doi: 103969/j. issn. 0490-6756. 2017. 03. 010

基于 GPU 的大规模基因片段并行匹配的方法

丁　莎[1,2], 赵士元[2], 林　涛[1]
(1. 四川大学计算机学院, 成都 610065; 2. 四川大学锦江学院计算机学院, 眉山 620860)

摘　要: 后缀树和后缀数组广泛用于生物信息学领域中,特别是通过启发式算法在对 DNA 基因片段进行匹配的阶段. 本文提出了在 GPU 的平台下,利用多核和超多核体系构成的后缀树以及后缀数组并行匹配大规模基因片段,从而加速基因搜索匹配过程. 相对于后缀树,后缀数组二分搜素算法具有内存占用少,缓存使用率高等优点. 在 GPU 的性能评估中,后缀数组执行效率明显超过后缀树,后缀数组占用的空间仅为后缀树的 20%～30%. 相对于 CPU 的串行实现,后缀树组达到了约 99 倍的加速比. 实验结果表明在基因片段匹配的过程中,基于 GPU 的后缀数组二分搜索是一种高效且实用的方法.

关键词: 后缀数组; 后缀树; GPU; 基因片段匹配; 并行
中图分类号: TP316　　**文献标识码**: A　　**文章编号**: 0490-6756(2017)02-0280-07

The new approach of multiple genome sequence parallel matching based on GPU

DING Sha[1,2], *ZHAO Shi-Yuan*[2], *LIN Tao*[1]
(1. College of Computer Science, Sichuan University, Chengdu 610065, China;
2. College of Computer Science, Jinjiang College of Sichuan University, Meishan 620860, China)

Abstract: Suffix trees and suffix arrays have been used widely in bioinformatics applications, especially for DNA sequence alignments in the initial exact match phase of heuristic algorithms. In this paper, a new GPU implementation and optimization of the suffix tree and suffix array on both multi-core and many-core platforms to accelerate multiple genome sequence searching is presented. The comparative performance evaluation between the suffix tree and suffix array is then carried out. The results showed that the suffix array needed only 20%－30% of memory space compared with the suffix tree, and that the mean search time of the suffix array was significantly shorter than the mean search time of the suffix tree because of the use of a binary search with coalesced memory access and tile optimization under the GPU architecture. Moreover, the GPU implementation of the suffix array gained a speedup of approximate 99 times compared with the corresponding CPU serial implementation. This study showed that the massively parallel sequence matching algorithm based on suffix array was an efficient approach with the high-performance in the process of multiple DNA sequence matching.

Keywords: Suffix array; Suffix tree; GPU; Genome sequence matching; Parallel

收稿日期: 2016-07-01
基金项目: 四川省科技厅支撑项目(2012GZ0091, 2013GZX0138); 四川大学青年教师科研启动基金(2015SCU11050)
作者简介: 丁莎(1983－), 女, 四川富顺人, 博士生, 研究方向为生物信息学, 计算机教育等. E-mail: dingsha73200@163. com
通讯作者: 林涛. E-mail: lintao@scu. edu. cn

1 引　言

近年来，现代化的多核和众核体系结构以其高性能计算的特点应用于计算领域中. 随着越来越多的处理核心被集成在单个芯片上，市场上涌现出大批超多核体系结构，例如基于统一计算设备构架 CUDA(Compute Unified Device Architecture)的 GPUs[1]，以及其他类似于 FPGA 于 Cell/BE 的加速器技术. 从 2007 年引入 CUDA 以来，数亿台配有 GPU 的计算机投入市场，较低的门槛限制以及高效浮点数执行吞吐量，促使许多的研究者对新算法及其应用展开研发，GPU 计算的黄金时代已经来临.

生物信息学是 GPU 计算的最主要的应用领域之一. 将高吞吐量技术应用于 DNA 测序和基因表达分析过程中，促使相关数据急剧增长. NCBI GenBank 数据库中 DNA 序列信息的增长以及 UniProKB/TrEMBL 数据库中蛋白质序列的增长都非常明显. 此外，下一代 DNA 测序技术跨越了实验受基因测序规模限制的障碍，为生命科学领域带来了新的生机[2,24]. 由于 GPU 性能远远超过 CPU，因此基于 GPU 的相关应用更为实用.

随着大量的科研工作在生物信息学领域的展开，国内外针对 DNA 序列相似性的方法也不断出现. 在国外研究中，从局部最优的序列比对算法，到 FASTA 算法的提出，再到经典算法 BLAST 的出现，2012 年 Treangen 提出了关于重复基因序列的计算挑战和解决方法，这些方法都为加快基因序列的比对提供了好的方法[3,4]. 国内在 DNA 序列分类问题上，主要有序列比对、统计模型、数据挖掘和传统机器学习分类法等. 孙晓楠提出了 RBF 神经网络分类法，着重从碱基的数量和碱基的排列顺序来分析 DNA 序列. 郑俊生提出了基于 Gauss-EMD 的 DNA 序列相似性分析[5-7].

字符串的后缀树索引结构是以其所有后缀的字典树形式构成，被广泛的应用在生物信息学中，例如 MUMmer 和 MUMmer GPU[8-10]. 许多算法能够使后缀树在线性时间内被构造出来[11-13]. 然而，随着参考序列的增长，后缀树会触发内存(DRAM)消耗的瓶颈. 因为缓存的使用以及空间占用率仅为后缀树的 20%～30%，后缀数组经常作为后缀树的替代使用. 同时后缀数组也能通过 DC3 算法在 $O(n)$ 时间内构造出来[14,15]. 在多核(CPUs)和众核(GPU)的两种平台下，本文利用 GPU 实现后缀树和后缀数组的并行匹配算法来加速进行基因序列匹配. 实验数据表明，相对于 CPU 串行的实现方式，后缀数组获得了接近 99 倍的加速，同时后缀树也有 44 倍的性能提升.

2 大规模平行处理器

从 2003 年以来，硬件厂商主要是通过两种策略来设计微处理器，众核更专注于并行程序的执行吞吐量，反之，多核策略在于维持串行程序被移植在多核上的执行速度[16]. 众核是由大量的微型核心构成，并且核心数量往往在新一代设备中得到加倍. NIVIDIA GPU 就是一个典型例子，其每个核心都是基于有序的多线程，单指令处理器，同时与其他核心共享控制和指令缓存. 由于这两种类型的处理器在设计原理上的区别，造成了性能的巨大差异，如图 1 所示.

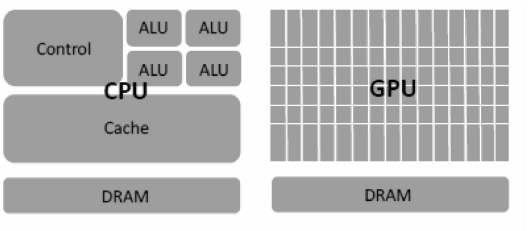

图 1　CPU 与 GPU 的区别

Fig. 1　The difference between CPU and GPU

早期通用 GPU 编程，因为编码复杂而难以得到广泛应用，而逐渐被统一计算设备构架(CUDA)所取代[10]. CUDA 程序不再通过图形接口，而是通过硅片上新的通用并行程序接口来影响 CUDA 程序. 在 CUDA 编程模型中，一个网格的所有线程执行相同的核函数. 网格中所有的线程以唯一的坐标来进行区分，同时用于识别预想的数据部分进行并行处理. 所有的线程通过坐标被统一组织成二级结构——块索引(blockIdx)和线程索引(threadIdx). *gridDim* *ridDim* 和 *blockDim* 分别表示网络维度以及每个块的维度. 根据 *gridDim* 和 *blockDim*，动态划分会引发有限资源之间的相互作用，例如，共享内存和寄存器.

在现代软件的应用中，程序的某个部分往往表现出大量的数据并行性，其中一个特性就是允许多个算法在其数据结构上安全执行. CUDA 设备就是利用该数据的并行性来加速应用程序的执行[10,17,18]. 近年来，在科学研究和商业应用中，许多高效解决问题的新技术如雨后春笋般涌现，除 CUDA 之外，还有

OpenCL,OpenACC 和 C++AMP [19-21].

3 构建后缀数组

我们给定一个参考序列 $S=s_1s_2\cdots s_n$,$i=1,2,\cdots,n$,$S(i,n)$ 表示的一个 S 后缀. 定义组成 DNA 的 4 个碱基对,腺嘌呤,鸟嘌呤,胸腺嘧啶,以及胞嘧啶为 $\sum=\{a,c,g,t\}$. 根据开始位置的字符来定义后缀. 如图 2 所示,$S=acggtacgtac$,$S_2=cggtacgtac$,$S_8=gtac$. 长度为 n 的参考序列 S 的后缀树结构具有以下特性:

(1) 任意一条 S 的子序列对应一条树边.

(2) 任意一个内部节点至少有两个子节点.

(3) 对于 $1<i<n$,S_i 对应一条从根到叶节点的标记路径.

(4) 同一内部节点没有分支是以相同字符开头的.

标准后缀树形式如图 2 所示.

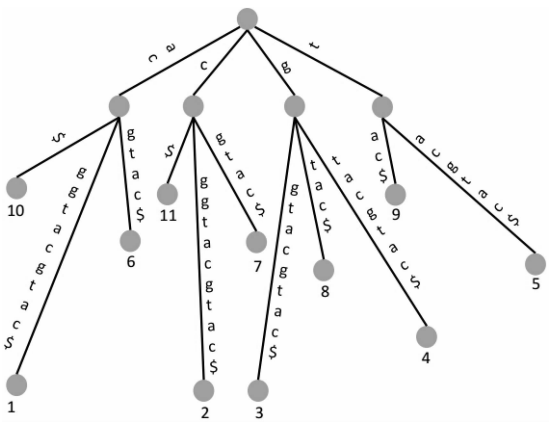

图 2 参考序列为 的后缀树

Fig. 2 The suffix tree of reference sequence

根据参考序列,在 $O(n)$ 线性时间内,后缀树能够被构造出来[17]. 然而后缀树却存在一个巨大的缺陷,其需要大量的空间来存储索引结构,而该结构的空间消耗接近初始参考序列大小的 20 倍. 当然,后缀树也可以通过转换成由数组构成的隐式树,其基于 GPU 的加速基因匹配已由 Encarnaijao G 等于 2012 年提出[22]. 即使如此,相对于后缀树,后缀数组仍被当作是一种节省内存空间的数据结构. 后缀数组是由参考序列的所有后缀按字典序构成的整数数组,如图 3 和表 1 所示.

表 1 参考序列 $S=acggtacgtac$ 的后缀

Tab. 1 The suffix array of reference sequence

Index	0	1	2	3	4	5	6	7	8	9	10
SA	10	1	6	11	2	7	3	8	4	9	5

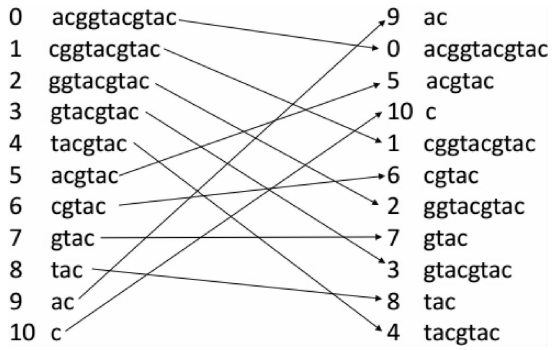

图 3 所有后缀按数据字典排列

Fig. 3 Sorting all the suffix by data dictionary

构造后缀数组最直接的方法是简单地创建一个数组,并通过后缀在原始参考序列的位置来进行赋值,之后使用排序算法来排序这些后缀. 本文利用 DC3 算法在 $O(n)$ 时间内实现后缀数组.

线性时间的后缀数组构造算法 DC3,其采取 2/3 递归分治策略:

(1) 开始位置满足 $i \bmod 3 \neq 0$ 的后缀被构造成后缀数组.

定义 $B_k=\{i\in[0,n]\,|\,i \bmod 3=k\,|\}$ 对于 $k=0,1,2$,$C=B_1\cup B_2$ 表示样本位置的集合,S_C 表示其样本的后缀集合. 对于参考序列 $S=acggtacgtac$,通过 $B_1=\{1,4,7,10\}$,$B_2=\{2,5,8\}$,推出 $C=\{1,4,7,10,2,5,8\}$. 当 $k=1,2$,构造序列 $R_1=[cgg][tac][gta][c00]$,$R_2=[ggt][acg][tac]$ $R=R_1\odot R_2=[cgg][tac][gta][c00][ggt][acg][tac]$. 如表 2 所示,利用基数排序算法[23],对所有的字符块进行升序排列,并重新获得排名 $\mathrm{rank}(S_i)$.

表 2 $S=acggtacgtac$ 样本后缀的排序

Tab. 2 The sorting table for $S=acggtacgtac$

i	0	1	2	3	4	5	6	7	8	9	10
Rank(S_i)	−	3	4	−	7	1	−	5	6	−	2

(2) 利用第一步得到的结果辅助构造剩余后缀的后缀数组.

非样本后缀 $S_i\in S_{B_0}$,通过 $(S_i,\mathrm{rank}(S_{i+1}))$ 对来描述. 对于所有 $i,j\in B_0$,$S_i\leqslant S_j\Leftrightarrow(S_i,\mathrm{rank}(S_{i+1}))\leqslant(S_j,\mathrm{rank}(S_{j+1}))$. 同样的,基数排序中所有的对. 如果 $S=acggtacgtac$,$B_0=\{0,3,6,9\}$. 因为 $(a,2)\leqslant(a,3)\leqslant(c,5)\leqslant(g,7)$ $S_0\leqslant S_9\leqslant S_6\leqslant S_3$.

(3) 使用标准比较方法来归并或合并以上两个后缀数组. 为了比较后缀 $S_i\in S_C$ 与 $S_j\in S_{B_0}$,本

研究需要区分两种情况：$i \in B_i$，$S_i \leqslant S_j \Leftrightarrow (S_i, \text{rank}(S_{i+1})) \leqslant (S_j, \text{rank}(S_{j+1}))$ $i \in B_2$，$S_i \leqslant S_j \Leftrightarrow (t_i, t_{i+1}, \text{rank}(S_{i+2})) \leqslant (t_j, t_{j+1}, \text{rank}(S_{j+2}))$.

由上述的 DC3 算法策略很容易证明其时间复杂度为 $O(n)$，同时基于该算法的并行分层存储器计算模型已经在文献[14,25]中被提出来了.

4　索引平行匹配

当所有的后缀被排序以后，索引结构用于搜索匹配模型序列. 如果参考序列 $S = acggtacgtac$，根据表 1 和图 3，$S_5 = cggtacgtac$ 和 $S_7 = ggtacgtac$. 如果 $P = c$，就位于排序后缀引表的 4−6 索引位置. 同理，$P = a$ 出现在 1−3 索引位置，即参考序列 S 的 10,1,6 索引位置. 这表明了对于任意出现在参考序列中的模型序列 P 都对应于排列后缀引表的一个索引范围，该范围是以左边界 LB 和右边界 RB 来表示. 如果 $P = a$，得到 $LB = 1$ 和 $RB = 3$. 如果 $P = c$，得到 $LB = 4$ 和 $RB = 6$. 对于 $P = ggtac$，$LB = RB = 7$. 因此，我们的任务在于搜索 LB 和 RB.

给定一个模型序列 P，二分搜索算法属于理想策略. 在二分搜索时，比较 P 与后缀组 S_i 的一个后缀，存在两种可能性：

(1) P 是 S_i 的一个前缀；

(2) P 不是 S_i 的一个后缀.

如果 P 是 S_i 的一个前缀，LB 可能是 i 或者 i 的左边，同时 RB 也可能是 i 或者 i 的右边，因此搜索 S_i 的左边和右边都是有必要的.

如果 P 不是 S_i 的一个前缀并且 $P < S_i$，假使 P 存在于 S_i，LB 和 RB 都必定是在 S_i 的左边.

如果 P 不是 S_i 的一个前缀并且 $P > S_i$，假使 P 存在于 S_i，LB 和 RB 都必定是在 S_i 的右边.

如果 P 是 S_i 的一个前缀并且 P 不是 S_{i-1} 的前缀，那么 $LB = i$. 同样地，如果是 S_i 的一个前缀并且 P 不是 S_{i+1} 的前缀，那么 $RB = i$. 如果 $S = acggtacgtac$，对于 $P = c$，得到 $LB = 3$ 和 $RB = 5$. 假设 $P = tac$，$LB = 9$ 和 $RB = 10$. 通过二分搜索算法能够得到 LB 和 RB.

基于 CUDA，该算法被改进用于多模型序列并行匹配，该算法的伪代码描绘在算法 1 中. 其中，*thd* 表示当前线程的索引号，被存储在每个线程私有的寄存器中. 所有的模式序列被组织到一个字符串数组 *patstr* 当中，使得不同索引的线程 *thd* 能够映射到 *patstr*[*thd*] 中相应的位置. 本文中参考序列被构造成后缀数组主要是为了利用 GPU 改进二分搜索算法，大规模线程使得多模型序列能够并行的进行匹配. 参考系列的边界是由变量 *left* 和 *right* 来确定的.

算法 1　基于二叉树搜索的大规模基因片段并行匹配方法

```
1) //geneSubsequence[thd] is one of genome subsequences
2) thd = threadIdx.x+blockDim.x? blockIdx.x;
3) //assign the boundaries to registers
4) L = left, R = right
5) ___shared___ midSuffix[segLength], patString[segLength];
6) //an O(m log n) search algorithm to determine LB
7) while(R>L+1)
8) {
9) pivot =(L+R)>>1
10) do {
11) adjust middleSuffixLength and patStringLength
12) //update and extract trailing pair bases
13) //send extracted segment to shared memory
14) midSuffix = extract segment from S[SA[pivot]]
15) patString = extract segment from patStr[thd]
16) if (midSuffix ! = patString) break
17) while (middleSuffixLength > 0 && patString Length>0)
18) if (midSuffix ≤patString)
19) then R = pivot
20) else
21) then L = pivot
22) }
23) LB = R
24) L = left, R = right
25) //an O(m log n) search algorithm to determine RB which similar to LB
26) while (R>L+1)
27) {
28) pivot = (L+R)>>1
29) do {
```

30) adjust middleSuffixLength and patStringLength

31) //update and extract trailing pair bases

32) //send extracted segment to shared memory

33) midSuffix = extract segment from S[SA[pivot]]

34) patString = extract segment from patStr[thd]

35) if (midSuffix ! = patString) break

36) }while(middleSuffixLength> 0 && patStringLength> 0)

37) if (midSuffix< patString)

38) then R = pivot

39) else

40) then L = pivot

41) }

42) RB = L

43) __syncthreads()

44) res[thd<<1] = LB

45) res[thd<<1 + 1] = RB

设参考序列长度为 n，模型序列长度为 m，该算法将花费 $O(m \log n)$ 时间来确定模型序列左边界 LB 或右边界 RB. (LB,RB) 对应模型序列 $patstr[thd]$ 的匹配结果. 如果 (LB,RB) 为空，表示当前没有与此模型序列匹配的后缀. 在算法 1 中，当通过 (L,R) 远大于 1 得到 $pivot$ 时，本研究从全局内存中的模型序列 $patstr[thd]$ 和当前对应的比较序列 $S[SA[pivot]]$ 中提取部分序列片段到共享内存，直到 $middleSuffixLength \leqslant 0$ 或者 $patternStringLength \leqslant 0$

$S[SA[pivot]] \rightarrow midSuffix\ patstr[thd] \rightarrow patString$.

由于全局内存虽然容量大但是访问速度有限，然而共享内存具有容量小，访问速度快等特性. 本文所使用的方法正是一种通用的策略，将数据分化成子集合，通常称作瓷片，每个瓷片依次被传到共享内存中进行相关运算. 在大部分类型的并行计算机系统中，瓷片算法是一种并行优化的通用策略.

5 实验结果

实验环境：3.5 GHz 的 Intel Core i7 quad-core 处理器，包含 1563 个流处理器，6008 MHz，2GB 内存的 NIVIDIA GeForce GTX 680（最新的 Kepler 架构），1600 MHz、16 GB 的 DRAM.

本研究从 NCBI Nucleotide 中提取 DNA 来评估改进的算法的性能. 参考序列为编号 NT 167186.1Homo sapiens chromosome 1 genomic contig 中提取的前 10^7 个核苷酸，用来构建索引数据结构——后缀树和后缀数组. 实验中所用到的模型序列长度定位 1024 个核苷酸，其模型序列合集是从 NT 167186.1 Homo sapiens chromosome 1 genomic contig 和 the NT 039173.8 Mus musculus strain C57BL6J chromosome 1 genomic contig 提取的 DNA 序列. 实验中使用了包含不同数量的模型序列的集合，每个集合的模型序列数量范围在 512 到 2097152 之间.

为了通过比较分析来获得真实的性能，本研究实现了基于后缀树和后缀数组的多基因序列并行匹配. 在同构的多核 CPU 中，该算法利用 OpenMP 来支持 2,4 以及 8 线程并行处理，同时也与基于 CPU 实现的软件 MUMmer 进行了对比分析[4]. 实验的结果如图 4 的所示，虽然后缀树花费了 $O(m)$ 时间来进行搜索匹配，而后缀数组却花费了 $O(m \log n)$ 时间，由于后缀数组能够更加高效的使用缓存，其渐进的运行时间仅略高于后缀树. 实验中也表明了基于 GPU 实现的并行匹配速度要远超过 MUMmer.

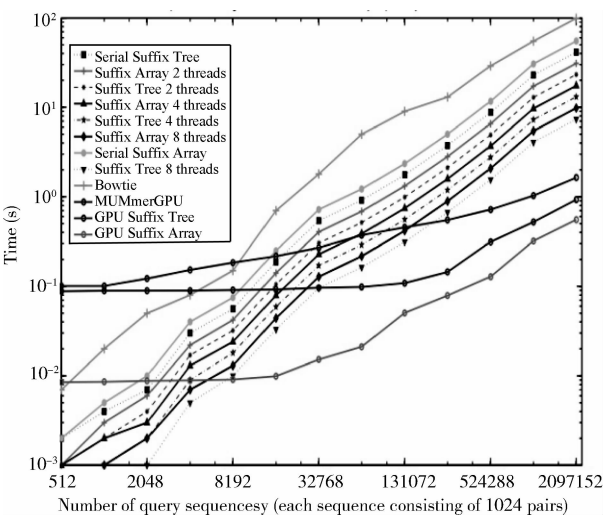

图 4 基于后缀树和后缀数组的匹配算法在 CPU 和 GPU 平台下的性能分析

Fig. 4 The performance analysis between CPU and GPU based on the suffix tree and suffix array

更进一步，进行评估 NIVIDIA GeForce GTX 680 实现的并进行匹配算法的性能. 实验结果描述如图 4 所示，结果显示并行匹配算法的时间是全部

的通信时间和核函数执行时间的总和. 数据首先要 CPU 从内存传送到 GPU 设备中，最后也会把得到的结果传回到 CPU 内存中，这就是所需的通信时间总和. 核函数执行时间是真正在进行并行搜索匹配过程的时间总和. 通信时间和内核执行时间的对比分析如图 5 所示.

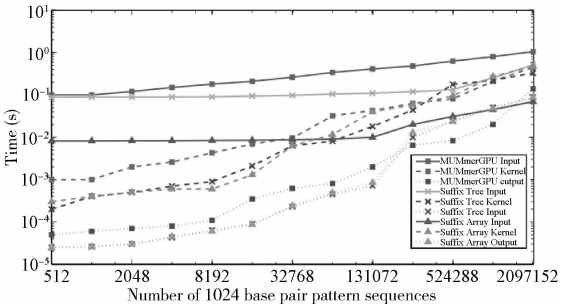

图 5　算法所需要内核执行时间和通信时间

Fig. 5　The kernel execution time and communication time of the algorithm

正如基于 CPU 的 MUMmer，MUMmer GPU 是完全基于 GPU 加速基因序列比对的工具[10]. 值得注意的是对比分析图中没有引入 CUDASW+，原因在于其参考序列做多容许 $64 * 10^3$ 核苷酸是最大的限制因素. 由于索引结构必须从 CPU DRAM 中转移到 GPU 设备上，输入时间对所有基于索引的搜索匹配算法都是至关重要的. 实际上，当进行匹配的模型序列数量很小时，数据的输入时间占据了绝大部分的时间总和. 然而当进行匹配的模型序列数量非常庞大时，基于 GPU 高并行度实现的并行匹配算法将会摊销输入时间的花费. 实验结果表明基于 GPU 实现的算法展现出惊人的性能提升，相对于 GPU 串行实现，基于后缀数组的并行匹配算法得到了将近 99 倍的性能提升，同时后缀树也获得了近 44 倍的加速比，详细的对比分析数据如图 6 所示.

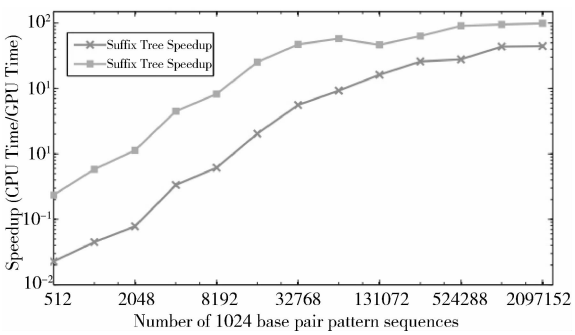

图 6　匹配算法在 CPU 与 GPU 之间实现的加速比

Fig. 6　The matching algorithm speedup between CPU and GPU

实验结果表明，基于 CPU 的后缀树匹配算法在执行时间上还略优于后缀数组，然而由于更贴近内存访问模型等优势，而通过 GPU 的实现更倾向于后缀数组. 另外，MUMmerGPU 搜索匹配多基因序列的运行时间明显高于后缀树和后缀数组. 因此. 在生物信息学应用中，基于 GPU 的后缀数组并行匹配算法是一种高效且实用的方法.

6　结　论

本文提出了后缀树与后缀数组实现并行匹配算法来加速 DNA 序列匹配过程的对比分析. 基于两种不同的并行平台：多核(i7 3770K quad－core)和众核(NVIDIA GeForce GTX680 GPU)，对这些索引结构进行全面的对比分析. 实验结果表明，在多核平台下，虽然基于后缀数组的渐进搜索所花费时间 $O(m\log n)$ 远高于后缀树的花费时间 $O(m)$，但是由于高效的缓存使用，是的后缀数组实际的匹配搜索仅仅是略高于后缀树. 然而，在众核平台下，凭借高效的缓存使用，更合适的执行流，以及空间站用量小等优势，基于后缀数组的并行匹配明显超过基于后缀树的算法. 同时后缀数组占用的空间也仅为后缀树的 20%～30%.

参考文献：

[1]　NVIDIA. CUDA compute unified device architecture programming guide[S/OL]. (2007-09-27). [2016-01-22]. http://docs. nvidia. com/cuda/cuda-c-programming-guide

[2]　Shendure J, Ji H. Next-generation dna sequencing [J]. Nat Biotechnol, 2008, 26(10): 1135.

[3]　Altschul S F, Gish W, Miller W, Myers E W, *et al*. Basic local alignment search tool[J]. J Mol Biol, 1990, 215(3): 403.

[4]　Treangen T J, Salzberg S L. Repetitive DNA and next-generation sequencing: computational challenges and solutions[J]. Nat Rev Genet, 2012, 13(1): 36.

[5]　孙晓楠. 基于 RBF 神经网络的 DNA 序列分类方法[D]. 吉林：吉林大学，2009.

[6]　郑俊生，张继红，马李昕. 基于 Gauss-EMD 的 DNA 序列相似性分析[J]. 大连交通大学学报，2013，34(1)：98.

[7]　李聪. 基于 OPENCL 的 DNA 序列并行比对算法的研究[D]. 哈尔滨：黑龙江大学，2015.

[8]　Gusfield D. Algorithms on strings, trees, and sequences [M]. Cambridge: Cambridge University

Press, 1997.

[9] Kurtz S, Phillippy A, Delcher A, *et al*. Versatile and open software for comparing large genomes[J]. Genome Biol, 2004, 5(2): 1.

[10] Schatz M, Trapnell C, Delcher A, *et al*. High-throughput sequence alignment using graphics processing units[J]. BMC Bioinform 2007, 8(1): 474 .

[11] Farach-Colton M, Ferragina P, Muthukrishnan S. On the sorting-complexity of suffix tree construction [J]. J ACM, 2000, 47(6): 987.

[12] Ukkonen E. On-line construction of suffix trees[J]. Algorithmica,1995, 14(3): 249.

[13] McCreight E M. A space-economical suffix tree construction algorithm[J]. J ACM,1976, 23(2): 262.

[14] Karkkainen J, Sanders P, Burkhardt S. Linear work suffix array construction [J]. J ACM, 2006, 53 (6): 918.

[15] Puglisi S J, Smyth W F, Turpin A H. A taxonomy of suffix array construction algorithms[J]. ACM Comput Surv, 2007, 39(2): 1.

[16] Hwu W, Keutzer K, Mattson T G. The concurrency challenge[J]. IEEE Des Test Comput, 2008, 25 (4): 312.

[17] Sanders J, Kandrot E. CUDA by example:an introduction to general-purpose GPU programming[M]. Boston, USA: Addison-Wesley Professional, 2010.

[18] Kirk D, Hwu W. Programming massively parallel processors-a hands-on approach [M]. California, USA: Morgan Kaufman Publishers, 2010.

[19] Gaster B, Howes L, Kaeli D, *et al*. Heterogeneous computing with OpenCL [M]. California, USA: Morgan Kaufmann Publishers Inc, 2012.

[20] Gregory K, Miller A. C++ AMP: accelerated massive parallelism with Microsoft R Visual C++[M]. Redmond, United States: Microsoft Press, 2012.

[21] Scarpino M. Open CL in action[M]. New York, USA: Manning Publications Company, 2014.

[22] Encarnaijao G, Sebastiao N, Roma N. Advantages and gpu implementation of high-performance indexed dna search based on suffix arrays[C]//Proceedings of the Ninth High Performance Computing and Simulation. New York, USA: IEEE, 2011.

[23] Cormen T H, Leiserson C E, Rivest R L, *et al*. Introduction to algorithms [M]. Massachusetts, USA: MIT Press, 2009.

[24] 刘森, 吴志红. 基于外存的场景加速数据结构研究[J]. 四川大学学报:自然科学版, 2016, 53 (2): 289.

[25] Sintorn E, Assarsson U. Fast pallallel gpu-sorting using a hybrid algorithm[J]. J Parall Distrib Comput, 2008, 68(10): 1381.