浙江大学

本科生毕业论文

题目 采用 GPGPU 加速的基因对齐算法研究

姓　　名 _____

学　　号 _____

指导教师 _____

专　　业 计算机科学与技术

学　　院 计算机科学与技术学院

A Dissertation Submitted to Zhejiang University for the Degree of Bachelor of Engineering

TITLE: On Algorithm for Gene Alignment with GPGPU

Author: _____

Supervisor: _____

Major: Computer Science and Tecnology

College: Zhejiang University

Submitted Date: _____

浙江大学本科生毕业论文（设计）诚信承诺书

1.本人郑重地承诺所呈交的毕业论文（设计），是在指导教师的指导下严格按照学校和学院有关规定完成的。

2.本人在毕业论文（设计）中引用他人的观点和参考资料均加以注释和说明。

3. 本人承诺在毕业论文（设计）选题和研究内容过程中没有抄袭他人研究成果和伪造相关数据等行为。

4. 在毕业论文（设计）中对侵犯任何方面知识产权的行为，由本人承担相应的法律责任。

毕业论文（设计）作者签名：

_____年_____月_____日

摘要

最新的NGS测序技术可以在一次运行中产生5000万到两亿个可以匹配的短基因序列，要将如此大规模的输入序列与人类基因组进行匹配需要更加快速的基因对齐算法，本文计划通过研究来验证通过 GPGPU 来加速原有基因对齐算法的可行性，我选择了普及度较高的BWA算法作为改进算法，利用CUDA平台进行并行运算程序的开发，编写了 BWA 算法核心部分的 GPU 版本，并且又重现了其算法的 CPU 版本，通过对这两个版本的 BWA 算法在处理不同长度的读入时的耗时，我发现在处理 32bp 长度的读入，读入的查询序列个数超过 1000 个时，或者在处理100bp长度的读入，读入的查询序列个数超过250个时，GPU的运行效率要超过CPU。同时，随着读入个数的增长，GPU版本的耗时的增长速率要远慢于 CPU 版本，因此本文认为，通过 GPGPU 来加速基因对齐算法这一思路是可行的。

关键词 基因对齐算法 BWA GPGPU 并行运算

Abstract

The NGS sequencing technology can generate a huge amount of query sequences at each run. Thus, aligning those short gene sequences to the reference genome sequence becomes a big challenge. To overcome this problem, we need faster read alignment program. In this article, I expect to use GPGPU parallel computation to enhance existing algorithm. I chose the widely-used BWA algorithm as the target. And design the GPU version of the BWA algorithm using CUDA, I also reproduce the CPU version of BWA for testing purpose. In the test I found that when using 32bp read, the GPU version is faster when dealing with more than 1000 short reads, when using 100bp read, the CPU version is faster when dealing with more than 250 reads, and when we added more reads to the test, the time consumed by the GPU version BWA grown much slower, therefore we can draw the conclusion that using GPGPU to accelerate BWA is promising.

Keywords **gene sequence alignment** **BWA GPGPU parallel computation**

目录

第1章 绪论

1.1 引言

自孟德尔发现遗传定律以来，人们从来没有停止过对生物遗传物质的探索，20 世纪早期，格里菲斯通过了肺炎双球菌转化实验，证明了 DNA 是生物体内的主要遗传物质，1951 年，生物学家詹姆斯沃森与物理学家弗朗西斯克里克，通过 DNA 的衍射照片，发现了 DNA 的双螺旋结构，进而推导出了 DNA 的复制，转录以及蛋白质的生成等一系列生理过程的发生机制，这标志着人们对于遗传物质的研究进入了分子时代。1990 年，人类基因组计划启动，该计划致力于测量人类的整个基因组序列，绘制出人类基因的图谱，1999 年，我国加入到这个浩大的工程之中，2000 年，人类基因组的草图初步绘制完成，2005 年，完整人类基因组的测序工作完成，该计划耗时 15 年，总斥资近 30 万美元，终于实现了人类对于基因世界全貌的一次伟大窥探。

但是人们对于基因的探索不止于此，众所周知，个体的基因差异会影响到诸多疾病的表型，因此，在临床医疗领域上，针对单个患者基因组的测量，对于其所患疾病的诊断，治疗以及预后都具有极其重要的意义，但是，想要对单个个体进行测序，并不是一件简单的事，如果使用与人类基因组计划中所用的技术，那么所带来的时间以及金钱上的代价，对于个体患者来说无疑是难以接受的，幸运的是，随着科技的发展，更加快速，更加省钱的基因测序技术被发明出来，它被成为 NGS（New Generation Sequencing）,中文翻译成高通量测序或者次时代测序，这种利用新型原理的测序技术可以在一天之内以 1000 美元以内的成本完成对单个个体的基因测序，这使将基因测序技术投入到临床当中成为可能。

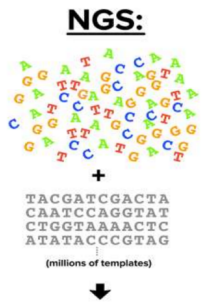


图 1-1

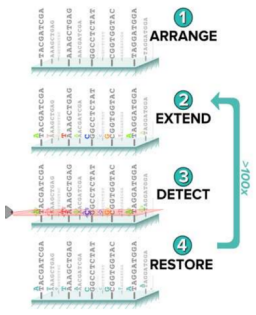


图 1-2

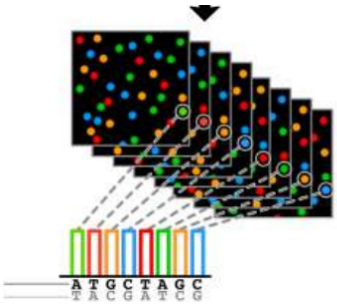


图 1-3

图 1-1 到 图 1-3 演示了 NGS 测序的基本原理

然而，使用新原理的基因测序技术却也为算法开发的工作人员带来了不小的挑战，因为使用 NGS 平台的测序仪一次运行可以产生数目众多的可匹配的短基因序列，而序列对齐（sequence aliment）又是整个基因测序体系中最为耗时的一个步骤，NGS 出现之前产生的序列对齐算法在面对如此大规模的输入数据时显得力不从心，因此，为了处理 NGS 所产生的数目巨大的输入序列，我们有必要开发一种可以快速准确匹配这些输入序列的算法。本文的目的，旨在验证能否使用并行运算的方法，改良当前已经出现的算法，使其可以更为迅速的匹配大量的输入。

1.2 课题背景

1.2.1 NGS

NGS（Next Generation Sequencing）是新一代的基因测序技术，其基本原理脱胎于 sanger 测序，在 NGS 的测序过程中，首先，双链 DNA 在解旋酶的作用下解旋成单链 DNA，在这之后，解旋完成后的长单链 DNA 被打碎成为长度在 100bp（bp，base pair，是基因序列的长度单位，表示一段序列拥有多少个碱基对）左右的片段，接下来，在 NGS 平台的测序仪内，这些基因片段的一端将会被固定在一个玻璃模板上，在这之后的测序过程中，每个基因片段在玻璃板上的位置是固定的。然后在 DNA 复制酶的作用下，经过特殊处理的荧光标记单核苷酸与长链 DNA 结合，由于荧光的单核苷酸无法与 DNA 复制酶结合，因此所有的单链 DNA 在第一个位点结合到荧光的单核苷酸之后停止复制，此时，显微照相机会记录下玻璃模板上每一个位置上的荧光

标记，由于不同种类的单核苷酸使用不同颜色的荧光进行标记，因此我们就可以通过荧光的颜色得知模板上黏附的每一段基因序列第一个位点的单核苷酸是什么，这之后，已经结合在基因片段上的单核苷酸上的荧光标记将被切除，这样一来，被标记的单核苷酸变成了普通的单核苷酸，DNA 链的复制可以继续，荧光标记的单核苷酸就可以继续附着在基因片段的第二个位点上，这个过程不断循环，就可以得到被打碎后的每个序列片段的测序结果。

为了将这些被打碎的基因片段恢复成完整的人类基因组序列，我们还需要知道每个片段在人类基因组之中的位置，各种基因对齐算法正是为了实现这个目的而设计的。我们通过基因对齐算法，将短的读入片段与完整的参照序列——人类基因组相匹配，通过匹配结果来定位其在基因组中的位置，就可以恢复得到我们想要的完整序列。现代的 NGS 系统可以在一次运行之中，产生 5000 万到 2 亿个可匹配的长度在 32bp-100bp 之间的读入短序列，要将高效的处理如此大规模的信息，先进的算法与硬件设施是必不可少的，在诸多生物信息学专家的努力之下，各种基于不同原理的基因对齐算法被研制了出来。

图 1-4 Hiseq 基因测序仪

1.2.2 基因对齐算法的演进以及现状

最开始的基因对齐算法包括 Blast 以及 Blat，这类序列对齐算法在 NGS 技术成熟之前就已经被投入使用，他们使用的算法类似于动态规划，主要被用来对齐单个基因序列，由于其效率较低，难以与 NGS 数目巨大的输入规模相适应，因此不能作为次时代基因测序的基因对齐算法来使用，但是尽管如此，这一类的算法仍然活跃在一些科研网站，为全世界的科研工作者进行单个序列的位置查询提供方便。

这之后为了适应 NGS 的大规模输入数据，一些新的算法逐渐被开发了出

来，这些算法可以大致分为两大类，第一类算法基于哈希表原理，这类算法又可以继续细分成两个子类，第一个子类对读入的短序列建立哈希表，这类算法包括 Eland，MAQ，ZOOM 等，这一类算法的缺点是由于没有对参照序列，建立索引，因此需要不断地对整个基因组进行扫描，每次扫描仅能匹配很小一部分的读入序列，最终消耗掉很多的时间。第二个子类对参照序列建立了哈希表，包括 SOAPv1,PASS 等等，这样做使得这一类算法可以轻松的实现多线程执行，但是同时也面临着一些比较严重的问题，首先，对整个人类基因组建立哈希表会占用很大的一块内存，并且，由于这一类算法中使用了循环的策略，因此会使得其对 NGS 测序结果中存在的错误更加敏感，进而会影响下游的各种分析工作。

表 1-1 各类算法的运行速度比较

	Single-end			Paired-end		
Program	Time (s)	Conf (%)	Err (%)	Time (s)	Conf (%)	Err (%)
Bowtie-32	1271	79.0	0.76	1391	85.7	0.57
BWA-32	823	80.6	0.30	1224	89.6	0.32
MAQ-32	19797	81.0	0.14	21589	87.2	0.07
SOAP2-32	256	78.6	1.16	1909	86.8	0.78
Bowtie-70	1726	86.3	0.20	1580	90.7	0.43
BWA-70	1599	90.7	0.12	1619	96.2	0.11
MAQ-70	17928	91.0	0.13	19046	94.6	0.05
SOAP2-70	317	90.3	0.39	708	94.5	0.34
bowtie-125	1966	88.0	0.07	1701	91.0	0.37
BWA-125	3021	93.0	0.05	3059	97.6	0.04
MAQ-125	17506	92.7	0.08	19388	96.3	0.02
SOAP2-125	555	91.5	0.17	1187	90.8	0.14

第二类算法基于前缀树，包括 Bowtie，SOAPv2，以及 BWA，这一类算法通过对参照序列进行 BWT 转换建立索引，之后利用在 BWT 转换序列上的后向搜索来模拟对前缀树的遍历过程，这类算法拥有几个显而易见的优点，首先是对参照序列建立一次索引之后，这个建立好的索引就可以永久使用，不必重复建立，再者，这类算法每次匹配一个短序列读入的时间与参照序列的规模无关，仅仅与读入序列的长度有关，第三，这种算法建立的索引消耗的内存空间比之前提到哈希表要少，并且可以通过各种办法进行进一步的压缩，第四，这一类算法功能齐全，可以应对包括基因的异位，以及缺失位，增添位等种种情况，并且可以根据调整对每一个对齐的匹配程度进行评分。最后，这类算法在速度上普遍优于之前提到的各种算法，而在准确率上却不输给之前提到的算法，因此这一类算法成为了目前主流的基因对齐算法。

其中 BWA 算法是目前使用范围最广的基因对齐工具之一，其可以接收 NGS

之中 illumina 平台以及 SOLID 平台的的读入，拥有多种对齐选项，并且可以为对齐的结果进行评分，其对齐的结果将以标准的对齐文件格式 SAM 输出，这个格式的文件有专门的 SAMtools 可以对对齐结果进行一系列的下游分析，因此我们将 BWA 算法作为本文的主要研究目标。

1.2.3 并行运算技术与显卡技术的发展

并行运算技术是指将原有的计算任务通过合理的划分形成可以同时执行的多个小单元，再由许多处理器同时处理划分出来的子问题，从而大大的降低程序运行的时间的技术。并行运算已经投入到应用当中很多年了，在初期，其使用主要集中于高性能运算的领域，如使用计算簇，系统网络作为并行运算的硬件来进行计算。但是最近，多核处理器的发展，使得并行运算技术可以在一台服务器主机甚至是个人 PC 上得到实现。

现代的图像处理单元 GPU 拥有着高度数据并行的构架，他们由许许多多个微小的核构成，每一个核上都有一些功能单元，例如 ALU，一个或多个这样的功能单元共同执行程序的一个线程，它们的集合被称作线程处理单元。同一个核上的线程处理单元都执行同一条指令，因为他们都共享着同一个控制单元。这就意味着 GPU 可以对一幅图像上的所有像素点执行相同的指令。也意味着其能够同时对不同的数据进行相同的处理，得到运算结果。

由于游戏产业的推动作用，近年来推出的 GPU 拥有着骄人的性能，英伟达最新的旗舰级显卡 GTX1080TI，采用了 pascal 构架，内含 3328 个流处理器，浮点数运算速度高达 10.8tflops，这个速度是 7 年前出产的 GTX480 的近 5 倍。与此同时，NVIDA 公司为自己的显卡设计了专用的并行运算平台，也就是 CUDA，该平台可以应用于 NVIDA 旗下的 GeForce，Quadro, 以及 Tesla 系列的 GPU 上，鉴于 NVIDA 显卡广阔的市场覆盖率，现阶段大多数电脑都可以使用经由 CUDA 平台编写的程序。CUDA 的出现使得程序员们可以使用接近于 C 语言的思考方式来编写并行运算程序，其最新的版本添加了对 printf，malloc 等高级函数的支持，并且提供了便捷的调试程序 NSIGHT，这进一步降低了并行运算的学习成本。

图 1-6 GTX 1080TI

基于以上事实，同时考虑到生物大数据的处理庞大的运算量，我产生了使用并行运算技术来进一步加速基因对齐算法的想法。

1.2.4 利用并行运算加速基因对齐算法

为了进一步提升基因对齐算法的效率，我们决定尝试使用并行运算的方式重新编写基因对齐算法，并行运算一直被视为处理生物大数据的最有效方法之一，近年来，已经有了许多使用并行运算成功加速对生物大数据处理的案例。

Barnarde 通过并行运算加速了一个用于寻找远距离同源序列的程序，远距离同源序列由于在区域上的相似度以及保守度很低，因此无法通过传统的序列对齐算法来侦测，因此人们开发了一些新的方法，通过构建一种叫做隐藏马尔可夫链的模型来进行特殊的对齐，这种方法可以帮助科学家们更好的寻找远距离同源序列，但是原先其效率较低，因此技术人员使用并行运算的方式对齐加以改进，使其可以运行在一个内部的集群之中，从而大大提高了运行的效率。

批注 [KW1]: 这些地方应该有引用标记

Crossbow 是用来从整个基因组序列中寻找 SNP（单核苷酸多态性）的工具，寻找 SNP 是为了可以更好的预测病人患特定疾病的概率。其中整合了很多对齐工具以及一些用于下游分析的算法工具如 Bowtie 以及 SOAPsnp。Cross 是基于 Hadoop 的，因此可以在簇跟云上进行基因组学的计算分析。但是其在处理大规模的 WGS 数据时还是会面临一些可扩展性，以及数据管理方面的问题。因此，人们又开发了开源，可扩展，在云系统上进行计算的 Rainbow，它可以将较大的文件分开到不同的云虚拟机器上去计算，进而提高了计算的效率。

同样也存在着一些使用并行运算技术来加速基因对齐算法的尝试，例如 SOAPv3，利用并行运算的方式，改进了原先的 SOAPv2 算法，使得该算法拥有着比使用单个处理器快的多的运算速度，实验证明该算法的运行速度达到了单核版本的 BWA 以及 Bowtie 算法的 7.5 到 20 倍。Heeseeung Jo, Gunhwan Koh 实现了对 BWA 转化输出部分的多线程执行，使得 BWA 算法可以更快的将中间数据结构转化成片段在基因组之中的位置进行输出。Weiguo Liu 等人在 2011 年开发设计了 CUDA-BLASTP 工具，借助高性能显卡的力量改造了古老的 BLASTP，一个用于做蛋白质序列分析的工具，其原理与基因对齐基本一致，使得在使用 GTX295 作为硬件运行的情况下，其速度比传统的 BLASTP 2.2.22 要快 10 倍，2012 年，Jose salavert Torres 等人，利用 BWT 转换原理，设计了新型的精准基因匹配的工具，比起原先的精准匹配程序提速达到了 12 倍。由于这些成功的例子，我认为具有高精准度，并且适用于不精准匹配的 BWA 算法同样可以通过 GPGPU 的算法改造来实现加速，因此我以此为课题做了以下的研究。

1.3 本文研究目标和内容

1.3.1 研究目标

验证通过 GPU 对 BWA 进行加速的可行性

文章 Fast and accurate short read alignment with Burrows - Wheeler transform 中详细描述了 BWA 短序列对齐的算法，我们发现，该算法的主体，即使用后向搜索进行的不准确匹配的单端对齐，是整个算法在正常情况下最耗时的部分， 我们希望利用现有的实验条件，根据 BWA 短序列对齐的论文，利用 GPU 并行运算的思想重新编写 BWA 短序列单端对齐算法的核心部分，同时编写相应程序的 CPU 单核运算版本，并生成模拟数据对两种算法进行测试，对比这两种算法的表现，以此来探究使用 GPU 加速 BWA 短序列单端对齐算法的可能性。为了验证这个可能性

批注 [KW2]: 引用记号

1.3.2 研究任务

针对这一目标， 我们计划编写一个能够便于进行实验测试的程序， 这个程序将满足如下条件：

（1） 生成 FMindex 的程序可以接受命令行输入输出，指定输入文件名并产生同名的输出文件

（2） 进行匹配的文件可以通过命令行选择使用 CPU 或是 GPU 模式下运行，同时可以选择匹配的短序列总个数，以及输出的对齐中间文件的文件名

（3） 编写一个产生测试序列的程序，改程序可以生成参照序列以及用于作为输入的短序列， 其中短序列可以调节出现的错误数以及长度，同时标记处短序列在参照序列之中的位置用于测试。

第2章 研究步骤

2.1 BWA 算法

BWA 算法是现在最主流的基因对齐算法之一，完整的 BWA 算法功能完备，速度可观，准确率高，支持不准确匹配以及双端匹配，不准确匹配时允许一定量的失配以及间隙（gaps），并可以以标准对齐格式 SAM 输出对齐结果，由于我所做的主要是 BWA 算法核心部分的实现，因此接下来我将介绍 BWA 算法核心部分的原理。

2.1.1 前缀树

字符串 X 的前缀树是一棵具有如下所述的特征的树，首先，树的每一条边都对应着一个符号，从树的一个叶到根的路径上所有的边的上的符号连起来可以构成 X 的一个唯一的前缀，由此我们可以知道。从这个前缀树的任意一个节点出发，到根的路径都是 X 的一个独一无二的子串，我们称这个子串为前缀树上对应节点所代表的的子串，由于 X 的前缀树与 X 的逆序列的后缀树是相同的，因此，对于后缀树成立的一些定理同样可以应用于前缀树。

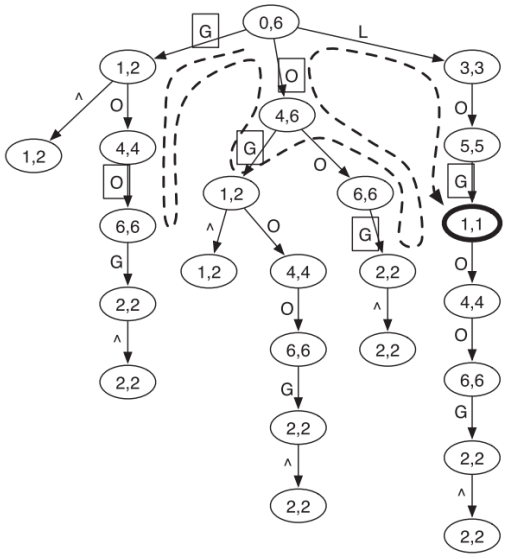


图 2-1 字符串“GOOGOL”的前缀树

有了前缀树，我们就可以更好的判断一个字符串是否是另一个较长字符串的子串，方法就是在那个较长字符串的前缀树中寻找代表着要查询字符串的节点，这项工作可以在 O(n)时间内完成，因为我们可以将查寻序列从尾到头一个符号一个符号的去从根开始匹配参照序列的前缀树的边。为了允许一些错误，我们可以用蛮力去遍历整个树，之后将查询序列向每一个可能的分支去匹配。上图给出了一个前缀树的实例，它所对应的字符串是“GOOGOL”，其中关于后缀数组区间（suffix array interval），我会在接下来的章节之中进行解释。

2.1.2 BWT 转换

BWT 转换是一种数据的编码方式，编码完成之后可以使得相似的数据聚集在一起，并且可以使用其中内含的信息来将编码完成的数据恢复成原先的数据。其编码的过程可以描述成，假设Σ是一个字母表，符号$并不在Σ这个字母表之中并且在字母序上小于Σ之中的所有符号。一个字符串 $X=a_0a_1\ldots a_{n-1}$ 是一个永远以符号$结尾的,且$只会出现在X的结尾的字符串。令X[i]=ai,i=0,1,….,n-1,是字符串X的第i个符号,,X[i,j]=ai…aj是字符串X的一个子串,Xi=X[i,n-1]是X的一个后缀，X的后缀数组（suffix array）S是从0到n-1的整形数的一个序列，满足S(i)是字母序排列第 i 小的后缀在 X 中的开始位置。X 的 BWT 转换B的定义如下，当S(i)=0时B[i]=$，否则B[i]=X[S(i)-1]，我们同时定义X的长度为|X|,由此可知|X|=|B|=n。

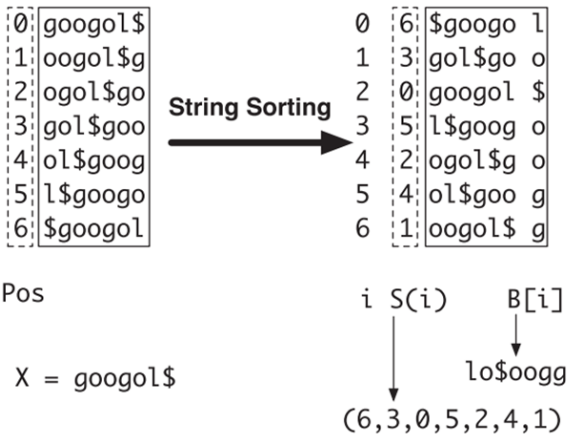


图 2-2 BWT 转换算法

上图给出了一个更简洁的实现 BWT 算法的例子，首先把要进行转换的字符串的末尾使用特殊符号$进行标记，注意到$在字母序上要小于所有在排列之中出

现的字母表的符号，之后将该字符串的第一个字符放到末尾，使原来第二字符变成移动后的第一个字符，这样一来得到了一个长度与原来相等的字符串，然后重复上面所描述的操作，知道$成为第一个字符，这时，这项操作已经进行了n次，并且得到了n个新的字符串，n即为字符串的长度，这之后将这些新的字符串连同原字符串按照字母序排列，就可以得到右面的表中所示的一组数组，将这些数组的最后一个字依次排列就得到了原先的字符串经过 BWT 转换得到的字符串，这之后如若想将 BWT 字符串重新还原成原字符串，只需要将转换得到的字符串中的字母按字母序排列一遍，之后我们就可以知道原字符串的最后一个字符的位点$以及字母的排列顺序，也就可以还原原字符串了。

批注 [KW3]: 直到

2.1.3 后缀数组区间与序列对齐

后缀数组区间是字符串 X 的子串所具备的一个固有的值，我们知道，如果 W 是 X 的一个子串，那么所有以 W 开头的 X 的后缀在进行 BWT 转换的排位这一步骤时都会被排在一块，这就意味着，W 在 X 之中所有可能出现的位点都会集中在后缀数组 S 的某个区间所显示的位置上。因此，我们可以做出如下的定义：

$$\underline{R}(W) = \min\{k : W \text{ is the prefix of } X_{S(k)}\} \quad (1)$$

$$\overline{R}(W) = \max\{k : W \text{ is the prefix of } X_{S(k)}\} \quad (2)$$

其中 $\underline{R}$ 代表着后缀数组区间的下界，$\overline{R}$代表着后缀数组区间的上界，普遍来说，如果 W 是一个空串，那么 $\underline{R}(W)=1$，$\overline{R}(W)=n-1$，区间$[\underline{R}(W), \overline{R}(W)]$被称为 W 的后缀序列区间（suffix array interval），W 在 X 中所有可能出现的位置构成的集合为$\{S(k): \underline{R}(W) \leqslant k \leqslant \overline{R}(W)\}$。知道了后缀序列区间我们就相当于知道了所查询字符串在参照字符串中的位置，这样一来，序列的对齐就等价于寻找匹配查询序列的 X 的子串所对应的后缀序列区间。对于精确匹配问题，我们只能找到一个对应的区间，对于不精确的匹配问题，我们可能会找到多个满足条件的区间。

2.1.4 准确匹配与后向搜索

首先定义两个数组：假设 C(a)是字符串 X[0，n-2]中字母序小于 a 的符号的数量，其中$a \in \Sigma$，假设 O(a,i)是在 X 的 BWT 转换 B[0,i]中出现的次数，Ferragina 与 Manzini 在 2000 年证明了如果 W 是 X 的一个子串的话，那么有：

$$\underline{R}(aW) = C(a) + O(a, \underline{R}(W) - 1) + 1 \quad (3)$$

$$\overline{R}(aW) = C(a) + O(a, \overline{R}(W)) \quad (4)$$

并且当且仅当 aW 是 X 的一个子串时，有 $\underline{R}(aW) \leqslant \overline{R}(aW)$。有了这个结果，我们就可以在线性时间内检测一个字符串是否为另外一个字符串的子串，方法就是先设 W 是一个空串，之后不断的把要检测字符串的最后一个字符加到 W 的前面，之后计算得到新的后缀数组区间，如果计算到最后依然满足后缀数组区间的下界小于上界，那么就说明这个字符串是参照序列的子串。这一过程被称作

后向搜索（backward search）

其实，后向搜索就是对前缀树的从上至下遍历的一种模拟，只要知道父节点的后缀数组区间，我们就可以在常数时间内计算子节点的后缀数组区间。这样一来，后向搜索就等价于在前缀树上作字符串的准确匹配，同时可以避免将整个前缀树的结构显式的存入内存。

2.1.5 不准确匹配

```
Precalculation:
  Calculate BWT string B for reference string X
  Calculate array C(·) and O(·,·) from B
  Calculate BWT string B' for the reverse reference
  Calculate array O'(·,·) from B'
Procedures:
  InexactSearch(W,z)
    CalculateD(W)
    return InexRecur(W,|W|−1,z,1,|X|−1)

  CalculateD(W)
    k ← 1
    l ← |X|−1
    z ← 0
    for i = 0 to |W|−1 do
      k ← C(W[i])+O'(W[i],k−1)+1
      l ← C(W[i])+O'(W[i],l)
      if k > l then
        k ← 1
        l ← |X|−1
        z ← z+1
      D(i) ← z
```

```
InexRecur(W,i,z,k,l)
  if z < D(i) then
    return ∅
  if i < 0 then
    return {[k,l]}
  I ← ∅
  I ← I ∪ InexRecur(W,i−1,z−1,k,l)
  for each b ∈ {A,C,G,T} do
    k ← C(b)+O(b,k−1)+1
    l ← C(b)+O(b,l)
    if k ≤ l then
      I ← I ∪ InexRecur(W,i,z−1,k,l)
      if b = W[i] then
        I ← I ∪ InexRecur(W,i−1,z,k,l)
      else
        I ← I ∪ InexRecur(W,i−1,z−1,k,l)
  return I
```

图 2-3 不准确匹配算法

上图给出了一个不准确匹配的算法，用于搜索与 W 匹配的子串 SA 区间的算法。参照序列 X 以$结尾，W 以 A/C/T/G 四种碱基结尾，函数 INEXACTSEARCH(W,z)返回与 W 匹配且错误不超过 z 个的子串的 SA 区间，INEXRECUR(W,i,z,k,l)递归的计算在 w_(i+1)与区间[k,l]匹配的情况下，与 W[0,i]匹配且错误不超过 z 个的子串的 SA 区间。以星号开头的行是为了应付基因序列插入与删除的情况，另外，D(i)是字符串 W[0,i]中可能含有的错误的下界。本质上讲，这个算法使用后向搜索来取样出基因组中不同的子串。这个过程以序列 D（）为界，其中，序列 D 是序列 W[0,i]与 X 匹配的编辑距离的下界。D 被估计的越精确，搜索的范围就会越小，算法的效率就会越高。最简单的设计思路是让 D(i)=0，即不使用 D 数组，但是这样一来所得到的算法复杂度会随着允许错误数的增多呈指数增长，从而变得低效。

为了理解 D 所起到的作用，我们回到之前的例子中，在字符串 X=GOOGOL$之中查找 W=LOL。如果我们把所有的 D(i)都设置成 0 并且不支持间隔（gap）的话（去掉算法的头两行），函数 INEXRECUR 的调用图将会是一棵树，它实际上会模仿图一中虚线所显示的搜索过程。然而，如果加入 CALCULATED 函数，我们就会知道 D(0)=0，D(1)=D(2)=1，这样我们就可以避免下降到 G 与 O 的子树上，从而减小了搜索的范围。

2.2 BWA 算法的 CPU 部分实现

BWA 算法可以大致分为三个部分，第一部分是将基因序列经过一系列的编码，包括 BWT 转换，构成索引 FM-index 的过程，第二步就是读入将要匹配的输入序列，之后利用后向搜索的原理进行匹配的过程，第三部是将匹配得到的结果 SAI 文件，转换为标准输出文件的过程，其中，第一步对于一个人类基因组参照序列来讲仅仅需要执行一次，且消耗的时间较少，因此可以选择在 CPU 上进行执行，而第二步，也就是对齐的过程，是最消耗时间的过程，也是限制整个对齐过程速度的关键，由于在这一部分中可以让多个读入序列同时进行匹配，所以可以利用并行运算的方式进行加速，因此是我计划利用 GPGPU 进行加速的部分，第三步转化输出是一个非常简单的过程，使用 GPU 加速的意义并不是很大，我们这里不做研究。

因此，我们的程序的第一部分，也就是构成 FM-index 的部分，计划将在 CPU 上执行。这一部分可以读入一个基因文件，基因文件为.ge 其格式为由八位的字符构成的二进制文件，这一类文件可以由两位表示一个基因的通用文件轻松转化得到。之后输出转化完成后的 FM-index，FM-index 将被储存到.fm 文件之中去，其中 fm 文件的文件格式为二进制，其中的内容依次是参照序列的长度（占用 32 位，人类基因组的长度要大于 32 位，这里使用 32 位作长度的储存仅用于实验），BWT 转换完成的数组里$的位置，这个位置在后向搜索中会用到。这之后是 BWT 转换的结果，之后存放的是数组 C 以及数组 O，其作用在之前的原理部分介绍过，四种碱基的存放顺序依次为 A,C,G,T，每个数组的第一个数值是对应碱基的 C，之后是 O。出于节省内存空间的目的，我们没有把原数组存入 FM-index 文件之中，原数组可以通过 BWT 的逆变换轻松得到。

2.3 BWA 算法的 GPU 部分实现

在 GPU 上实现的是算法的主体对齐部分，这一部分分为三个步骤执行，第一步是将信息从文件之中读入内存，之后传入 GPU 的显存之中，第二步是执行对齐，找到查询序列对应的后缀序列区间，第三步是将结果拷贝回内存并输出到文件中。

其中第一步较为容易，需要注意的是我使用的 FM-index 是一个使用多级指针的较为复杂的结构，因此拷贝的过程之中不能直接使用 cuda 自带的传送函数 cudaMemcpy，而要多次调用这个函数，从而在显卡之中正确的重现 FM-index 的 struct。

前面的章节提到过，使用显卡进行不准确匹配使用的是一个递归的程序，但是由于 GPU 自身的限制，当递归的层数过多时，程序就会崩溃，因此我们重新建立了一个栈，栈中记录着每次递归调用的参数以及运行到的阶段数。由于 NGS 平台上单个可以匹配的基因片段的长度基本不会超过 200bp，同时，BWA

在保证准确率以及效率的情况下允许的最大输入长度就是 200bp，根据算法，我们可以推断出栈中的 snapshot 数不会超过 200 个，因此，栈的总大小不会超过 4KB，对于拥有 2G 显存的 GTX 850m 来说，我们在 640 个核上同时运行，也只会占用不到 3MB 的显存，这个占用程度是完全可以接受的。

之前的章节提到过，当进行不准确匹配的时候，可以有多个符合条件的后缀序列区间，但是 CUDA 中对于显卡程序的返回有一定的时间限制，因此，在一些情况下，我们无法让对齐的程序完整的运行找到所有匹配的后缀序列区间，幸运的是，在大多数情况下，不需要遍历整个前缀树就可以找到满足搜索条件的与查询序列匹配的参照序列的子串，但是，由于在庞大的参照序列中可能存在着许多个满足条件的子串，因此我们不能确定找到的一定是查询序列应该被匹配到的正确位置，一般来说，匹配到的子串与查询序列之间的编辑距离越小，其是正确匹配的可能性就越大。在进行不准确匹配的时候，根据算法的原理我们很容易得知，同一个匹配，可以在不同的前缀树的子树上被对此检索到，经过多次试验，我发现，与查询序列编辑距离较短的匹配，其出现的几率就会越大，因此，为了兼顾程序的运行效率，我将程序设计成当一个后缀数组区间连续两次出现时，就认定其为正确的匹配结果予以输出。对于最后选定的同一个后缀数组区间中所代表的不同位置，我们可以通过比较原位置的子序列以及参照序列来准确的找到编辑距离最小的匹配，这个过程非常简单，可以通过 BWT 的逆变换轻松找到，可以在第三个步骤中再进行实现，这里不作讨论。

2.4 对比测试

接下来的任务是编写 CPU 版本的 BWA 程序,之后将其与 GPU 版本的 BWA 程序进行对比,由于 GPU 版本的程序与 CPU 版本的程序的主要区别就在于第二阶段的运行是位于 GPU 上的，因此我们只需要测量两个版本的程序在第二阶段完成对一定量输入的匹配速度就可以知道使用 GPU 加速 BWA 算法这一思路是否可行。

由于测试机器的显存容量问题，我无法使用真正的人类基因组规模的参照序列进行测试，但是我们知道，显存问题对于计算用显卡来说不是问题，因为最新的计算用 gpu 基本都有 20G 以上的显存，如 tesla P40， 就拥有高达 24GB 的显存容量，足以容纳整个人类基因组的 FM-index，另外，观察上面提到的基因对齐算法我们可以知道，程序运行的时间与参照序列的规模是基本无关的，仅仅与查询序列的规模相关，所以，使用一个较小的参照序列所得到的实验结果基本可以代表使用大规模参照序列的实验结果。

由于在 NGS 不同的测序平台上，产生的读入序列长度存在差异，因此我计划在不同的输入长度下，比较 GPU 版本的 BWA 算法与 CPU 版本的 BWA 算法的运行效率。经过文献查阅，我发现 32bp 以及 100bp 为比较常用的两个输入长度，因此我将分别测试两个版本的算法在这两个长度下的表现。对于一般的基因序列。

测试方法如下：首先随机生成一段 10M 的参照序列，根据之前提到的，参照序列的长度是不影响匹配速度的，之后在这段参照序列的随机位点截取一段 32bp 或 100bp 的序列作为查询序列，每个长度下各截取 5000 个查询序列，对于长度为 100bp 以内的序列，我们认为其匹配的条件是编辑距离小于 5，因此我们将这些截取出来的序列随机更改掉其中的 5 个位置上的碱基，之后分别在 CPU 版本以及 GPU 版本的 BWA 程序上测试当有 50,100,250,500,1000,1500,2000,2500,3000,3500,4000,4500,5000 个输入查询序列时，全部对齐完成后所消耗的时间。测 5 次，去掉最高与最低的时间，剩下三个数据取平均值最为该数目输入下消耗的时间。

批注 [KW4]: 顿号

第3章 实验结果

3.1 测试环境

测试环境如下：
操作系统：windows 8.1
CPU 型号：intel i7-4710M
GPU 型号：GTX 850m

批注 [KW5]: 注意大小写

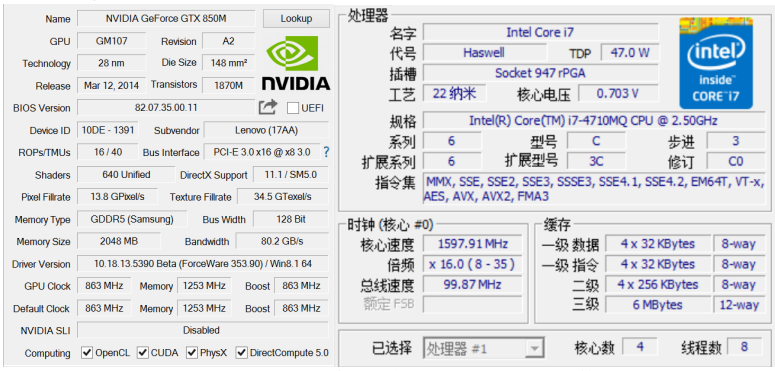


图 3-1 测试环境所用的 GPU 以及 CPU 参数

3.2 实验结果

表 3-1 输入长度 32bp，允许编辑距离 5，GPU,CPU 版本 BWA 匹配用时对比

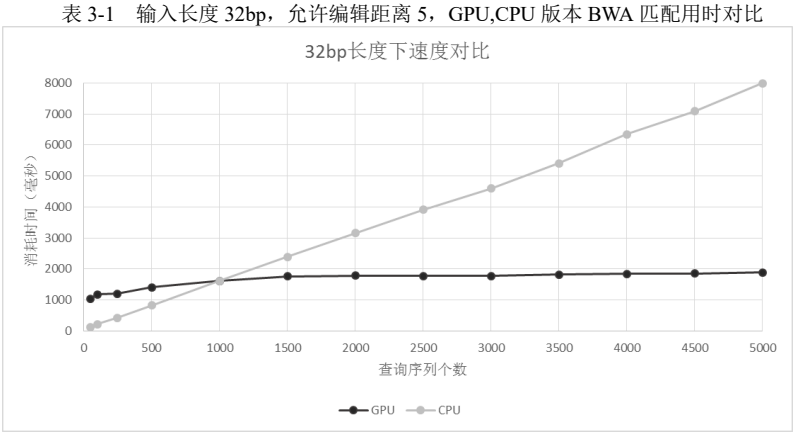


表 3-2　输入长度 100bp，允许编辑距离 5，GPU,CPU 版本 BWA 匹配用时对比

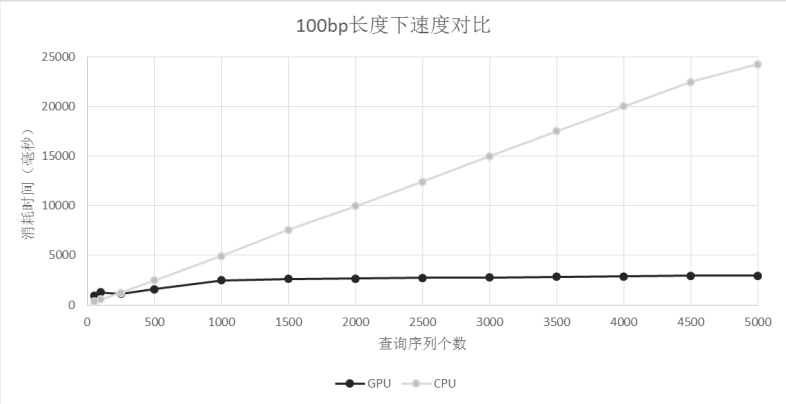


表 3-3　输入长度 100bp，精准匹配，GPU,CPU 版本 BWA 匹配用时对比

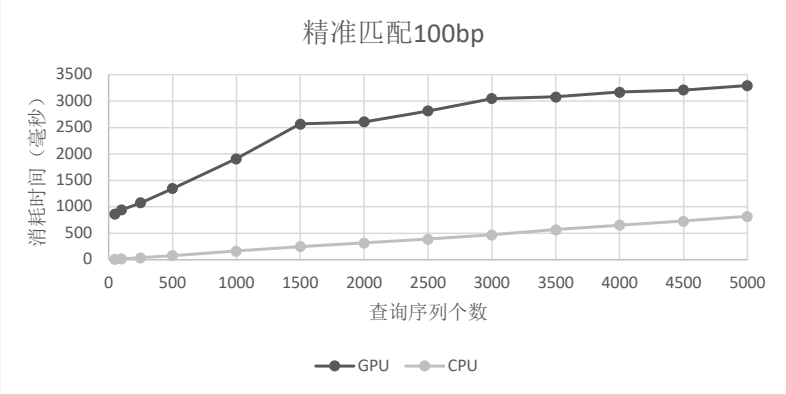
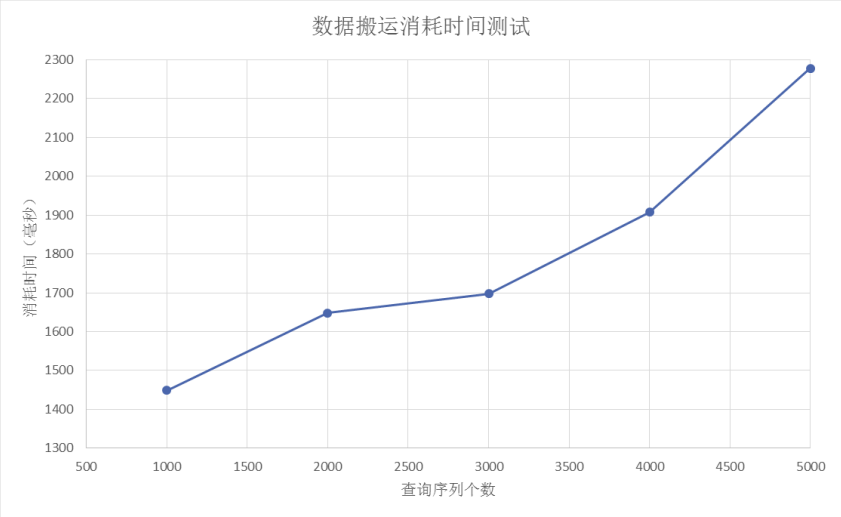


表 3-4 原始实验数据表格

读入长度100bp		单位ms		读入长度32bp		单位ms		精准匹配100bp		单位ms
读入个数	GPU	CPU		读入个数	GPU	CPU		读入个数	GPU	CPU
5000	2965	24241		5000	1884	7989		5000	3293	822
4500	2962	22478		4500	1853	7088		4500	3210	729
4000	2894	20002		4000	1846	6341		4000	3167	653
3500	2853	17515		3500	1812	5411		3500	3077	567
3000	2781	14999		3000	1775	4594		3000	3045	472
2500	2730	12426		2500	1772	3914		2500	2811	389
2000	2659	9981		2000	1784	3152		2000	2605	319
1500	2615	7591		1500	1762	2394		1500	2564	250
1000	2500	4954		1000	1611	1612		1000	1911	166
500	1609	2478		500	1411	823		500	1346	81
250	1125	1250		250	1195	423		250	1077	40
100	1281	578		100	1175	219		100	942	16
50	922	390		50	1035	118		50	865	11

表 3-5 GPU 程序中数据搬运所消耗的时间测试



3.3 实验结果分析

从上面的实验结果，我们可以看出，无论是对于 32bp 的短序列，还是 100bp

的较长序列，当处理的读入个数达到一定数量的时候，GPU 的效率就开始高于CPU，对于 32bp 的序列来说，这个临界值约为 1000 个读入，对于 100bp 的序列来说，这个临界值在 250 个读入。并且，随着读入数量的上升，CPU 版本的 BWA 算法的增长率要远远高于 GPU 算法，其增长率具体大约为 GPU 算法的 23 倍左右，也就是说在相同实验条件下，在输入数量足够大的情况下，使用 GPU 加速可以将 BWA 算法的速度提升 23 倍。这显示着使用 GPU 对 BWA 算法进行加速是可行性很高的。

但是我们同时应该注意到，当我们使用没有任何错误的查询序列进行测试时，如果设置对齐允许的最大编辑距离为 0，即不允许任何错误出现，这样的话，当使用 5000 个以内输入序列进行测试时，CPU 版本的表现要明显优于 GPU，分析算法我们可以发现，当设置允许最大的编辑距离为 0 时，递归的深度将大大的降低，因此程序运行的时间应该会显著减少，但是实际情况是，CPU 版本的 BWA 程序的运行时间显著降低，但是 GPU 版本的程序并没有出现显著的运行时间的减少，特别是在输入序列较多的情况下。我们认为造成这种现象的原因主要有两方面，首先，将输入序列以及 FM-index 结构传入 GPU 要消耗掉很长一部分时间，同时将结果拷贝回内存也需要消耗掉一部分时间，通过测试我们可以看出在数据转以上消耗的时间经常会占到整个程序运行时间的 2/3 以上，但是在实际的工作情况中，由于我们可以使用更大容量显存的显卡，以及读入的输入序列数要远大于 5000，因此移动数据所占用时间的比例推测会降低到可以接受的程度，尽管如此，我们还是需要注意在这上面时间的消耗，尽可能的在设计程序是避免无用的数据搬运。第二点可能是因为，GPGPU 与 CPU 使用不同的指令集与内存访问模式，因此在编译 calculateD 等函数时采取了不同的策略，因此导致 GPGPU 对允许错误的个数并不敏感。在实验的后期我又补充简单测试了允许十个错误发生的情况，实验结果显示，GPU 版本所消耗的时间与允许 5 个错误时差距不明显，而 GPU 版本的速度明显要慢于允许 5 个错误的版本。

第4章 分析与讨论

由实验结果，我们可以分析得知。对于长度为 32bp 的短输入序列，使用 GPU 加速过的 BWA 算法可以在 1.88 秒之内匹配完 5000 个输入序列，而使用 CPU 则需要 7.99 秒，对于长度为 100bp 的输入序列，使用 GPU 加速过的对齐算法可以在 2.97 秒内完成 5000 个输入序列的对齐，相比之下 CPU 版本则需要 24 秒以上，由此可见，使用 GPU 加速过的对齐算法在处理多个读入序列是表现要明显由于 CPU 版本的对齐算法。虽然在进行精准匹配的时候，CPU 表现出了更好的性能，但是在实际应用过程中，因为事先无法得知某一段序列中是否存在突变，也无法的知某一段序列在测序过程之中是否出现过错误，因此我们有必要在运行时根据读入序列的长度，来设置一个允许的编辑距离范围，而在这种情况之下，GPU 的对齐效率要明显高于 CPU。

根据以上的实验结果，我们可以认为使用并行运算的方式来加速 BWA 拥有着不错的前景，但是如果要将其投入使用，还需要注意几个问题，首先，我们通过实验得知了，在内存与显存之间搬运数据产生的时间消耗是很大的，因此应尽量减少内存与显存之间的数据交换次数，避免使用内存作为虚拟显存来运行程序的情况，要打到这一目的，一方面要使用大容量的显存，另一方面要通过一些压缩算法缩小 FM-index 的体积。其次，要深入理解显卡的内存构架，在显卡上执行计算时，从不同的显存区块中调用数据的代价相差极大，我认为这也是造成 GPU 版本程序在精准匹配过程之中速度较慢的原因之一，因此在编写可以投入使用的 GPU 版本的 BWA 程序时应该合理的分配显存，使得 GPU 中的每个核可以用最快的速度访问到需要的数据。第三，尽管 BWA 理论上可以支持任意长度的输入序列的对齐，但是实际上随着输入序列长度的增加，算法的效率会迅速降低，因此，官方建议将输入序列的长度限制在 20bp 到 200bp 之间，另外，用于显示的 GPU 通常有运行时长限制，在处理超过 5000 个的输入时，我们需要解除这个限制，这在 linux 系统之中比较容易做到，在 windows 系统当中，如果要兼顾显示功能，建议将核函数分段执行，或者整合成流的形式，尽量避免频繁的数据搬运。

由于实验条件的限制，我们未能使用最为先进的 GPU 进行测试，尽管如此，我所编写仅供实验使用的 GPU 版本的 BWA 算法在 gtx850m 上的运行速度仍然是官方提供的最初版本的 BWA 的 2 倍以上，这充分证明了使用 GPGPU 加速基因对齐算法，以及其他生物大数据处理算法上的庞大潜力。

参考文献

[1] Heng Li, Richard Durbin " Fast and accurate short read alignment with Burrows–Wheeler transform", BIOINFORMATICS, Vol. 25 no. 14 2009, pages 1754–1760

[2] Heng Li, Richard Durbin, "Fast and accurate long-read alignment with Burrows–Wheeler transform" , BIOINFORMATICS, Vol. 26 no. 5 2010, pages 589–595

[3] Ben Langmead, "Aligning short sequencing reads with Bowtie", Curr Protoc Bioinformatics. 2010 December ; CHAPTER: Unit–11.7

[4] Chuming Chen, Sari S Khaleel etc. "Software for pre-processing illumina next-generation sequencing short read ", Source Code for Biology and Medicine, 2014,9,8

[5] Turki Turki, Usman Roshan, "MaxSSmap: a GPU program for mapping divergent short reads to genomes with the maximum scoring subsequence", BMC Genomes, 2014, 15:969

[6] Heeseung Jo3, Gunhwan Koli "Faster single-end alignment generation utilizing multi-thread for BWA", Bio-Medical Materials and Engineering 26 (2015) S1791-S1796

[7] Dale Muzzey , Eric A. Evans , Caroline Lieber "Understanding the Basics of NGS: From Mechanism to Variant Calling", CUIT Genet Med Rep (2015) 3:158-165

[8] Ruiqiang Li, Yingrui Li, Karsten Kristiansen and Jun Wang, "SOAP: short oligonucleotide alignment program", Bioinformatics, Vol. doi:24 10.1 no. 093/ 5 bioinformatics/ 2008, pages 713-714

[9] Chi-Man Liu, Thomas Wong, etc "SOAP3: ultra-fast GPU-based parallel alignment tool for short reads", Bioinformatics, Vol. doi:28 10.1 no. 093/bioinformatics/bts06 6 2012, pages 878-879

[10] Kary Ocaña1, Daniel de Oliveira, "Parallel computing in genomic research: advances and applications", Applications in Bioinformatics and Chemistry, 13,nov,25

[11] NVDIA, "CUDA C PROGRAMMING GUIDE", Sep 20150

[12] Heng Li, Jue Ruan, etc, “Mapping short DNA sequencing reads and calling variants using mapping quality scores”, Genome Research 1851, 2008

[13] Hui Liang Khor , Siau-Chuin Liew and Jasni Mohd, “A Review on Parallel Medical Image Processing on GPU”, ICSECS, August 19-21, 2015

[14] Hongjian Li, Bing Ni, etc “A Fast CUDA Implementation of Agrep Algorithm for Approximate Nucleotide Sequence Matching”, IEE, 978-1-4577-1213-5/11, 2011

致谢

本科生毕业论文（设计）任务书

一、题目：

二、指导教师对毕业论文（设计）的进度安排及任务要求：

起讫日期 200 年 月 日至 200 年 月 日

指导教师（签名） 职称

三、系或研究所审核意见：

负责人（签名）

年 月 日

毕 业 论 文（设计） 考 核

一、指导教师对毕业论文（设计）的评语：

指导教师(签名）

年 月 日

二、答辩小组对毕业论文（设计）的答辩评语及总评成绩：

成绩比例	文献综述占（10%）	开题报告占（20%）	外文翻译占（10%）	毕业论文（设计）质量及答辩占（60%）	总 评成绩
分值					

答辩小组负责人（签名）

年　月　日