开始执行，每个线程执行整个比对流程。这种多线程模式不适用于GPU设备上进行并行。因此我们将其优化为图3.8的多线程模式。用更小的，多个kernel完成比对任务。



图 3.6 BWA-MEM多kernel并行框架

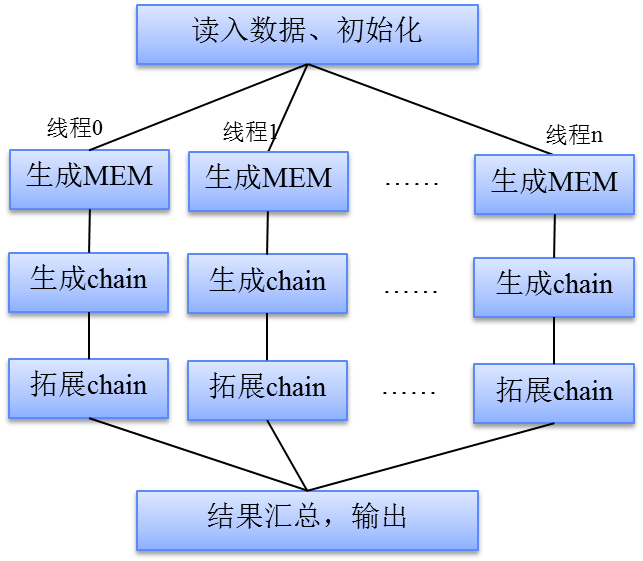


图 3.7 BWA-MEM重构前的多线程模式

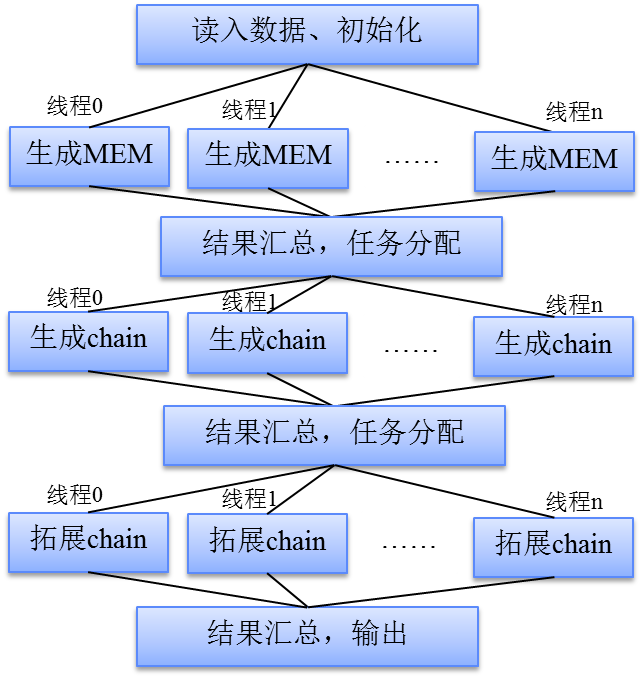


图 3.8 BWA-MEM重构后的多线程模式

任务划分涉及到两方面的工作，分别是Grid、Block维度划分和序列与线程的划分。

根据CUDA编程模型，在维度选择时优先考虑 Block 维度设计，Grid一般越大越好。Block 维度设计需要根据实际计算中每个 SM 中使用的寄存器、共享存储器数量与硬件每个SM上可用的资源量来确定。通过在编译选项中加入“--ptxas-options=-v”选项，我们可以查看每个kernel实际使用的资源数量。nvcc默认会按2.0计算能力来计算，我们需要制定参数”--gpu-architecture sm\_35”来指定按K40拥有的3.5计算能力来计算。我们需要重点关注的是寄存器数量和共享存储器的大小。结果如图3.9所示，我们可以看到mem\_collect\_intv这个kernel占用了66个寄存器，496字节的常量存储器。对于generate\_chain这个kernel占用了46个寄存器，416字节常量存储器。

ptxas info : Compiling entry function '\_Z21cuda\_mem\_collect\_intviPjP9bwtintv\_tS1\_S1\_S1\_PcPii' for 'sm\_35'

ptxas info : Function properties for \_Z21cuda\_mem\_collect\_intviPjP9bwtintv\_tS1\_S1\_S1\_PcPii

10504 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads

ptxas info : Used 66 registers, 388 bytes cmem[0], 8 bytes cmem[2]

ptxas info : Compiling entry function '\_Z19cuda\_generate\_chainiPmPjP9bntann1\_tP9bwtintv\_tPcPiiP11mem\_chain\_tP10mem\_seed\_tS6\_' for 'sm\_35'

ptxas info : Function properties for \_Z19cuda\_generate\_chainiPmPjP9bntann1\_tP9bwtintv\_tPcPiiP11mem\_chain\_tP10mem\_seed\_tS6\_

0 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads

ptxas info : Used 46 registers, 408 bytes cmem[0], 8 bytes cmem[2]

图 3.9 nvcc编译移植的BWA-MEM存储器使用情况

接着由硬件型号确定每个SM的资源，如每SM中warp总数上限、每SM上block总数上限、每SM寄存器数量、每SM上共享存储器数量。最后根据二者计算block 的维度。一般来说，维度的选择会让每个block 中的线程数量是32的整数倍，线程数量则一般是64~256。由于BWA-MEM算法对全局存储器合并访问较困难，因此 grid 和 block 维度对性能的影响也更加复杂。再结合需要在全局存储器中为每个线程预留足够的运算结果存储空间，我们选定block大小为64，grid大小为64，均为一维。

接下来是reads与线程的划分。在比对过程中reads具有天然的无关性，即不同reads之间完全独立，只需要参考序列信息，不需要其他reads的信息。因此我们采用最直观的处理方法，每个reads分配一个线程进行运算。

CUDA设备与主机端进行数据传输时，只能够对支持位拷贝的数据进行传输。也就是说，对于包含指针变量的结构体，指针的指针等数据结构，无法直接在设备和主机端进行传输。因此我们需要对数据结构进行调整，将需要在设备与主机端传输的数据选取出来，将这些数据独立分配为一块连续的存储空间，加快数据传输速度。以保存生成chain的数据结构为例，优化前后的chain存储模式分别如图3.10、图3.11所示

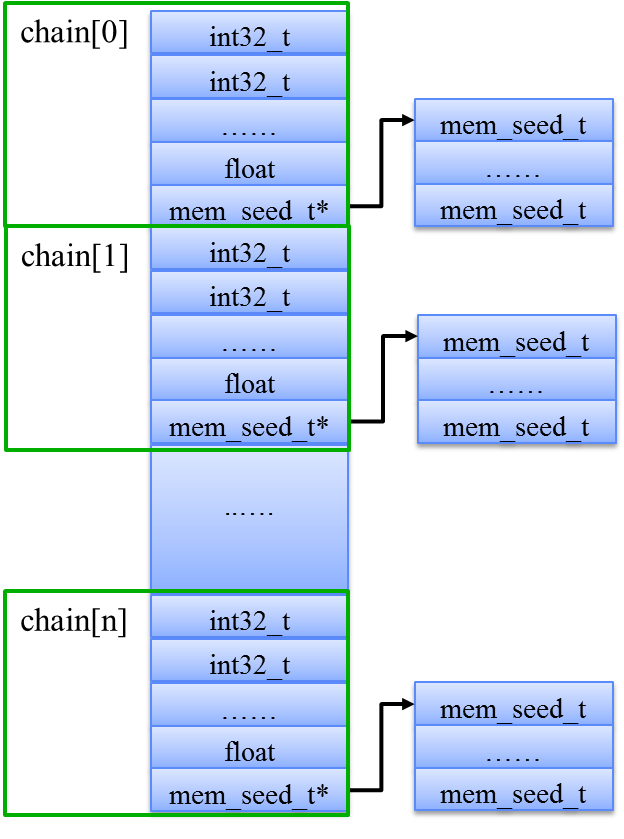


图 3.10 优化前的chain存储格式

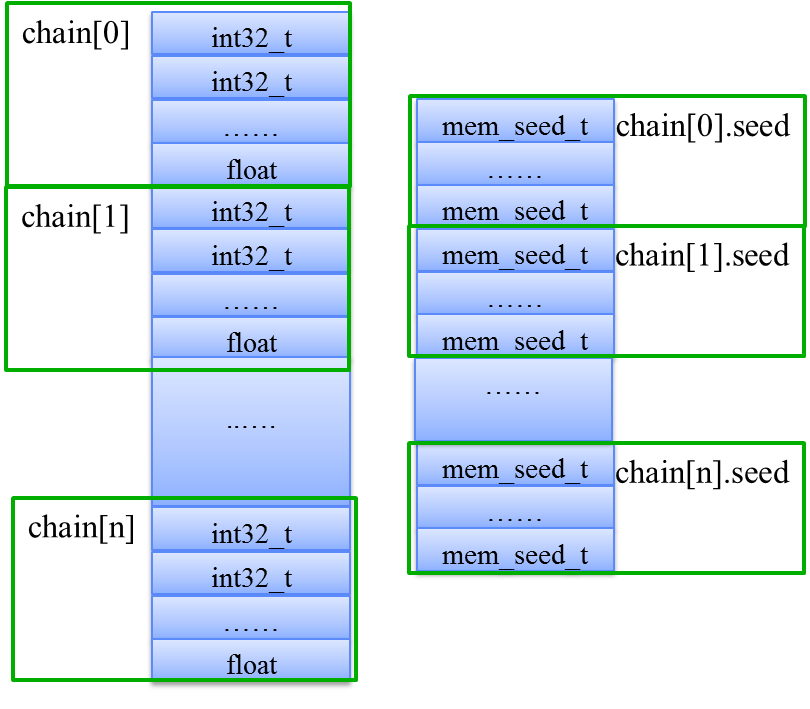


图 3.11 优化后的chain存储格式

GPU内部的6种存储器存储空间有限，因此需要根据数据的数据量和访问特点来确定其存储位置。BWA-MEM的kernel中使用的数据表3.1所示。

表 3.1 BWA-MEM传输到GPU数据的存储方式

|  |  |  |  |
| --- | --- | --- | --- |
| 变量 | 含义 | 长度/属性 | 存储位置 |
| seqs | 查询序列 | 查询序列长度，只读 | 共享存储器 |
| bwt->bwt | 参考序列BWT | 很大，只读 | 纹理存储器 |
| bwt | bwt数据结构信息 | 很小，只读 | 常量存储器 |
| cuda\_opt | 比对参数 | 很小，只读 | 常量存储器 |
| bwt->sa | 参考序列后缀数组 | 很大，只读 | 纹理存储器 |
| global\_chain | 生成的chain | 较大，读写 | 全局存储器 |
| global\_seeds | chain中的seeds | 较大，读写 | 全局存储器 |

我们将30万条不同长度特征的reads数据比对到人类参考基因组序列，并统计mem\_collect\_intv模块和generate\_chain模块运行时间，与相应模块的CPU平台运行时间对比如表3.2、表3.3所示。

表 3.2 mem\_collect\_intv模块CPU平台和GPU平台运行时间对比

|  |  |  |
| --- | --- | --- |
| 数据规模 | CPU时间（秒） | GPU时间（秒） |
| 300,000 (100bp) | 25.645 | 20.147 |
| 300,000 (200bp) | 55.126 | 54.421 |
| 300,000 (250bp) | 72.163 | 78.148 |

表 3.3 generate\_chain模块CPU平台和GPU平台运行时间对比

|  |  |  |
| --- | --- | --- |
| 数据规模 | CPU时间（秒） | GPU时间（秒） |
| 300,000 (100bp) | 22.852 | 73.174 |
| 300,000 (200bp) | 48.174 | 150.614 |
| 300,000 (250bp) | 62.051 | 201.325 |

由测试数据可以看出，对于查找MEM的模块mem\_collect\_intv，GPU平台运行时间与CPU平台运行时间近似，而生成chain模块在GPU平台运行速度则远不如CPU平台。通过分析generate\_chain模块代码，我们发现在generate\_chain模块中存在大量的不规则访存操作，而生成的mem和chain数据由于其规模较大并且重复访问较少，只能放在全局存储器中，导致该模块在GPU平台性能较差。

## 3.4 GATK Base Recalibration算法

在上一节中，我们采用GPU加速的BWA-MEM算法在实际测试中并没有取得很好的加速效果，而BarraCUDA加速的BWA-ALN算法、nvBowtie加速Bowtie2算法却取得了比较好的加速效果。接下来我们将深入比较BWA-MEM算法与BWA-ALN、Bowtie2的不同特征，研究BarraCUDA和nvBowtie的GPU加速方法，研究这些加速方法是否能够应用于BWA-MEM算法，探寻BWA-MEM算法在GPU平台加速效果受到限制的根本原因。

## 3.5 GATK Haplotype Caller算法

BarraCUDA是对BWA 0.6版本的GPU加速，只实现了BWA中的BWA-backtrack算法。其具体实现可以概述如下。

首先从磁盘中读取整个BWT参考序列，并将其传输到GPU设备的纹理存储器，BarraCUDA使用一维的uint4类型数组获得更好的存取性能。reads查询序列分块读入GPU存储器并且放入纹路存储器中加速读取。

接下来启动GPU的kernel函数，其任务划分方式是，每一条reads由一条线程负责比对，程序同时启动数千个线程并行执行。当kernel函数执行完成后，比对结果会由GPU传送回CPU，再写入磁盘。

BWA-backtrack使用的是与其他基于BWT的序列比对算法类似的后向搜索算法。对于非精确匹配，BWA-backtrack算法进行搜索的空间复杂度达到。BarraCUDA的搜索方法与BWA-backtrack有所不同。BWA使用的是时间效率上更有优势的广度优先搜索算法（BFS），如图3.12左图所示，这种方法每个线程最大需要40MB的存储空间。但是在GPU平台上，通常有上千个线程在同时运行，每个线程能够使用的存储空间非常有限。因此在BarraCUDA中使用了一种有限界的深度优先搜索算法（DFS）来解决这个问题，如图3.12右图所示。这种方法可以节省大量的存储空间。但是这种方法无法保存当前所有部分匹配的得分情况，而只能保存当前分支的得分，会使算法对于某些数据产生不同的结果。但是这种方法能够解决存储空间的占用问题，使同时运行的线程数增加，提高程序性能。

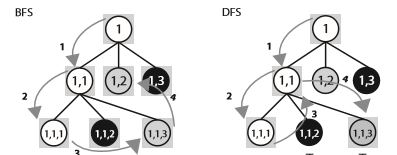


图 3.12 BarraCUDA对BWA-backtrack算法搜索方法的优化 （引自[40]）

BWA-backtrack算法与BWA-MEM算法的一个重要区别是对seed的定义不同。如图3.13所示，BWA-backtrack算法中的seed在序列最后端，长度默认为32bp。在这个位置上测序质量最高，因此最多只允许存在两个错配，不允许存在indel，比对条件更加严格。查询序列的访问顺序均为从后向前的单向访问，当比对进行到某个位置已经可以确定在参考序列中不存在这个查询序列时，比对立刻停止并开始下一条比对。

BarraCUDA针对BWA-backtrack的这种比对方式在GPU移植时做了优化，将查询序列按固定长度分段，进行多轮比对。一轮比对中只比对其中一段，每轮比对完成后，将无需继续比对的序列剔除，将需要继续比对的序列重新打包后启动kernel进行下一轮比对。这种方法减少了由于部分线程比对任务提前结束而等待其他线程造成的计算能力利用不足。

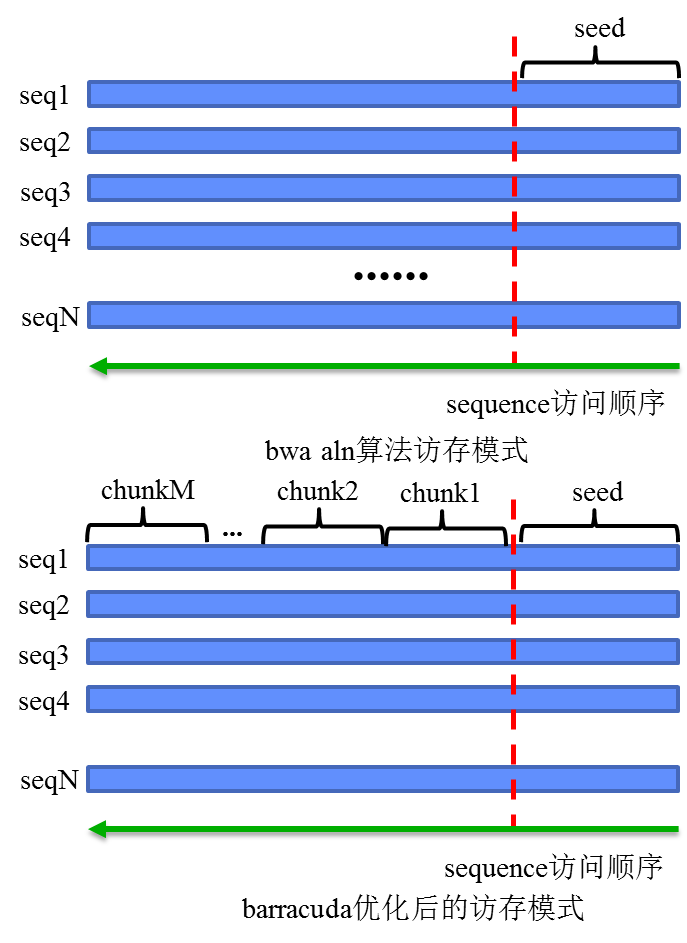


图 3.13 BWA-backtrack seeding模式与BarraCUDA优化方法

## 3.6 Mutect算法

Bowtie2采用的seeding方式与BWA-backtrack类似，采用的是固定位置的seed，默认参数是28个bp，位置是reads的高质量端，如图3.14所示。seed包含部分，位于高质量端的14bp和位于低质量端的14bp，分别被称为“hi-half”和“lo-half”。这样，对于在seed中允许两个错配的下，最终的比对结果会是以下四种情况中的一个：（1）seed中没有错配；（2）在“hi-half”中没有错配，在“lo-half”中有一个或两个错配；（3）在“lo-half”中没有错配，在“hi-half”中有一个或两个错配；（4）在“hi-half”和“lo-half”中各有一个错配。这四种情况在非seed部分都允许任意数量的错配，但是要符合质量得分的限制。Bowtie算法会根据这四种情况在前向索引和镜像索引中进行选择。无论对于哪种情况，其比对过程都是单向的。

nvBowtie结合GPU的架构特征对Bowtie2进行了重构。nvBowtie支持end-to-end比对和局部最优比对两种模式。其任务划分方式为每个线程负责一条reads。但是与BarraCUDA的动态划分不同，nvBowtie采用固定的block为96，并将其分成3个warp，每个warp有32个线程。除与BarraCUDA类似使用了纹理存储器加速bwt的读取外，nvBowtie还使用了少量shared memory用于不同warp之间的通信。nvBowtie将reads行存储改为列存储，结合比对过程为单向前进这一特点，可以加快序列读取的速度。

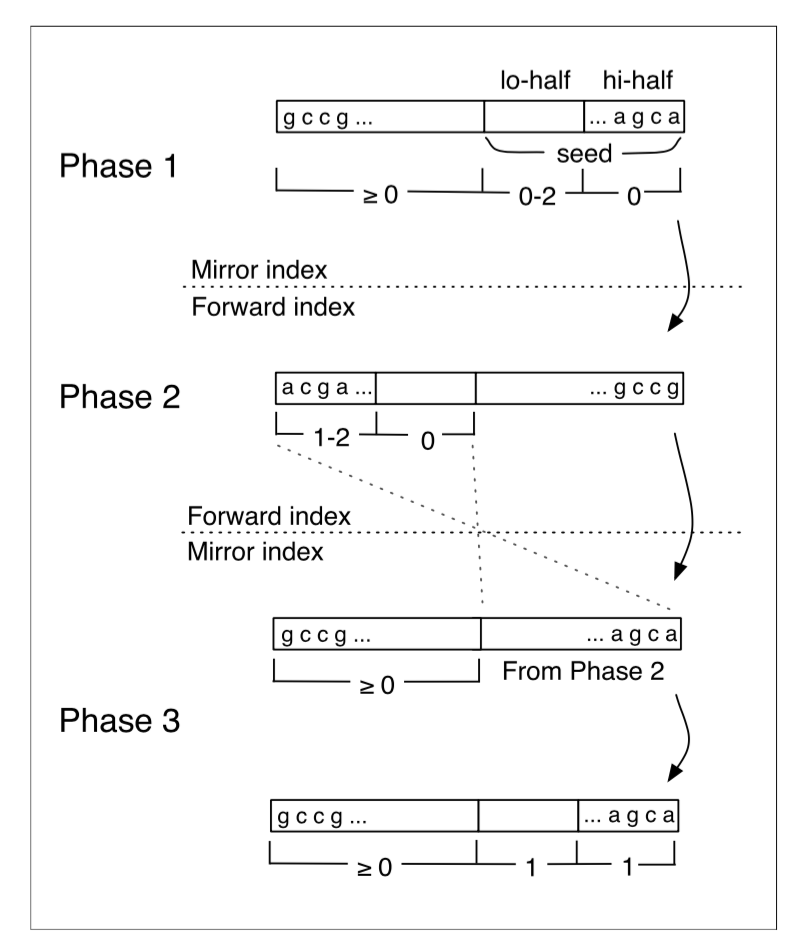


图 3.14 Bowtie2的固定seed模式 （引自[18]）

## 3.7 算法特征与并行性分析

相比于BWA-backtrack和Bowtie2中的固定位置seed，BWA-MEM算法中的seed模式则更为复杂。BWA-MEM采用的seed是“超级最长精确匹配（super-maximal exact matches）”，简称“SMEM”，其定义为，读段序列中与参考基因组能够精确匹配的最长的子序列，并且这个序列不能向两边进一步的拓展，同时，这个子序列不能被其他SMEM包含。BWA-MEM所采用的寻找seed的方法是作者Heng Li在[46]中提出的基于FMD-INDEX的方法，伪代码如图3.15所示。我们根据伪代码并用图3.16来说明BWA-MEM采用的前向-后向搜索算法寻找seed的特征。

对于每条reads，寻找包含位置的SMEM时，首先使用前向搜索，寻找以开始的右边最长的精确匹配（MEM），在搜索过程中，当*interval*的大小发生变化时，说明*interval*中的一部分不再满足最长精确匹配这一性质，将这个位置加入队列。*interval*的大小会随着前向搜索的进行而逐渐减小，直至减小到0之后，后向搜索开始，从位置向后，对于每一次拓展，依次取出队列中的每一个前向搜索终止位置，记为，在的基础上，向后进行后向搜索，重复此过程，interval持续减小。当interval减小为0，设此时搜索到位置，则为一个包含的SMEM。

对于不同的reads，其包含的SMEM长度、位置、数量都有很大的不同。当我们在GPU平台用采用每个线程处理一个reads的任务划分方式时，这会造成不同线程之间严重的不同步，包含较短SMEM的reads需要等待包含较长SMEM的reads查找完毕，而对于包含较短SMEM的reads，其包含的SMEM的个数又更多，这样包含较长SMEM的reads又反过来需要等待包含较短SMEM的reads，这种互相等待的情况导致计算资源利用率极低，加速效果非常有限。

同样，由于BWA-MEM的这种与输入数据高度相关的seeding方式，每个reads包含的seed数量、长度都会有很大不同，这又会影响下一个模块中根据seed生成chain的过程，导致生成chain的模块的不同线程同步性很差，进一步限制了加速效果。

我们尝试了多种手段来尽可能降低这种不一致对GPU平台性能产生的影响，包括用额外的空间来保存中间结果，使搜索过程能够保持单向、有序。但是由于算法的运行特征与输入数据高度相关，这些方法无法同时适合不同特征的输入数据。GPU平台对于同一warp内指令一致性的要求使BWA-MEM算法的加速效果受到了严重限制。

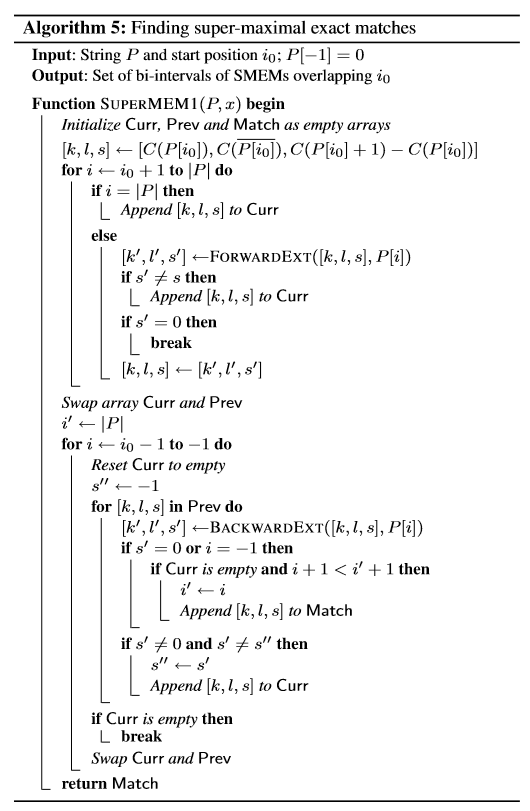


图 3.15 寻找超级最长精确匹配（SMEM）伪代码 （引自[46]）

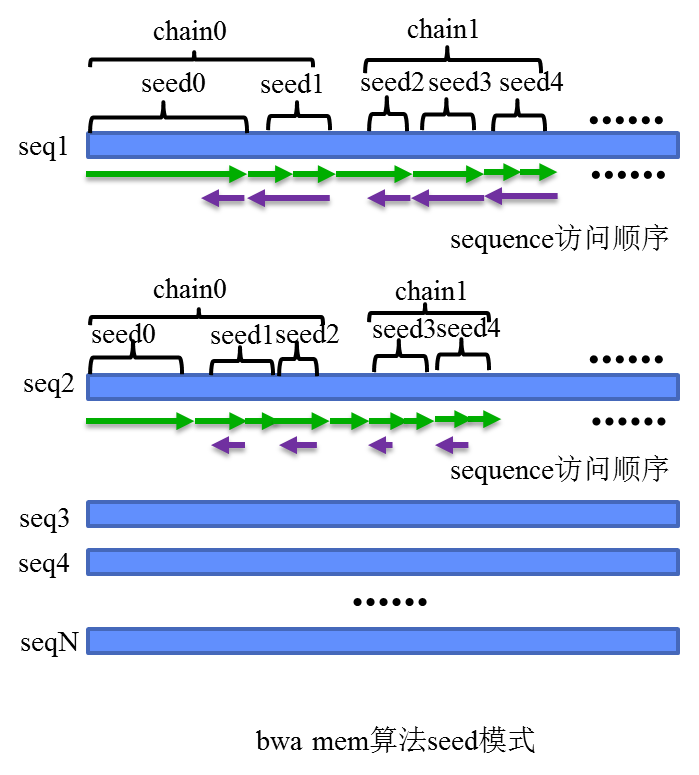


图 3.16 BWA-MEM采用的前向-后向搜索算法图示

## 3.8 本章小结

本章研究基于GPU平台的BWA加速方法与效果。首先对BWA-MEM算法的整体流程进行了分析，将整个流程划分为若干模块，并对各个模块进行了热点测试。接下来我们利用CUDA设备对BWA-MEM中的热点模块进行加速，并根据GPU的架构特点对BWA-MEM算法的流程进行调整，优化了存储空间分配与访问模式，进行了性能测试。最后我们根据性能测试结果，对BWA-MEM算法的特点进行了深入分析，与GPU加速的BarraCUDA、nvBowtie的加速效果进行对比，总结了BWA-MEM算法在GPU平台加速效果受到限制的原因。

# 第四章 基因数据分析编程框架SparkSeq设计与实现

由于BWA-MEM算法的自身特性，利用GPU对mic设备进行加速难以取得较好的效果。但是Intel众核处理器平台（MIC）的各个核心之间具有较好的独立性，能够提供更高的访存带宽，对于BWA的加速是一个更为合适的解决方案。

本章将将结合BWA-MEM算法的特点，结合Intel集成众核架构的特性，综合利用多种优化手段，对BWA-MEM算法进行了有效的加速。其在单MIC上的综合性能可以达到10-12核英特尔至强CPU的性能，单MIC设备处理能力是BarraCUDA单GPU设备的2倍。在这一章，我们将详细描述BWA在MIC上优化采用的方法。

## 4.1 整体设计

MIC众核处理器是是基于X86架构的，多核CPU所采用的并行模型都能够在MIC设备上使用，MIC卡自带一个精简的操作系统μos，因此，将CPU程序移植到MIC平台相对于移植到GPU平台要简单很多。

CPU多核和MIC协同计算有以下几种模式：

1. Multi-Core Hosted：等同于只使用CPU多核运算；
2. Offload：加载模式，特征是在主函数在CPU端发起，在CPU端调用MIC端的kernel在MIC上之行；
3. Symmetric：对等模式，即CPU多核和MIC都有自己的主函数，通过网络协同工作；
4. Reverse Offload：逆转加载模式，主函数由MIC发起，CPU辅助运算；
5. Many-Core Hosted：native模式，仅使用MIC设备执行程序，CPU空闲。

在实际中Offload模式和Native模式使用较多。相比之下，Offload模式更为常见，因为在该模式下，可以只将计算最为密集的部分函数转移到MIC平台执行，充分利用MIC设备的运算能力，而非计算密集的部分，继续保留在CPU端执行。这种划分方式的依据是，MIC设备对于非计算密集程序如分支预测能力相对于CPU有较大劣势，MIC设备的单核运算能力也远远不如单核CPU的运算能力。采用Offload模式可以使计算密集部分程序充分利用MIC众核处理器的特性，同时非计算密集部分保留在CPU端执行。但是Offload模式需要在MIC端与CPU端进行大量数据传输，其MIC上执行的部分类似于CUDA中的核函数，每次调用MIC设备都需要经历数据传入——计算——数据写回三个步骤。如上一章中所述，这种数据传输需要占用大量时间，并且需要对原数据结构进行大量修改，结合BWA-MEM的算法特性，并行运算从读入reads后就开始了，一直持续到当前reads处理结束，整个运算过程处于一种高并发的模式，因此Offload模式相对于Native模式不会有明显的性能优势。综合考虑程序可靠性与性能收益，本章采用Native模式对BWA-MEM算法进行加速。

Native模式是仅在MIC卡上执行程序的模式，需要在编译时加入*-mmic*选项，其编译出的代码仅能在MIC设备上执行，不能在CPU上执行。MIC端不支持SSE，AVX-256等向量代码，只支持512bit的AVX-512向量代码。MIC设备上运行着一个精简版的Linux操作系统，μos。要将一个CPU程序移植到MIC Native模式运行，需要以下几个步骤。

1. 判断程序是否适合使用Native模式运行；
2. 在CPU端用Native模式编译源程序；
3. 在CPU端用Native模式编译程序依赖的库；
4. 将编译好的二进制可执行程序以及依赖的运行库的二进制程序，拷贝到MIC设备上；
5. 如果程序的输入输出数据过大无法放在MIC设备的存储空间中，可以将数据存储在Host端。这样需要将Host端的目录Mount到MIC设备上。
6. 通过控制台登陆到MIC设备的操作系统，配置运行时的环境变量等，建立好运行时需要的库的路径，然后启动程序。

经过上述步骤，即可将CPU端的bwa程序移植到MIC端运行，但是上述步骤仅能够实现一个在MIC设备上运行的bwa基础版本，其性能上并没有充分利用MIC设备强大的并行运算能力。因此，在基础版本的基础上，我们进行了扩展性优化、多级流水线优化、数据对齐与自动向量化优化、手工向量化优化等。最终，我们完成了一个经过深度优化的适用于MIC平台的BWA，其对250bp碱基长度的reads序列的处理能力与12核至强CPU处理能力相当，对100bp的短序列的处理能力可以接近barracuda在Nvidia K40平台的两倍。

## 4.2 API设计

由于MIC也是X86架构的处理器，因此能够在CPU端执行的程序通过比较简单的步骤移植到MIC端。在参照Intel MIC产品的相关手册完成诸如MIC设备驱动安装、MPSS软件栈安装、MIC端与Host端环境配置、Host端编译器安装之后，对于BWA的CPU平台代码，还需要经过以下几个步骤将其移植到MIC平台。

首先，需要正确配置编译器的环境变量。Intel Composer XE提供了一个自动配置脚本，通过运行这个自动配置脚本可以快速完成编译器所需的环境变量配置。这个配置脚本的路径在最新版的Intel Composer XE 2016中已经Intel官方网站上提供的路径有一些不同。

source ~/opt/intel/compilers\_and\_libraries/linux/bin/compilervars.sh intel64

其中路径应为对应机器上Intel Composer XE的安装路径。接下来可以用*icc -V*验证一下icc是否正确配置。

$ icc –V

Intel(R) C Intel(R) 64 Compiler for applications running on Intel(R) 64, Version 16.0.2.181 Build 20160204

Copyright (C) 1985-2016 Intel Corporation. All rights reserved.

编译器环境检查无误，就可以使用*icc –mmic*将代码编译成MIC平台的可执行程序。

BWA使用zlib来压缩存储reads数据。在Linux环境下编译CPU端代码时，Linux系统一般都提供了zlib库可供调用，因此无需单独编译。但是使用MIC native模式时，zlib库没有编译好的二进制版本可供MIC端使用，需要下载zlib库的源码，将其编译成能够在MIC端运行的二进制代码。

编译MIC端zlib库的方法与编译BWA源码类似，只需使用icc编译器在编译选项中加入*-mmic*，即可产生适用于MIC端的代码。zlib库的编译选项等细节在zlib项目页面<http://www.zlib.net/>有详细的描述，在此不再赘述。在BWA中，默认是将zlib库作为静态链接库来使用的。

MIC设备上运行着精简的Linux操作系统，其在设备内存中构建了一个虚拟的文件系统，可以存储程序运行所需的数据文件。该文件系统可以使用ssh和scp通过网络与Host端进行文件传输。

在默认配置中，MIC设备驱动程序会为每个MIC设备分配一个名字和IP地址，用户通过名字或IP地址访问MIC设备。在默认配置中，用mic0表示第一个MIC设备，用mic1表示第二个MIC设备，以此类推。

以使用第一个MIC设备mic0为例，首先需要通过ssh登录到MIC设备的系统中。

$ ssh root@mic0

登陆成功后可以在MIC设备上进行创建文件、文件夹等操作。

$ mkdir /tmp/reference

接下来可以将运行时依赖的动态库和运行时需要的数据传送到MIC设备的文件系统中。BWA默认使用静态库，因此不需要传输动态库，只需把参考基因组数据传送到MIC设备上。

$ scp /tmp/reference/human.fasta\* root@mic0:/tmp/reference

值得注意的是，MIC设备上虚拟的文件系统空间是有限的，并且虚拟文件系统使用的存储空间与MIC设备的内存是共享的，因此当参考基因组数据很大时，程序运行时会因内存不足导致程序异常退出。对于这种情况有两种解决方法，其一是如下一小节中所述将较大的文件放置在Host端存储，通过NFS文件系统来访问数据。另一种方式是将数据分段读取后，放置在shared memory中，在计算时直接从shared memory读取数据，而不是从文件系统中读取数据，这种方法只要参考基因组保持不变，在每次启动比对时就无需重新读取参考基因组数据。这两种方法对于很大的人类全基因组数据具是非常必要的。

由于MIC设备存储空间非常有限，并且需要与程序运行时的内存共用，因此对于空间占用非常大但是非频繁存取的数据，可以将其放在Host端的文件系统中通过NFS来进行访问。在序列比对中，参考序列是保持不变的，并且在程序开始时会被一次全部读入内存，因此当参考基因组数据过大导致将其放在MIC端程序无法执行时，可以用这种方法来解决存储空间占用问题。

经过上述步骤，BWA已经能够在MIC设备上以Native模式运行。我们采用的MIC设备具有60个核心，由于在Native模式下需要保留一个核心运行设备上的操作系统，所以我们实际使用的线程数为118个线程来进行测试。我们通过测试100万条100bp长度的序列比对到人类基因组数据hg19所需要的运行时间，来测试基础版本的性能，结果如图4.1所示。

图 4.1 MIC平台基础版本加速效果

从结果中我们可以看出，通过将BWA移植到MIC平台使用Native模式运行，在没有进行进一步优化的情况下，其比对时间接近4核CPU的比对时间，其加速效果远远好于GPU基础版本的加速效果。这一方面是由于MIC设备使用与CPU相同的x86架构，在CPU端BWA是一个使用了大量优化手段的高性能程序，这些优化手段被移植到MIC设备上时，同样具有很好的加速效果。但是由于GPU架构与CPU架构的巨大差异，这些优化手段在GPU端并不适用，甚至反而会严重影响GPU程序的性能。另一方面，由于BWA-MEM算法自身的特性，以reads为划分方式进行多线程并行时，不同reads之间完全独立，这种独立性使其更适合在同样各个核心独立性很强的MIC设备的不同核心上运行，而不会由于线程间的不同步导致在GPU设备上出现的大量停顿。综上所述，BWA-MEM由于其算法的特性更适合用MIC设备来加速。在后续章节中，我们将通过多种优化手段，将MIC版本的BWA性能再提升一倍以上。

## 4.3 流程执行引擎设计

基础版本只是对BWA向MIC平台的移植的可行性进行了验证，接下来我们需要对其进行更进一步的优化以充分利用MIC设备的计算性能。我们再对基础版本重新进行一次热点分析。首先我们将整个比对过程分成三个阶段，（1）读取reads数据，（2）将reads比对到参考基因组，（3）将比对结果写回。我们选取了一些典型的数据集，分别统计这三个阶段的运行时间占比。结果如图4.2所示。

图 4.2 MIC平台BWA各阶段运行时间对比

可以看到，reads的读取与比对结果的写回所占用的时间平均达到36%左右，在这两个阶段程序主要进行数据传输和I/O操作，而计算部件利用率很低。在比对阶段则主要是使用设备的计算能力，而I/O则为空闲。对于输入输出数据较大而必须存放在Host端通过NFS方式访问时，在实际测试中我们发现用这种方式读取Host端磁盘上的数据的带宽只有10Mbps左右，读取较大数据集会非常缓慢，其非计算部分时间占比会上升至40%以上。因此，采用流水线来隐藏数据读入和写回时间是非常有必要的。

为此，我们采用三级流水线来隐藏数据读入和写回的时间。在原程序中，如图4.3所示，序列比对是按照读取——比对——写回的顺序串行进行的。采用三级流水线，如图4.4所示，将数据的读取与写回与计算过程并行起来，可以隐藏阶段的读取和写回数据的时间。在三级流水线模式下，程序运行时间等于全部计算时间与第一段读取数据和最后一段写回数据时间之和，中间各段的读取和写回时间被隐藏了。

读取

比对

写回

比对

写回

读取

…

图 4.3 BWA无流水线执行模式

读取

比对

写回

比对

写回

读取

…

读取

比对

写回

读取

图 4.4 BWA在MIC平台上三级流水线执行模式

虽然读取和写回阶段主要是进行I/O操作，但是在诸如解压缩序列时依然需要一定的计算资源，并且需要读写内存，会与计算阶段的访存操作产生资源竞争，降低比对的速度，因此我们将这种影响力争降到最低。我们对三级流水线进行一个优化，保证在同一时刻读取和写回这两个操作只有一个在进行。因为读取与写回时间之和一般小于数据的比对时间，所以加入这个限制后总时间依然等于全部比对时间与第一段读取时间、最后一段写回时间之和。但是可以降低读取和写回对比对时间的影响，在一定程度上缩短比对时间。如图4.5所示。

读取

比对

写回

比对

写回

读取

…

读取

比对

写回

读取

图 4.5 优化后的三级流水线处理模式

我们使用前文中采用的数据集对流水线优化进行性能测试。图4.6是未使用流水线的原BWA程序执行的真实时序图。红色的计算环节中间夹杂着读取和写入操作。图4.7是使用流水线优化后的程序执行真实时序图，红色的计算环节已经连在一起，与读取和写回重叠。

图 4.6 无流水线时的程序执行时序图

图 4.7 优化后的三级流水线程序执行时序图

对比程序对同一数据集的比对时间，从图4.8中可以可以看到引入流水线后，读取和写回的时间基本上被隐藏，对前文中的数据集，比对时间缩短了37.5%，与CPU平台对比，其性能已经接近8核CPU的性能。

图 4.8 采用三级流水线优化后性能测试结果

我们对三级流水线中的比对环节进行了进一步性能测试，发现在比对环节的计算部件占用率在比对环节的前60%时间里保持在一个非常高的水平，而在最后约40%的时间里计算部件利用率较低。我们进一步分析代码发现，这部分计算资源利用率较低的代码主要是在生成结果数据，以访存代码为主，计算代码量较少。因此，我们尝试将比对环节再分成两个阶段，成为比对阶段和生成阶段，采用四级流水线来尝试将比对与生成阶段重叠，如图4.9所示。

读取

比对

写回

读取

生成

比对

写回

生成

读取

比对

写回

生成

…

读取

图 4.9 四级流水线优化设计

我们采用多个数据集测试四级流水线的性能，将其与三级流水线的性能进行对比。我们发现四级流水线的性能并没有获得性能提升，反而导致性能下降了约20%~30%。结合数据分析其原因，虽然生成环节的计算模块利用率很低，但是却存在大量的访存操作，而访存操作会影响比对环节的执行时间，如同将比对与读取和写回重叠后，比对时间变长一样，比对环节与生成环节重叠后，导致比对时间进一步变长，其重叠获得的性能收益不足以弥补其带来的性能损失，导致了整体性能的下降。在后续工作中，如果能够更细致的分析不同阶段的访存特征，对访存进行进一步优化，以及进一步合理的划分比对和生成环节的线程数量，更合理的利用计算资源，相信四级流水线性能会有更大的提升。

## 4.4 算法实现

Intel MIC设备上有众多的物理计算核心，其每个计算核心上最多可开4个线程，最佳线程数量一般是每个核设置2~4个线程。但并不是MIC卡上的线程数量越多越好，线程数量越多，线程开销也更大。通常任何程序的扩展性都是有限的，在CPU平台上，使用Intel Xeon E5-2670处理器，16个物理核心，当BWA-MEM采用8个以上的线程运行时，性能不再线性增长，在12个线程时已经达到性能极限，继续增加性能反而会导致性能下降。

### 4.4.1 BWA mem算法实现

### 4.4.2 Mark Duplicate算法实现

### 4.4.3 Indel Realigner算法实现

### 4.4.4 Base Recalibration算法实现

### 4.4.5 Haplotype Caller算法实现

### 4.4.6 Mutect算法实现

MIC拥有60个以上的物理核心，最多支持200多个线程的并行。因此，我们首先测试基础版本在MIC设备上的扩展性。在Native模式下需要留一个核心供MIC设备上的Linux操作系统使用，因此，我们使用59个物理核心，分别测试BWA-MEM在59、118、177、236个线程时的性能。结果如图4.10所示。可以看到，当采用118个线程，即每个物理核上运行两个逻辑线程时，程序性能已经达到最大，继续增加线程数，性能反而会有轻微下降。因此，我们采用118个线程来运行BWA-MEM程序。

图 4.10 BWA-MEM扩展性测试

## 4.5 算法与流程优化

对于Intel Xeon Phi协处理器来，所谓向量化，其实就是指使用Intel的SIMD（Single Instruction, Multiple Data）指令集，利用向量部件，来实现一条指令处理多个数据，来提高程序性能。根据数据流和指令的概念将计算机的体系结构划分为四种基本类型，即SISD（单指令单数据流）、MIMD（多指令多数据流）、SIMD（单指令多数据流）、MISD（单指令多数据流）。

### 4.5.1 算法内部实现：Shuffle操作避免

### 4.5.2 流程间优化：冗余计算发现与消除

传统的计算机都是串行执行，在同一时刻只能执行一条指令（只有一个控制流）、处理一个数据（即只有一个数据流），因此被称为单指令流单数据流计算机Single Instruction Single Data（简称SISD）。向量计算机的最重要的特征是具有能进行向量计算的向量化处理单元。在执行向量操作时，一条指令可以对一个向量，即多个数据进行运算，这就是单指令流多数据流（Single Instruction Multiple Data，SIMD）的概念。



图 4.11 SIMD向量部件示意图

如图4.11所示，利用SIMD指令处理加法操作时，在一个时钟周期内可以处理多个数据，示意图中向量寄存器可以保存4个数据，即一次性可以处理4个数据的加法操作，而对于单指令单数据的操作，需要4个时钟周期来完成这一操作。可以看到，如果充分利用处理器的向量处理能力，那么就可以获得不错的性能提升。

在Intel的处理器系列中，很早就加入了对SIMD的支持。从MMX到SSE，再到AVX，以及最新推出的Intel Xeon Phi中加入的AVX-512指令集。Intel Xeon Phi所使用的AVX-512指令级，具有512位向量处理器，对于32位的float形或者是int32类型数据，可以一次处理16个数据。对于64位的double或int64类型的数据，可以一次处理8个数据。同时，少部分指令还支持以8位或者16位的粒度进行运算。如果能够充分发挥SIMD向量指令的优势，可以使Smith-Waterman拓展部分的性能获得进一步的提升。

Intel Xeon Phi的向量化可以通过两种方式来实现。我们以数组求和的代码分别来演示。第一种是利用编译指导指令指导编译器进行向量化。Intel的icc编译器可以自动分析循环中的数据依赖关系，如果编译器可以确定没有数据依赖，会对代码进行自动向量化，产生向量化后的代码。而对于编译器无法确定的数据依赖关系，编译器不会进行自动向量化。这时候就需要用户通过编译指导指令来指导编译器进行自动向量化。具体来说，一种方法是可以通过*#pragma ivdep*预编译指令，来指示其后面的循环中不存在数据依赖关系，这样编译器就可以对代码进行自动向量化。另一种方法是使用*#pragma simd*预编译指令强制编译器将该预编译指令后的代码进行向量化。这两种编译指导语句没有本质区别。以*#pragma simd*为例，欲实现一个经过向量化优化的求和操作可以通过以下代码来实现。

float sum\_auto (float \*buf, int N) {

float sum = 0;

**#pragma simd**

for (int i=0; i<N; i++) {

sum += buf[i];

}

return sum;

}

图 4.12 指导向量化代码实例

使用icc编译上述代码，并指定编译选项产生向量化报告“*-vec-report 3*”，通过编译器输出的内容可以看到代码被向量化了，还可以从输出的编译报告中可以看到循环部分的代码被向量化的细节信息。通过生成的汇编代码，也可以看到代码中使用了simd指令。使用制导语句进行向量优化非常方便，对程序的编写者难度不大，但是当循环中有数据依赖时，无论是编译器自动向量化还是利用制导语句进行向量化，都可能会造成计算结果的错误。因此，对于有数据依赖的情况，这种向量化的方式是不可取的，需要通过其他技术手段来进行向量化。

第二种实现向量化的方式是通过Intel提供的向量化函数进行向量化，在代码中直接使用AVX-512指令。使用AVX-512等指令时，需要注意的一个问题就是内存地址对齐，对于512位的AVX-512向量寄存器来说，它一次性读入并处理16个int32型数据，要求数组的首地址必须按64字节（即512位）对齐。手工向量化难度比较大，但是它可以处理编译器向量化不能处理的问题，例如上文提到的数据依赖问题。手工向量化更灵活，通过simd优化，我们可以最大限度的挖掘处理器的单核处理能力。性能甚至可以成倍的提升。

具体到BWA-MEM的MIC版本进行优化的过程，对于数据独立性比较明显的循环，icc编译器在默认参数编译时会对这些循环进行自动向量化。在编译选项中加入“*-vec-report 3”*，编译后生成源程序文件名对应的.optrpt文件。我们通过查看这个文件的内容，保证热点循环能够被充分向量化，如图4.13所示。

LOOP BEGIN at bwt.c(59,2)

remark #25096: Loop Interchange not done due to: Imperfect Loop Nest (Either at Source or due to other Compiler Transformations)

remark #25451: Advice: Loop Interchange, if possible, might help loopnest. Suggested Permutation: ( 1 2 ) --> ( 2 1 )

remark #15389: vectorization support: reference bwt has unaligned access [ bwt.c(63,3) ]

remark #15381: vectorization support: unaligned access used inside loop body [ bwt.c(63,3) ]

remark #15300: LOOP WAS VECTORIZED

remark #15451: unmasked unaligned unit stride stores: 1

remark #15475: --- begin vector loop cost summary ---

remark #15476: scalar loop cost: 33

remark #15477: vector loop cost: 10.750

remark #15478: estimated potential speedup: 3.030

remark #15479: lightweight vector operations: 170

remark #15488: --- end vector loop cost summary ---

remark #25018: Total number of lines prefetched=2

remark #25019: Number of spatial prefetches=2, dist=8

remark #25021: Number of initial-value prefetches=2

remark #25139: Using second-level distance 2 for prefetching spatial memory reference [ bwt.c(63,3) ]

remark #25015: Estimate of max trip count of loop=16

LOOP BEGIN at bwt.c(61,3)

remark #25436: completely unrolled by 4 (pre-vector)

LOOP END

LOOP END

图 4.13 icc编译器向量化报告示例

图4.13就是一个典型的编译器向量化结果报告。其中estimated potential speedup是编译器根据对标量循环的代价和向量化循环的代价估算出的向量化加速效果。报告中不仅包含编译器是否对循环成功进行向量化，还会包含向量化指令中的访存对齐情况，如报告中提示bwt数据结构没有采用对齐的访问模式，我们在接下来的数据对齐环节会对这些未对齐的访存进行优化。

但是，并不是所有应该被向量化的代码都能够被编译器自动向量化，对于编译器无法确定数据依赖关系的循环，编译器不会进行向量化。这时就需要我们进行手工干预，对于比较简单的循环，我们可以通过加入*#pragma ivdep*预编译指令来说明数据独立性情况。但是，对于像Smith-Waterman等存在明显数据依赖而又是程序设点的代码段，就需要如下一节所述，通过调整算法并使用Intel提供的向量化函数来手工进行向量化。

在使用SIMD的过程中需要注意很多问题。上文中提到，使用AVX-512等SIMD指令要求内存按照一定的规则对齐。所谓内存对齐是指强制编译器在内存中特定大小的边界上创建存储空间。这样可以提高数据加载与存储的速度。对于非对齐的数据，处理器可能需要两次加载操作才能完成数据的加载，对于数据写回存储也是如此。而对于对齐的数据加载和写回只需一次内存操作。采用Intel MIC架构的Intel Xeon Phi处理器，其存储读取与写入都是按照64字节边界来优化的，因此也应该指导编译器在512位（64字节）的倍数上为边界来创建数据空间。

另一方面，在默认条件下，编译器在访问数据时无法指导并且不会假设其访问的数据是按照64字节对齐的，所以我们需要通过预编译指令来告诉编译器数据的对齐情况，使编译器产生优化后的代码。具体来说，数据对齐需要进行两部操作：

1. 对齐数据；
2. 在程序中对性能存在重要影响的部分使用编译指导指令提示编译器数据的对齐情况。

对齐数据有两种方法，一种是使用编译器提供的内存分配函数或修饰符，由编译器完成存储空间的对齐分配。另一种方法是自己编写内存分配回收函数，在分配空间中多分配一些，然后从特定的字节开始使用空间。

编译器对于静态分配和动态分配按特定字节对齐的空间所使用的关键字是不同的。对于静态分配的数组，使用\_\_attribute\_\_((aligned(64)))来指定。

float A[1000] \_\_attribute\_\_((aligned(64)));

对于动态分配数据，需要使用数据对齐动态分配函数\_aligned\_malloc()，\_mm\_malloc() 和\_mm\_free()来进行动态分配和释放。

buf = (char\*) \_mm\_malloc(bufsize, 64);

\_mm\_free(buf);

但是对于C/C++来说，只是采用内存对齐的动态分配方式是不够的，在使用动态分配的数据之前，还需要使用*\_\_assume\_aligned(a, 64)*来说明相应的变量的内存对齐方式，这样编译器才能为其产生高效的代码。

在BWA-MEM算法中，向量指令访问频率较高的数据包括参考基因组bwt、bwt查找过程中不断变化的interval以及各种辅助数据结构，还包括下一节Smith-Waterman向量化中所使用的得分矩阵等，为了能使向量化获得更高的性能，这些数据结构都需要以64字节为单位进行对齐。

在引入pipeline的基础上，我们进行了指导向量化与数据对齐优化后，对于同一个典型测试数据，在默认比对参数下，其比对时间缩短了14.6%，如图4.14所示。

图 4.14 数据对齐与指导向量化加速效果

Smith-Waterman算法是用于计算局部最优匹配的动态规划方法。在BWA-MEM算法中，基于Smith-Waterman算法用于对每个seed进行双向拓展。在BWA-MEM的Single-end比对算法中，所有seed以其所属的chain的长度为第一关键字，以其自身长度为第二关键字进行排序，由长度最长的seed开始逐个进行拓展。当拓展到某一个seed时，如果这个seed已经被之前找到的成功比对的区间覆盖，则直接跳过这个seed不再拓展，否则将使用一种有限界的gap罚分动态规划方法进行拓展，来寻找下一个可能存在的比对信息。

BWA-MEM算法的seed拓展方法与标准的Smith-Waterman拓展算法主要有两点区别。第一，BWA-MEM加入了启发式剪枝，当Smith-Waterman拓展进行到参考基因组的位点，reads的y位点时，如果在点的最优匹配结果大于，则立刻停止继续拓展，其中是一个可调节的剪枝参数，是gap extension的罚分。这个启发式剪枝算法能够提高比对速度。第二，在seed拓展过程中，BWA-MEM会记录每一个reads末端的最优得分，如果这个得分与局部最优得分的差值小于一个阈值，那么即使局部最优得分更高，也不会被选取。BWA-MEM通过这种策略实现在局部最优与全局最优比对的自动选择。

BWA-MEM中利用Smith-Waterman动态规划方法以reads作为查询字符串将其比对到参考基因组序列上。定义查询串，对应的参考基因组序列为，定义为将reads序列中的替换为所需代价的矩阵。对于新出现一个gap和延续一个已有的gap的罚分分别定义为和。Smith-Waterman的核心计算过程需要三个矩阵，矩阵代表在reads末端以gap结尾的得分矩阵，矩阵表在reference末端以gap结尾的得分矩阵，为整体得分矩阵。这三个举着的递推关系可以用以下几个公式表示。

在BWA-MEM算法的代码中，涉及到Smith-Waterman的部分有*ksw\_extend2*，*ksw\_global2*，*ksw\_i16*，*ksw\_u8*这四个函数。其中前两个函数会在single-end模式中被调用，原代码没有进行SIMD优化，而后两个函数会在pair-end模式中被调用，原代码使用SSE2进行了SIMD优化，但是在MIC设备上不支持除AVX-512指令集外的向量指令，而SSE2指令相对于AVX-512有一些显著区别，因此需要对这部分代码进行调整和优化。

根据上述得分矩阵的计算公式，对齐进行向量化需要首先分析公式中存在的数据依赖关系。、、矩阵计算过程中的数据依赖关系如图4.15所示。

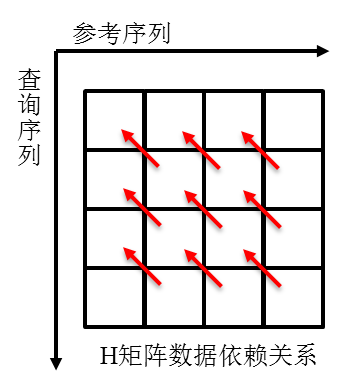
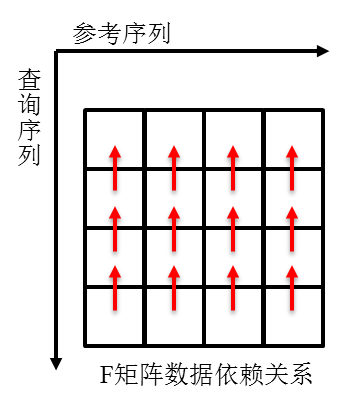
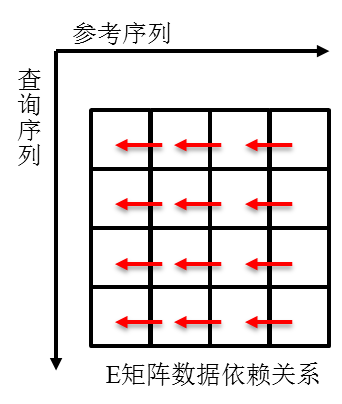


图 4.15 E、F、H矩阵数据依赖关系

从图中我们可以看出，矩阵中每个元素与其前一列元素有数据依赖，矩阵与其上一行元素有数据依赖，矩阵与其左上角元素有数据依赖。

在向量化时，我们需要充分考虑数据依赖关系对向量化程序效率的影响。以得分矩阵的计算为例，图4.16左图是一种最为直观的任务划分方式，将内层循环分成若干组，组的大小为向量处理器一次能够处理的元素个数，将每一组的处理结果保存在起来，供给下一列计算时使用。这种方法在能够完成向量化任务，但是并不是一个性能较好的方法。当计算矩阵时，由于每一列元素依赖的是位于其左上角的元素，因此，在内层循环的每一个向量运算之前，需要将保存上一列结果的向量存储器的值全部“平移”一个位置，并且还需要将上一列的下一个寄存器的第一个结果读取到当前位置寄存器的最后一个位置。这一系列操作需要付出较大的代价，会严重影响性能。

为了解决这一性能瓶颈，我们参考[48]的方法来调整我们的算法。同样以矩阵的计算为例，我们将矩阵的存储顺序调整为图4.16右图的“带状”存储方式。将长度为的查询序列分成段长度为的等长的片段，为向量处理器一次能够处理的数据个数，即

如果序列不能被整除需要在序列末端添加一些空元素占位。向量处理器中的每个处理单元负责一个片段，即向量处理器的第一个位置对应，第二个位置对应，.……，第16个位置对应。这样，我们再来分析相邻两列的数据依赖关系，可以发现，当前以一个向量寄存器为单位整体依赖于前一列的上一个寄存器，我们只需要把上一列中对应的向量寄存器结果复制到当前寄存器中，而不再需要在内层循环中再进行平移操作。只有第一个寄存器依赖于前一列的最后一个寄存器的值，并需要平移一个数据单元。这个平移操作只需要在循环开始前执行一次，因此，相比于前文中才取得划分方法，这种“带状”划分能够获得很大的效率提升。

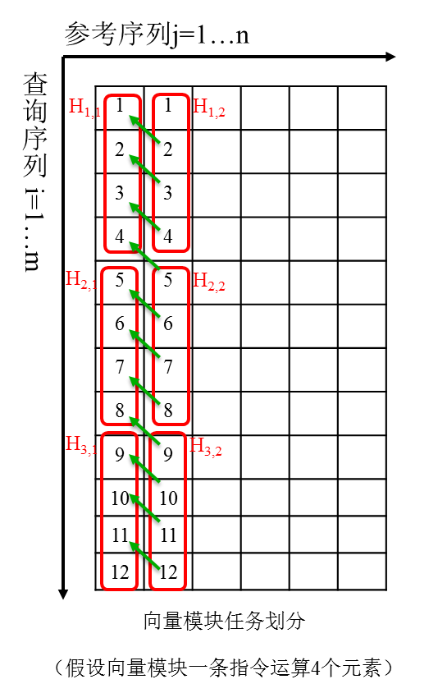
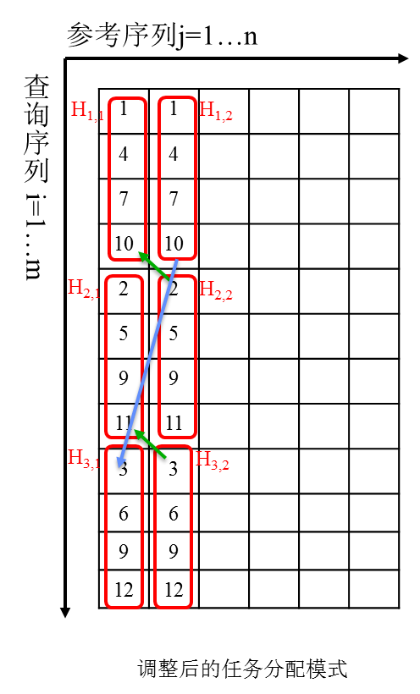
 

图 4.16 向量任务划分方式

注意到，在矩阵的递推函数中，还存在一个变量，出于程序效率考虑，为了避免频繁查找的值，我们将矩阵在主循环开始前，先计算出来一个矩阵，这个矩阵的元素与矩阵一一对应，并且只需要在主循环开始前计算一次，这样可以避免的频繁查找导致的程序性能瓶颈。同样的，因为我们调整了矩阵元素的排列顺序，对应的矩阵的元素顺序也需要做相应调整并于矩阵保持一致。计算的伪代码如图4.17所示。

foreach a in “ACGT“

h = 0;

for i = 0 … S

j = i;

for k = 1 … 16

if (j > L)

qp[a][h] = 0;

else

qp[a][h] = W(a, q[j]);

endif

h ++;

j += S;

endfor

endfor

endfor

图 4.17 矩阵计算伪代码

采用上述优化方法，我们使用一个典型的250bp长度的single-end比对数据进行测试。对比优化后程序运行时间，可以发现对这一数据集，向量化可以使*ksw\_extend2*这个函数运行时间缩短37%以上，如图4.18所示。

图 4.18 ksw\_extend2向量化加速效果

根据一些对Intel Xeon Phi协处理器的性能测试报告，对于计算密集度较高的程序，经过充分优化的利用向量化部件进行运算的代码能够比非向量化代码性能提升1-2倍。但是我们在对BWA中的Smith-Waterman算法做向量化优化时，受到了几个方面因素的限制，影响了向量化性能的进一步提升。这几个方面的因素包括：

1. single-end模式下，*ksw\_extend2*中采用了*banded-extend*方法，只需要计算得分矩阵中一部分元素的值。而目前采用的向量化方法依旧需要计算整个矩阵的值，计算量增加很多。
2. pair-end模式下，原代码中Smith-Waterman已经使用SSE2指令进行了优化，可以用8个16bit为单位进行并行计算。而AVX-512指令最小粒度为32bit，即同时进行16个数据并行计算，运算能力只提升了一倍。如果使用支持AVX-512BW指令的设备，可以额外获得一倍加速
3. 向量指令执行代价要比标量指令更大
4. AVX-512指令集缺少一些必要的指令，要实现相应的功能需要2-3条向量指令共同完成，如寄存器整体左移指令，指令条数更多

上述几个因素制约了Smith-Waterman部分向量化的性能提升，如果能够采用更合适的算法、支持AVX-512BW指令集的MIC设备，Smith-Waterman部分向量化的性能提升会更为明显。

## 4.6 本章小结

本章详细介绍了在Intel MIC平台上加速BWA序列比对算法的过程。基于Intel MIC同样也是采用x86架构这一重要特点，我们将BWA原始程序以及其依赖的静态和动态链接库移植到Intel MIC平台，在解决了较大数据传输的问题后，我们的基础版本性能与4核至强CPU的性能相当。接下来通过对移植的基础版本进行深入分析，我们发现在BWA算法中读取reads数据和写回比对结果的时间占比非常可观，因此我们对引入了三级流水线，将reads读取、序列比对计算、比对结果写回的过程重叠，很大程度上隐藏了数据读取和写回的时间开销。我们通过对多个不同特征的数据集进行测试，得到了最合适的线程数与分块大小。利用编译指导语句指导编译器的向量化，我们以相对简单的方式解决了编译器无法自动完成的向量化。为了能让向量化部分代码获得更好的性能，我们对频繁读取的关键数据结构进行了数据对齐。为了能进一步提升Intel MIC平台的性能，我们又花费大量的精力对无法被自动向量化的Smith-Waterman算法进行了一些算法上的优化，使用Intel的向量化函数实现了手动向量化，进一步提升了程序性能。最终，我们基于Intel MIC平台开发的BWA-MEM算法性能与10-12核至强CPU的性能相当，相比于基础版本性能提升了2-3倍。

# 第五章 性能评测

在前文中，我们分别尝试用GPU平台和MIC平台对BWA-MEM算法进行加速，在测试中我们发现MIC平台对BWA-MEM算法具有更好的加速效果。通过深入BWA-MEM算法与BWA-backtrack，BWA-SW的不同特征，并与已有的GPU加速序列比对算法进行对比，我们总结了BWA-MEM算法在GPU平台加速时所受到的限制和MIC平台更适合BWA-MEM算法加速的原因。接下来，我们将进一步全面测试MIC平台的BWA-MEM算法在不同比对模式下的性能，并与CPU平台的BWA、GPU平台的BarraCUDA进行性能对比。

## 5.1 实验配置

本文的实验平台采用的是曙光星云高性能服务器和INTEL S2600GZ服务器。曙光服务器搭配2个INTEL Xeon E5-2620六核CPU和1个Tesla K40 GPU；INTEL服务器搭配2个INTEL Xeon E5-2670八核CPU和1个INTEL Xeon Phi 5110P 协处理器。各硬件设备的主要参数如表5.1所示。

表 5.1 实验平台处理器主要参数

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *设备* | | Intel  Xeon E5-2620 | Intel  Xeon E5-2670 | NVIDIA  K40 | Intel Xeon Phi 5110P |
| *架构* | | Sandybridge | Sandybridge | Kepler | Knights Corner |
| *时钟频率(GHz)* | | 2.1 | 2.6 | 0.75 | 1.053 |
| *电源功耗(Watt)* | | 95 | 115 | 235 | 300 |
| *核心数* | | 6 | 8 | 2880 | 60 |
| *数据宽度* | | 64bit | 64bit | 384bit | 512bit |
| *FP (GFLOP/s)* | | 141.66 | 168 | 4290 | 2022 |
| *DP (GFLOP/s)* | | 141.66 | 168 | 1430 | 1011 |
| *DRAM* | *type* | DDR3 1.3GHz | DDR3 1.3GHz | GDDR5 3GHz | GDDR5 5GHz |
| *size* | 48GB | 64GB | 12GB | 8GB |
| *bandwidth* | 32GB/s | 32GB/s | 288GB/s | 320GB/s |
| *PCIe2.0* | | X16，8GB/s | | | |

其中NVIDIA K40 GPU总共有2880个CUDA核心，拥有15个流多处理器单元，每个单元由192个0.75GHz的流处理器组成。其单精度浮点峰值可达4290 GFlops。每个流多处理器拥有48KB的片上共享内存单元和65536个寄存器文件，片外的全局显存容量为12GB，常量内存为64KB。在同一个流处理单元上每个kernel最多可启动1024个线程。和以前的G80和K20相比，K40增加了CUDA核心数量和显存容量

Xeon Phi 5110P 协处理器拥有60个频率为1.053GHz的x86计算核心，每个核心拥有一个独立的512bit位宽的向量处理器。其单精度浮点峰值达到了2.022TFlops。拥有全局可见的512KB L2Cache，通过片上环形互联总线连接。MIC的核心支持硬件线程技术，每个核心可以启动4个硬件线程，具有60个核心的Xeon Phi 5110P 协处理器最多可以启动240个硬件线程。

数据方面，本文使用的参考序列数据为人类基因组8号染色体数据，不同长度的reads数据是使用WgSim生成的。

## 5.2 BWA-MEM加速效果测试

测试BWA-MEM在MIC平台的性能，我们分别选取不同长度的reads测试他们在single-end和pair-end模式下的性能。

### 5.2.1 Single-end模式

Single-end数据测试结果如表5.2所示。

表 5.2 Single-end模式测试结果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| reads长度 | reads数量 | BWA-MEM | | BarraCUDA | BWA-MEM-MIC |
| 1 CPU | 8 CPU |
| 100bp | 1000000 | 275.606 | 35.412 | 42.988 | 33.328 |
| 250bp | 1000000 | 675.182 | 85.764 | 结果错误 | 72.812 |
| 500bp | 500000 | 786.415 | 101.956 | 结果错误 | 90.389 |

其中，BarraCUDA对于100bp以上的reads比对结果与正确结果相距甚远，查阅BarraCUDA代码可以发现在该GPU加速版本在代码中不支持100bp以上reads，会产生异常。根据图中数据我们可以看到，在Single-end模式中BWA-MEM-MIC比对速度超越了8核CPU和BarraCUDA。

MIC平台BWA-MEM产生的输出结果，与BWA-MEM在CPU平台的输出结果完全一致。

### 5.2.2 Pair-end模式

Pair-end数据测试结果如表5.3所示。

表 5.3 Pair-end模式测试结果

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| reads长度 | reads数量 | BWA-MEM | | BarraCUDA | BWA-MEM-MIC |
| 1 CPU | 8 CPU |
| 100bp | 1000000 | 560.421 | 67.221 | 86.732 | 79.935 |
| 250bp | 1000000 | 1282.465 | 168.944 | 结果错误 | 135.440 |
| 500bp | 500000 | 1498.241 | 190.788 | 结果错误 | 179.391 |

同样，在pair-end模式BarraCUDA同样不支持100bp以上序列比对。根据图中数据我们可以看到，在pair-end模式中BWA-MEM-MIC比对速度同样超越了8核CPU和BarraCUDA。

在pair-end模式下，BWA多线程模式下的输出结果与一次处理的reads分块大小有关。[42]由于MIC平台与CPU平台的分块大小差别较大，因此二者输出结果存在一定的不同。但是如果指定两个平台上采用同样的分块大小，则在测试中可以取得同样的比对结果，可以证明我们的算法是正确的。

## 5.3 BWA-MEM加速算法微架构分析

接下来，我们进一步对MIC平台的BWA-MEM算法进行更为细致的测试，从MIC设备微架构角度分析当前的性能瓶颈。

为了尽可能减小程序性能监测计数导致的误差，我们选择更大规模的数据进行性能测试。在这里我们采用500万条250bp的single-end数据和pair-end数据来进行测试。性能测试工具使用Intel公司的Intel® VTune™ Amplifier 2016版本，相对于历史版本，该版本的VTune能够提供更全面的性能指标。我们利用VTune测试在single-end和pair-end模式下各自的访问频率最高的几个热点模块，包括seed查找相关的bwt\_occ，bwt\_occ4，bwt\_2occ4，seed拓展相关的基于Smith-Waterman算法的ksw\_extend2，ksw\_global2，ksw\_i16，以及后缀数组查找的bwt\_sa模块，进行了深入的热点分析。由于MIC设备上的性能计数器不如CPU设备丰富，我们利用能够获取到的性能参数，尽可能的揭示从微体系机构角度来看存在的性能瓶颈。

### 5.3.1 CPI指标

CPI是用来衡量程序性能的一个重要指标，它表示平均执行每条指令所需要的时钟周期数，在VTune中统计的是所有硬件线程的平均值。理论上最佳的CPI值为2，在程序热点模块中如果CPI值大于4时往往表明程序中存在较大的数据延迟，需要重点优化。利用VTune我们可以测量出整个BWA-MEM算法的CPI为2.384。进一步的，我们采用Yasin[49]提出的Top-down性能分析方法对指令执行测数最多的几个热点模块进行进一步分析，结果如图5.1所示。

图 5.1 CPI测试结果

根据图5.1的测试结果我们可以观察到：

（1）ksw\_extend2、ksw\_global2、ksw\_i16这三个模块的CPI均小于2，这三个函数都是基于Smith-Waterman算法对seed进行拓展，前两个应用于single-end模式，最后一个应用于pair-end模式。如4.6节所述，我们对这三个模块都进行了手动向量化优化，可以看出这几个模块的CPI最小，加速效果最好；

（2）bwt\_occ、bwt\_occ4、bwt\_2occ4这几个模块的CPI介于2~3之间，这几个模块主要是基于FMD-index的seed查找模块，如前文中所述，我们采用的是编译指导语句指导编译器完成向量化，这几个模块的CPI略高于（1）中手动向量化的几个模块，考虑到查找过程中访存局部性要远差于Smith-Waterman算法，因此这几个模块的CPI也在可以接受的范围。

（3）bwt\_sa模块的CPI较高，达到了3.523，该模块主要完成对后缀数组的查找，其访问频率不如其他几个模块，我们目前未对其进行深入优化。可以看出未优化的模块CPI要远高于经过优化的其他模块。在后续工作中我们可以对该模块进行进一步的优化。

### 5.3.2 Cache利用率

在现代计算机体系结构中，充分利用cache可以大幅提高程序性能，同样，如果cache命中率较低，则会严重影响程序性能。我们使用的Knights Corner架构的MIC设备，其cache信息如表5.4所示

表 5.4 Intel® Xeon Phi™ 协处理器cache信息

|  |  |  |
| --- | --- | --- |
| Cache类型 | 大小 | 特征 |
| L1 Instruction | 32KB | 8-way, 64B line size |
| L1 Data | 32KB | 8-way, 64B line size |
| L2 Instruction+Data | 512KB | 8-way, 64B line size |

Intel® Xeon Phi™ 协处理器的计算核心由环形总线连接，每个核心上最多可以运行4个线程，其线程切换方式为round-robin。每个计算核心包括一个有序的双发射流水线，一个本地L1和L2缓存，一个向量运算单元。此外协处理器还有8个双通道的GDDR5存储控制器，每个通道可以提供5.5 GT/s的速度，理论最高带宽为352GB/s

通过VTune，我们可以测得程序的cache命中率、cache miss导致的延迟大小等性能指标，并采取相应的优化方法进行优化。在VTune中，L1 cache访问可能存在三种结果：命中、未命中和预取命中。所谓预取命中是指访问的数据没有在cache中，但是已经通过数据预取技术取出，其访问延迟要比cache命中长一些，但是远低于未命中的情况。VTune统计的L1 cache命中率包括命中和预取命中，其计算公式为：

通过VTune我们还可以统计L1 cache未命中导致的访问延迟时钟周期个数，该指标可以用来评估L1 cache的未命中是否能够由L2 cache来提供缺失数据，如果该指标小于145，说明超过半数肚饿L1 miss能够在L2 cache命中。该指标的计算公式为：

通过VTune，我们测得整个程序的L1 cache命中率为99.3%，进一步的，我们对热点模块进行更详细的测试，结果如表5.5所示。

表 5.5 Cache命中率及未命中延迟测试结果

|  |  |  |
| --- | --- | --- |
| 模块 | L1 cache命中率 | 未命中延迟 |
| bwt\_occ | 99.90% | 724.912 |
| bwt\_occ4 | 99.00% | 662.948 |
| bwt\_2ooc4 | 99.80% | 1508.017 |
| bwt\_sa | 98.20% | 534.154 |
| ksw\_extend2 | 99.90% | 6054.653 |
| ksw\_global2 | 99.90% | 5701.481 |
| ksw\_i16 | 99.90% | 10197.032 |

通过表5.5的测试结果我们可以观察到：

1. L1 cache命中率普遍比较高，几个热点函数均能达到98%以上的L1 cache命中率，对于ksw\_extend2，ksw\_global2，ksw\_i16这几个经过手动向量化的模块，其命中率达到了99.9%。bwt\_sa模块由于访存局部性较差，其命中率最低，为98.2%。
2. 未命中延迟普遍较大，其中手工向量化的几个模块的未命中延迟达到了5000以上，我们分析这是由于向量化部件对访存局部性要求更高，并且要求对齐访问，而在这些模块中由于算法原因存在一部分非对齐访问，造成延迟很大。另外，我们在优化过程中没有有针对性的使用数据预取技术，在后续工作中可以继续改进。不过由于这几个模块cache命中率都比较高，由此带来的性能损失影响不大。

### 5.3.3 向量化效果

充分利用向量部件对程序性能的提升有重要帮助。icc编译器会自动对能够进行向量化的部分进行向量化。对于编译器无法完成向量化的部分，化可以通过编译指导指令指导编译器完成向量化，也可以使用向量化函数手动向量化。在BWA-MEM算法中，除自动向量化的部分外，手动向量化的重点主要包括两部分，一是在bwt查找时的向量化，通过编译指导语句实现，包括bwt\_occ，bwt\_occ4，bwt\_2occ4这几个函数；二是基于Smith-Waterman算法的seed拓展模块——ksw\_extend2, ksw\_global2, ksw\_i16这几个函数的向量化。下面我们分别测试并分析这几个模块的向量化效果。

VTune可以对于向量化性能的检测可以提供三个主要性能指标。其中，向量化程度是被向量指令处理的数据元素个数与向量指令数的比值。在MIC平台的BWA-MEM算法中，MIC平台向量运算宽度为512位，以32位为一个单元进行运算，因此最佳向量化程度为16。向量化程度计算公式为

第二个衡量向量化性能的指标是计算与数据访问比值，这个比值用来衡量程序的计算密度，即每次数据加载后可以进行的计算次数。适合在MIC设备上运行的程序应能充分向量化，加载一次数据进行尽可能多的运算。通过VTune，我们可以分别测得L1和L2 cache的计算与数据访问比。对于L1 cache，其含义为平均每次L1 cache访问的向量化操作次数，计算公式如为

对于L2 cache，其定义为平均每次L2 cache访问能够进行的向量化操作的次数。其计算公式为

对于L1 cache，计算密度对比值的影响最为关键，而对于L2 cache，则更多是说明代码是否在高效执行。

首先我们测试了热点模块的向量化程度，对于整个程序，VTune测得的整体向量化程度为10.922，对各个热点模块的测试结果如图5.2所示。

图 5.2 热点模块向量化程度测试

通过图5.2的测试结果我们可以发现，通过编译指导语句指导编译器进行向量化的seed查找相关函数向量化程度较低，为此我们对这几个模块的代码进行了分析，发现这几个模块向量化程度较低的原因是运算量不足，其核心的gather操作对象普遍不超过7个，这是由于seed查找算法的特性决定的，因此虽然其向量化程度较低，但对于当前的seed查找算法均为合理的结果，但是目前的算法也浪费了向量部件一半的运算能力。

对于seed拓展算法，经过我们的手动向量化之后，三个模块的向量化程度都比较高，达到了14~16，我们的运算都是按照32位为一个单元，一次只能完成16个数据的运算，但是，实际上只需要16位就可以满足计算精度，但是在AVX-512指令集中，算术运算的最小粒度为32位，不支持更小粒度的计算。如果采用AVX-512BW指令级，则可以支持以16位为单位进行运算，向量化程度有望接近32。

对于计算与数据访问比值这一指标，我们测试了L1和L2两级cache，对于L1 cache，程序整体指标为10.803，对于L2 cache，程序整体性能指标为1629.602。对各个热点模块进行更详细的分析，结果分别如图5.3和图5.4所示。从图中可以看出，基于Smith-Waterman的seed拓展模块性能指标要远远好于seed生成模块，这是因为基于Smith-Waterman的动态规划算法计算密度更高，而seed查找模块则相对计算密集度不足。对这一性能指标进行优化，需要进一步优化seed查找算法。

图 5.3 L1 COMPUTE / DATA ACCESS测试结果

图 5.4 L2 COMPUTE / DATA ACCESS测试结果

## 5.4 本章小结

本章采用不同长度的reads对基于MIC平台的BWA进行性能测试，测试覆盖了single-end和pair-end多种模式。从测试结果可以看出，MIC平台的BWA在各个分支中性能都超越了8核CPU和BarraCUDA。接下来，我们利用VTune对程序的主要性能指标进行了测试，并分析了各项性能指标的瓶颈及改进方法。

# 第六章 结束语

## 6.1 本文工作总结

本文研究利用众核处理器加速BWA序列比对的方法，分别利用GPU和MIC平台对BWA进行加速，深入分析了最新的BWA-MEM算法的整体流程和热点模块，设计并实现了BWA-MEM算法的GPU移植，结合GPU架构与硬件特性对算法进行优化，设计并实现了BWA在Intel MIC平台的移植工作，对比了不同平台对BWA序列比对算法的加速效果。

在GPU加速BWA算法的研究中，我们结合GPU平台的特性调整了算法流程，为BWA-MEM算法的热点模块，SMEM查找模块和chain生成模块编写kernel函数并放在GPU设备上执行，对数据存储方式进行了优化，尝试了多种任务划分方式。通过对GPU加速BWA-MEM算法、BarraCUDA加速BWA-ALN算法、nvBowtie加速Bowtie2算法进行深入比较，我们总结了BWA-MEM由于其算法特性在使用GPU平台进行加速时会收到很大的限制。

我们用Intel MIC平台加速了BWA算法。基于Intel MIC平台同样采用x86架构这一重要特点，我们将BWA原始程序以及其依赖的静态和动态链接库移植到Intel MIC平台，形成了一个基础的移植版本。我们深入分析了基础版本中存在的性能瓶颈，采用了三级流水线、数据对齐、指导向量化等优化手段对基础版本进行优化。我们还对BWA-MEM中采用的基于Smith-Waterman的拓展算法进行了手工向量化，取得了更好的加速效果。最终我们Intel MIC平台的BWA的性能与10~12核Intel Xeon CPU性能相当，相对于基础版本性能提高了2~3倍。

## 6.2 下一步工作

本文基于众核处理器对BWA序列比对加速算法进行了研究，将BWA算法移植到GPU平台和MIC平台，并进行了大量测试，取得了不错的效果。不过本文的工作只是对BWA序列比对算法在众核平台上的加速效果进行了初步的研究与实现，通过对加速效果进行深入分析可知，还有很多工作可以继续。

首先，对于GPU平台，我们对最新的BWA-MEM算法中的seed查找和chain生成模块利用GPU进行了加速，但是由于chain生成模块受访存限制，该模块加速效果并不理想。如果能对seed和chain的存储方式进行深入优化，降低chain生成模块的访存要求，该模块在GPU平台的加速应该能够取得更好的效果。

其次，对于MIC平台，我们采用的是Native模式运行，全部运算都在MIC设备上进行，CPU则一直处于闲置状态。如果能够同时利用CPU与MIC设备来进行比对，比对速度将可以得到进一步提高。

最后，本文对BWA的加速始终保证产生的比对结果与原BWA程序的比对结果一致。实际上，目前多数GPU、MIC、FPGA平台的序列比对工具的加速，如nvBowtie、BarraCUDA等，为了获得更好的性能，都对原程序的算法流程进行了较大的修改，增减了一些限制条件，使得产生的比对结果与原程序存在一定的出入。因此，对于最新的BWA-MEM算法的加速研究，接下来也可以尝试对原算法进行更多的修改，在准确率满足一定要求的条件下，进一步获得更好的加速效果。

参考文献

1. 孙啸，陆祖宏，谢建明. 生物信息学基础. 清华大学出版社. 2005. p87.
2. Sanger, F., et al. 1977. DNA sequencing with chain-terminating inhibitors. Proc. Natl. Acad. Sci. USA 74: 5463-5467
3. Moore’s Law: Raising the Bar, 2005.
4. NVIDIA CUDA C Programming Guide. <http://NVIDIA.com/cuda>
5. Intel Xeon Phi Coprocessor Architecture
6. Yue G D, Gao Q, Luo L H, et al. The application of High-throughput sequencing technology in plant and animal research. SCIENTIA SINICA Vitae, 2012, 42: 107-124, doi: 10.1360/052011-634
7. MARTIN J A, WANG Z. Next-generation transcriptome assembly[J]. Nature Reviews Genetics, 2011, 12(10): 671-682.
8. LI H, HOMER N. A survey of sequence alignment algorithms for next-generation sequencing[J]. Briefings in Bioinformatics, 2010, 2(5): 473-483.
9. LI R, LI Y, KRISTIANSEN k, *et al*. SOAP: short oligonucleotide alignment program[J]. Bioinformatics, 2008, 24(5): 713-714.
10. JIANG H, WONG W H. SeqMap: mapping massive amount of oligonucleotides to the genome[J]. Bioinformatics, 2008, 24(20): 2395-2396.
11. LI H, RUAN J, DURBIN R. Mapping short DNA sequencing reads and calling variants using mapping quality scores[J]. Genome Research, 2008, 18(11): 1851-1858.
12. RUMBLE S M, LACROUTE P, DALCA A V, et al. SHRiMP: Accurate mapping of short color-space reads[J]. PloS Computational Biology, 2009, 5(5): e1000386.
13. SMITH A D, XUAN Z, ZHANG M Q, ET AL. Using quality scores and longer reads improves accuracy of Solexa read mapping[J]. BMC Bioinformatics, 2008, 9(1): 128.
14. DELCHER A L, KASIF S, FLEISCHMANN R D, ET AL. Alignment of whole genomes[J]. Nucleic Acids Research , 1999, 27(11): 2369-2376.
15. DELCHER A L, PHILLIPPY A, CARLTON J, et al. Fast algorithms for large-scale genome alignment and comparison[J]. Nucleic Acids Research, 2002, 30(11): 2478-2483.
16. LI H, DURBIN R. Fast and accurate long-read alignment with Burrows-Wheeler transform[J]. Bioinformatics, 2010, 26(5): 589-595.
17. LI H, Durbin, R. Fast and accurate short read alignment with Burrows-Wheeler transform[J]. Bioinformatics, 2009, 25:1754-1760.
18. LANGMEAD B, TRAPNELL C, POP M, et al. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome[J]. Genome Biology, 2009, 10(3): R25.
19. LAMTW, SUNG W K, TAM S L, et al. Compressed indexing and local alignment of DNA[J]. Bioinformatics, 2008, 24(6): 791-797.
20. ABOUELHODA M I, KURTZ S, OHLEBUSCH E. Replacing suffix trees with enhanced suffix arrays[J]. Journal of Discrete Algorithms, 2004, 2(1):53-86.
21. FERRAGINA P, MANZINI G, MAKINEN V, et al. An Alphabet-Friendly FM-index[C] // APOSTOLICO A, MELUCCI M. 11th International Conference. Padova: SPIRE, 2004, 150-160.
22. Burrows M, Wheeler D J. A block-sorting lossless data compression algorithm. 2004, Technical report 124, Palo Alto, CA, Digital Equipment Corporation.
23. Open MP [EB/OL]. http://www.openmp.org, Accessed 2013-04-12.
24. Malhis N, Butterfield YSN, Ester M, et al. Slider–maximum use of probability information for alignment of short sequence reads and SNP detection[J]. Bioinformatics, 2009, 25:6-13.
25. NCBI-BLAST[CP/OL]. http://blast.ncbi.nlm.nih.gov/Blast.cgi, Accessed 2013-04-10.
26. Nguyen VH, Lavenier D. PLAST: parallel local alignment search tool for database comparison [J]. BMC Bioinformatics, 2009, 10:329.
27. Darling AE, Carey L, Feng W. The design, implementation, and evaluation of mpiblast[C]. In: proceedings of 4th International Conference on Linux Clusters: The HPC Revolution 2003 in conjunction with Cluster World Conference & Expo, San Jose, CA, 2003.
28. SALSA [EB/OL]. <http://salsahpc.indiana.edu/>
29. Hadoop-Blast [CP/OL]. <http://salsahpc.indiana.edu/tutorial/hadoopblast.html>
30. Open ACC [EB/OL]. <http://www.openacc-standard.org/>
31. Schatz MC, Trapnell C, Delcher AL, et al. High-throughput sequence alignment using Graphics Processing Units [J]. BMC Bioinformatics, 2007, 8:474.
32. Manavski SA, Valle G. CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment[J]. BMC Bioinformatics, 2008, 9(Suppl 2):S10.
33. Striemer GM, Akoglu A. Sequence Alignment with GPU: Performance and Design Challenges[C]. In: IEEE International Symposium on Parallel and Distributed Processing, 2009.
34. Liu Y, Maskell DL, Schmidt B. CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units[J]. BMC Res Notes, 2009, 2:73.
35. Vouzis PD, Sahinidis NV. GPU-BLAST: using graphics processors to accelerate protein sequence alignment[J]. Bioinformatics, 2011, 27 (2):182-188.
36. Liu W, Schmidt B, Muller-Wittig W. CUDA-BLASTP: accelerating BLASTP on CUDA-enabled graphics hardware[J]. IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB), 2011, 8(6):1678-1684.
37. Needleman SB, Wunsch CD. A general method applicable to the search for similarities in the amino acid sequence of two proteins [J]. Journal of Molecular Biology, 1970, 48(3):443-453.
38. Liu CM, Wong T, Wu E, et al. SOAP3: Ultra-fast GPU-based parallel alignment tool for short reads [J]. Bioinformatics, 2012, bts061v1-bts061.
39. Li R, Yu C, Li Y, et al. SOAP2: an improved ultrafast tool for short read alignment [J]. Bioinformatics, 2009, 25:1966-1967.
40. Klus P, Lam S, Lyberg D, et al. Barra CUDA - a fast short read sequence aligner using graphics processing units [J]. BMC Research Notes, 2012, 5(1):27.
41. Liu Y, Schmidt B, Maskell DL. CUSHAW: a CUDA compatible short read aligner to large genomes based on the Burrows-Wheeler transform[J]. Bioinformatics, 2012, 28(14):1830-1837.
42. Li H. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM[J]. arXiv preprint arXiv:1303.3997, 2013.
43. Yongchao Liu and Bertil Schmidt: "Long read alignment based on maximal exact match seeds". Bioinformatics, 2012, 28(18): i318-324
44. Yongchao Liu and Bertil Schmidt: "CUSHAW2-GPU: empowering faster gapped short-read alignment using GPU computing". IEEE Design & Test of Computers, 2014, 31(1): 31-39
45. Yongchao Liu, Bernt Popp, and Bertil Schmidt: "CUSHAW3: sensitive and accurate base-space and color-space short-read alignment with hybrid seeding." PLOS ONE, 2014, 9(1):e86869
46. LI H. Exploring single-sample SNP and INDEL calling with whole-genome de novo assembly[J]. Bioinformatics, 2012, 28:1838-1844.
47. Houtgast E J, Sima V M, Bertels K, et al. GPU-Accelerated BWA-MEM Genomic Mapping Algorithm Using Adaptive Load Balancing[M]//Architecture of Computing Systems--ARCS 2016. Springer International Publishing, 2016: 130-142.
48. Farrar M. Striped Smith–Waterman speeds database searches six times over other SIMD implementations[J]. Bioinformatics, 2007, 23(2): 156-161.
49. Yasin A. A top-down method for performance analysis and counters architecture[C]//Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on. IEEE, 2014: 35-44.

致谢

在我的硕士课题和论文即将完成之际，谨在此向攻读硕士学位期间关心和指导我的老师，以及一直以来支持和帮助我的家人、朋友和同学致以崇高的敬意和衷心的感谢！

首先，感谢我的导师孙凝晖研究员和谭光明研究员！孙老师肩负所长重任，平日工作十分繁忙，却从未因此忽视对学生的指导与关心。每周的讨论班上我们不仅收获了知识，更开阔了眼界。我更要感谢谭老师，在我整个研究生期间对我的研究工作的殷切指导。从选择课题到完成论文，谭老师始终都是耐心地指导，并给予不懈的支持。谭老师对科研工作要求严格、精益求精，同时又给予了我充分的自由空间，在我参加科技竞赛和求职期间给予了充分的理解，并指导我赶上工作进度，令我非常感激。孙老师和谭老师敏锐的洞察力、渊博的知识、严谨的治学态度、精益求精的敬业精神，无论我以后从事何种职业，都值得我学习！

感谢张春明老师、姚二林老师，有你们这样的良师益友，在我学习、工作、生活上给予关心、帮助，让我受益良多。感谢张佩珩老师，是您把我带到了高性能中心，感谢您在夏令营时对我的鼓励，让我在错失外校保研资格后依然能坚持考研来到这里。感谢霍老师，作为我们的辅导员，在生活上处处为我们着想，解决我们遇到的问题。感谢阳纯辉、何玉晓两位老师，是你们细致入微的工作让我们能够安心进行科研工作。

感谢张中海师兄、李强强师兄、隽鹏辉师兄，你们在科研工作和求职方面都给予我很多宝贵的建议，让我少走很多弯路。感谢李红印、王元戎、刘军红、郭浩强、邵慧子同学，能与你们一起度过宝贵的研究生时光，给了我很多难忘的回忆。感谢李雪琦、曾平、王炳琛师弟，你们的活力和认真的态度无时无刻不在感染着我。

最后，我想感谢我的父母和家人对于我这么多年来的支持，你们让我更加乐观地面对生活中的一切，你们对我如此无私的付出，我会用一生去回报。感谢我的朋友，是你们在我承受巨大压力时帮我舒缓，在我遇到困难时伸出援手。感谢我生命中的每一个人！

刘闯

2016年5月