计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2014.20130987
51(9):1980-1992,2014

基于定制协处理器的基因重测序加速技术研究

汤 文[1,2] 张春明[1] 谭光明[1] 张佩珩[1] 孙凝晖[1]
[1](中国科学院计算技术研究所高性能计算机研究中心 北京 100190)
[2](中国科学院大学 北京 100049)
(tangwen@ncic.ac.cn)

A Customized Coprocessor Acceleration of Genome Re-Sequencing

Tang Wen[1,2], Zhang Chunming[1], Tan Guangming[1], Zhang Peiheng[1], and Sun Ninghui[1]
[1](*High Performance Computer Research Center*, *Institute of Computing Technology*, *Chinese Academy of Sciences*, *Beijing* 100190)
[2](*University of Chinese Academy of Sciences*, *Beijing* 100049)

Abstract Since January 2008 when the next-generation DNA sequencing platforms were developed, the sequencing throughput has been significantly improved. However, this technology has been challenged by the large amount of sequencing data which grows dramatically even over the Moore's Law. As an emerging data-intensive workload, the high-throughput re-sequencing tools like Hash-based programs shows different characteristics from traditional computational applications. Both low arithmetic intensity and irregular memory access pattern are major sources of inefficiency on commodity multi-core platforms. In this paper, we propose co-processor architecture for accelerating a short reads mapping algorithm. The complete mapping flow in one processing element (PE) is integrated to an exclusive memory port to improve the parallel performance. This proposed architecture is then implemented on a Convey HC-1ex reconfigurable computer. The design includes 64 parallel PEs on 4 Xilinx Virtex-6 LX760 that operate at 150 MHz. Compared with an Intel Xeon 8-cores CPU, the speedup achieves 28.5 times, and the average memory read bandwidth achieves 22.59 GBps. Therefore, this proposed design can potentially supply a solution to the large-amount data challenge and be applied in high throughput re-sequencing.

Key words high-throughput sequencing; short reads mapping; Hash-index; field programmable gate array (FPGA); heterogeneous architecture

摘 要 自2008年1月高通量测序技术应用以来,测序的通量和成本都在不断下降.然而基因数据的爆发式增长速度已经超过了摩尔定律,对海量数据的计算处理能力成为制约基因测序应用推广的瓶颈.以基于Hash索引的重测序算法为目标,对计算和访存行为进行分析,从而提出了一个现场可编程门阵列(field programmable gate array, FPGA)作为协处理器的架构,并在Convey公司的HC-1ex平台上进行了设计与实现.其基本处理单元内部采用全流水的设计及FIFO隔离计算模块和访存模块,可以完整执行重测序算法的核心流程.通过将基本处理单元和访存端口的一对一绑定,在4块Xilinx Virtex-6 LX760上实现了64路并行处理流程,总平均读内存带宽可达22.59 GBps.与8核Intel Xeon处理器相比,可以提升28.5倍的性能.

收稿日期:2013-07-09;**修回日期**:2013-08-15
基金项目:国家"九七三"重点基础研究发展计划基金项目(2012CB316502)

关键词　高通量测序技术;短序列比对;Hash 索引;现场可编程门阵列;异构体系结构

中图法分类号　TP302

基因测序作为获取基因序列信息的直接手段,是分析和破译遗传信息的基础,推动了医学、制药、生命科学等多方面的发展.随着人类基因组计划[1]的不断开展,基因测序技术也在不断革新.当前广泛应用的测序技术是高通量测序技术,又称"下一代"测序技术(next generation sequencing, NGS),主要特点是能一次性对大规模的 DNA 分子进行测序.相对于以 Sanger 测序为代表的第 1 代测序技术,NGS 测序得到的序列长度较短,一般为 30～200 个碱基对(base-pair, bp).NGS 应用进一步分为重测序(re-sequencing)和从头测序(de-novo sequencing)两种.从头测序主要针对未知物种的测序,通常通过构建 De-Bruijn 图[2]然后寻找欧拉路径的方法来获取一个全新的序列.而重测序则基于相同物种的不同个体的基因具有高度相似性的原理,以一个已知个体的基因序列作为参考模板,通过定位大量序列碎片并根据重叠部分的情况确定待测个体的序列.得益于人类基因组计划的完成,人类不同个体的基因测序都有参考模板,可以使用重测序这种更为高效和精准的测序手段.随着对基因密码的不断解析,基因测序逐渐会成为个体医疗中的重要手段,重测序有着更为广阔的应用前景.

测序方法的变革给基因序列处理算法带来了重要的变化.BLAST(basic local alignment search tool)方法[3]主要针对早期 Sanger 测序技术较长的查询序列(大于 1 000 bp),是基于 seed-extension 思想选取一定长度的序列片段作为 seed,并根据 seed 构建 Hash 索引.在进行查询时,首先利用 Hash 方法对 seed 进行定位,然后使用动态规划方法如 Smith-Waterman 算法[4]计算两个包含 seed 的扩展序列间的相似度.这种方法在面对 NGS 测序技术产生的海量短序列信息时,数据处理能力显得严重不足,即使利用 FPGA 对 Smith-Waterman 算法进行加速能获得约 3 个数量级的加速比[5-6],仍不能满足应用需求.为此,近年来出现了众多短序列比对工具,如 SOAP2[7], Bowtie[8], BWA[9], SHRiMP[10], PerM[11]等.相对于 BLAST,这些新的序列比对工具已经在很大程度上提高了基因数据处理的速度,但其数据处理能力与 NGS 测序数据产生能力之间的差距仍在不断拉大.

以 Illumina 公司的测序仪 HiSeq 2500[12]为例,完成一次测序过程约需 27 h,可产生 120 Gbp(giga base pair)的数据,即为每天 106 Gbp.而当前最快的处理速度仅为每天 5 Gbp,与测序仪的数据产生速度相差约 21 倍.此外,当测序仪获得的查询序列长度进一步增加时,序列比对的碱基对误差(mismatch)的容忍数量也需随之增加,其数据处理速度则会下降.以 PerM 为例,当误差的个数由 2 个增加到 3 个时,其处理速度会下降约 84%.若对 100 bp 长度的查询序列容忍 5 个误差,通用处理器和测序仪之间的差距将会拉大到 40～100 倍.

事实上,重测序算法中查询序列完全独立,具有极高的并行度,但通用的多核处理器结构并不能完全挖掘出其并行潜力.通过对典型测序算法如 PerM 的分析,发现这类算法具有低计算访存比和大量随机访存这两个限制多核并行效率的关键特征(参见 2.2 节).此外,还有一个重要特征是无浮点指令,导致在以追求浮点性能的通用处理器上执行存在极大的资源浪费.考虑到基因测序的广阔应用前景,本文研究采用高效的定制硬件如 FPGA 来实现对基因测序算法的加速.针对基于 Hash 索引的重测序算法,本文通过在 FPGA 上实现定制的处理单元和访存控制单元,获得了较好的加速效果.主要贡献如下:

1) 从体系结构的角度对基于 Hash 索引的短序列比对算法的计算行为特征进行了分析,主要从计算和访存两方面考察该应用对体系结构的特殊需求;

2) 提出了适合基于 Hash 索引的重测序应用的加速体系结构,包含 64 个数据处理单元和访存控制单元;

3) 对基于 Convey HC-1ex 的可重构计算平台进行了实现和验证.最终相对于 8 核 Xeon X7550 处理器具有 28.5 倍的平均性能提升.

1　重测序算法

短序列比对工具根据核心算法可以分为两类:基于 BWT(burrows-wheeler transform)算法和基于 Hash 索引算法[13-14].基于 BWT 算法的工具是根据基因序列字符种类少、重复次数多的特点,借用 BWT 的压缩技术将非精确匹配化简为更短的精确匹配,提高了序列比对的速度,同时降低了内存资源的消耗.在高通量测序技术出现早期,查询序列的长

度较短(小于 35 bp),基于 BWT 算法的工具在比对速度和内存开销方面都有较大优势.但随着测序技术的不断发展,数据通量越来越高,查询序列长度也在不断增加,BWT 算法的不足之处渐渐显现.在 BWT 算法采用了压缩方法降低复杂度的同时,为了容忍误差的存在,必须要采用回溯的方法.当查询序列长度增长时需要容忍的误差数量也会随之增加,BWT 算法需要回溯的开销也会不断增大.当序列长度突破临界值时回溯带来的开销会大于压缩所带来的好处.在可以预见的将来尤其是在单分子测序技术[15-16]普及之后,查询序列的长度会大幅增加,BWT 算法的适用性将受到限制.

基于 Hash 索引的算法继承了 BLAST 的思想,在选取 seed 和基于 seed 的扩展比对两方面针对短序列这一特点作了优化和改进,能够获得和 BWT 算法相当的性能.而在查询序列长度的可扩展性方面,Hash 索引算法要优于 BWT 算法.Hash 索引算法的主要缺点是其巨大的 Hash 表需要占用大量的内存空间.目前构建大容量内存系统的技术已经较为成熟,价格也呈下降趋势,制约 Hash 索引算法发展的硬件壁垒在逐渐消除,Hash 索引算法因其数据结构规整、计算流程简单、无需回溯等特点,应用会更加广泛.因此,本文将 Hash 索引算法作为主要研究对象.考虑到实际应用的广泛性,本文的工作都基于以人类全基因组为参考序列,其长度约为 3 Gbp.

下面介绍基于 Hash 索引的重测序算法流程,以 PerM 算法为例,量化分析其计算行为特征,从而设计合适的加速体系结构.

1.1 基于 Hash 索引的重测序算法

由于各个开发小组不同的应用背景,基于 Hash 索引的比对工具都有着各自的与生物学、统计学相关的复杂附属流程,但这些附属流程大多在预处理或后处理期间执行.占据绝大部分时间开销的是以 Hash 索引为中心的 seed 查询过程和 seed 扩展部分的比对过程.不同的比对工具中与 Hash 索引相关的流程大致相同,可以归纳为以下 3 个步骤:

1) 构建 Hash 索引.Hash 索引是一种常用的以空间换时间的提高性能的手段.在基于 Hash 索引的比对工具中,分为对参考序列建立索引和对查询序列建立索引两类.参考序列相对固定,查询序列则变化频繁,规模也在不断增长.因此,考虑到实际执行效率和扩展性,通常选择为参考序列构建 Hash 索引.

2) 计算 seed 的 Hash 值,从 Hash 索引中查询获取 seed 匹配的位置.通过不同的 seed 选取方式可以在 seed 内容忍一定数量的误差,如 PerM 中使用不同的循环空位 seed 可以在 seed 内容忍 2～4 个误差[9].由于基因序列的重复片段较多,数据量较大,通常采用两级索引的方式:第 1 级索引用于保存 Hash 值对应的 seed 片段在第 2 级索引中的位置;第 2 级索引用于保存在参考序列中的偏移量.

3) 获取 seed 扩展部分,计算查询序列与参考序列的相似度.由于应用需求的不同,对比对结果的要求各不相同.当前大多数工具只支持比对误差,对 2 个序列进行异或操作即可满足精度要求;少部分工具支持碱基对缺失(gap),需要使用 Smith-Waterman 算法进行比对.此外,还有对质量文件的支持,通过引入碱基对质量信息来提高比对精度.

1.2 计算行为分析

在上述 3 个步骤中,由于运行参数通常不会频繁变换,构建 Hash 索引可看作一次性操作,对性能的影响可以忽略,对计算行为的分析限定于对查询序列进行查询比对的过程,所消耗的时间约占完整执行流程耗时的 88.25%.我们使用 VTune[17] 对核心比对过程进行了分析,对 CPU 的行为特征进行了统计,主要包括 CPU 指令统计、CPI、Cache 缺失率等.基于 Hash 索引的短序列比对算法主要包括以下 3 个特征:

1) 低计算访存比.表 1 显示了核心比对过程中 CPU 的主要指令信息.其中,所有的与计算相关的指令,如整数运算、逻辑运算、移位运算等仅占指令总数的 26.34%,说明在 CPU 执行过程中计算能力并不是性能瓶颈.

2) 存在大量随机访存.表 2 显示了核心比对过程中各个主要步骤的 CPI 和 Cache 缺失率.在查询一级索引、二级索引以及获取参考序列对应片段时,仅需对访存获得的数据进行一次比较以确定是否需要进行下一步操作,计算量较小.同时由于 Hash 索引的离散性,使得对这两级索引和参考序列的访问都是随机访存,从而导致 CPI 和 Cache 缺失率都比较高,CPU 中的 Cache 几乎失去了作用.Cache 缺失较为严重的 3 个步骤占据了总消耗时间的约 73%,可见大量的随机访存是整个程序的性能瓶颈.

3) 无浮点操作.从表 1 还可以看出,在整个比对过程中没有调用任何浮点指令,计算操作只有仅含加减法的整数运算和各种位运算.这使得 CPU 中的浮点计算单元不能发挥其性能,浪费了 CPU 的计算能力.

Table 1　Statics of Instructions of Main Process

表 1　核心比对过程 CPU 指令统计

Type	Instruction	Instructions Retired	Percentage/%	
			Individual	Total
Arithmetic	add	31 124	16.22	17.17
	sub	1 820	0.95	
Logic	and	524	0.27	1.45
	or	2 212	1.15	
	xor	56	0.03	
Function	mov	116 776	60.87	71.88
	lea	12 320	6.42	
	cmp	3 896	2.03	
	test	4 914	2.56	
Shift	shl	2 890	1.51	7.72
	shr	11 282	5.88	
	sar	634	0.33	

Table 2　Last Level Cache Miss and CPI of Kernel Functions

表 2　核心比对过程 CPI 和 Cache 缺失率

Function	Time Percentage/%	CPI	LLC Miss
Get Hash Value	2.18	0.433	0.04
Query L1index	8.15	3.670	0.44
Query L2index	40.35	1.603	0.62
Get Subsequence from Reference	24.45	2.127	0.84
Get Similarity	13.12	0.498	0.00

由于短序列比对中所有查询序列都是完全独立的,具有非常优秀的并行潜力.因此,如果能使用定制部件获得高随机访存带宽,并集成简单的定点运算和位运算功能,将会大大提升短序列比对工具的性能.

2　协处理器开发平台

当前通用处理器的发展依然基本遵循着摩尔定律的轨迹,发展趋势已从提高主频转变到多处理核心并行的方向上.多年以来,内存技术的发展严重滞后于处理器的发展速度,长期积累形成了"内存墙",制约了处理器性能的发挥.由于通用处理器是以 cache line 为最小粒度进行访存的,而随机访存中需要的有效数据往往小于 cache line 的大小,使得原本并不充裕的有效访存带宽进一步下降,成为以随机访存为主要特征应用的首要性能瓶颈.

Convey 公司开发了一款以 FPGA 作为协处理器的异构平台,其结构如图 1 所示.与 Nallatech[18] 等基于通用处理器插座的协处理器架构不同,该系统构建了一个协处理子系统,通过前端总线(FSB)与主 CPU 进行通信.协处理子系统拥有自己的内存(co-processor memory, CP memory),与主 CPU 内存(host memory)共享地址空间,并维护 Cache 一致性.协处理器子系统主要由应用引擎(application engines, AEs)、应用引擎桥片(application engines hub, AEH)、内存控制器(memory controller, MC)3 部分组成.

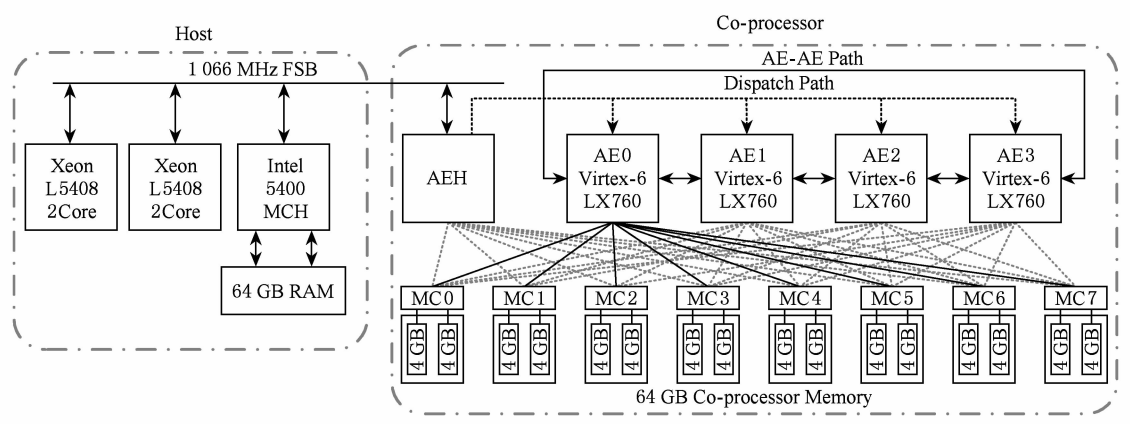


Fig. 1　Architecture of Convey system.

图 1　Convey 系统结构

Convey 为应用引擎提供了一个基础框架,包括指令接口、交互接口和访存接口.指令接口和交互接口都与 AEH 相连.指令接口负责自定义指令的解析与执行.交互接口负责数据交互,包括通过 AEH 的寄存器进行少量数据的交互以及通过内存操作(分配、复制、迁移等)的交互.访存接口与内存控制器相连,每个 AE 都可访问 8 个内存控制器,每个内存控制器都对应 AE 内的 2 个端口,即每个 AE 中

有 16 个访存端口.

如何利用 FPGA 高度可定制的特性以及 4AE 所提供的高访存带宽,是决定应用加速性能的关键因素,也是协处理器设计的难点所在,主要体现在以下 4 方面:

1) 软硬件划分. 整个短序列比对流程中既包含必须串行执行的部分,也包含适合并行加速的部分. 因此,需要对加速对象进行全面细致的分析,对计算任务进行恰当的软硬件划分,并尽可能减少由于软硬件划分所产生的软硬件之间的信息交互,以充分发挥各个计算部件的性能.

2) 数据结构设计. 由于 FPGA 灵活的可定制性,且 scatter-gather 内存支持最小粒度为 8 B 的访存,可以使用紧凑的数据结构达到很高的有效数据利用率. 但同时考虑到软硬件之间交互由于数据结构的转换可能带来额外的计算开销,需要采用折中的设计方案,使得在保证较高的数据利用率的同时尽可能避免额外开销.

3) 访存带宽的高效使用. 由于协处理器一端的访存是通过 AEs 与 MCs 之间的 32 条访存通道来实现的,为了有效利用这些访存通道所提供的高并发的随机访存带宽,必须仔细地设计访存控制细节,将访存请求尽量均匀地分布在 32 条访存通道上,以充分发挥其访存性能.

4) 访存延迟容忍. 由于每次访存请求都必须从 DRAM 中获取数据,数据返回的延迟相对较高(约 75 个周期). 为了保证流程执行的高效性,需要将流程中的计算和访存分离,并通过在计算模块和访存模块之间插入缓存的方式,组合成高效的流水线结构.

3 重测序加速处理设计与实现

3.1 软硬件划分

在同一批序列比对任务中,各项运行时参数如查询序列的长度、seed 的选取、Hash 函数的选取等不会发生变化,对参考序列构建的索引可以重复使用. 构建索引涉及到内存空间动态分配等操作不适宜使用协处理器进行,比对结果的归并处理以及输出到磁盘采用串行执行的方式更为合适,这两部分都由主 CPU 处理. 当查询序列过多时可分批进行,主 CPU 的预处理、后处理部分和协处理器计算可同时进行.

当索引全部读入到内存中后,需要对查询序列逐条进行查询序列 seed 生成、Hash 索引查询、位置索引查询以及序列比对 4 个步骤. 由于查询序列互相之间完全独立,具有非常良好的可并行性. 通过放置多个同构的处理单元,每个单元分别处理不同查询序列的完整流程,且相互之间无需进行通信,可以显著提升并行度并简化设计. 协处理器内的最大并行度即为处理单元的个数.

执行流程如算法 1 所示,主 CPU 首先读取索引和参考序列信息到 CP memory 中,并在 CP memory 中分配查询序列的空间和用于存储比对结果的空间. 然后,主 CPU 每次读取一批查询序列到 CP memory 中,发出调用协处理器的指令,协处理器开始并行处理查询序列. 各个处理单元以其序号为基础,总数量为步长,分别读取查询序列,继而进行索引比对流程,直至输出比对结果. 最后,主 CPU 根据返回值从 CP memory 中读取比对结果,进行归并处理,输出到磁盘,接着进行下一批查询序列的计算,直至任务全部结束.

算法 1. 主 CPU 和协处理器执行流程.

输入: *readlist*, a DNA READ list; *ref*, REFERENCE sequences; $\langle L1idx, L2idx \rangle$, Index table; *size*, size of READs buffer;

输出: Mapping result of all READs in *readlist*.

```
Host Thread:
Idxlist = Build-Hash-Index (ref) or Read-Hash-Index(L1idx, L2idx);
Copy-To-Coprocessor(Idxlist);
readsnum=number of total READs in readlist;
while readsnum>0 do
  readset=Get-Read-Buffer(readlist);
  readsnum=readsnum-size;
  ret=Call-Coprocessor(readset);
  if ret>0 then Output-Result();
endwhile
Co-processor:
foreach read in readset do
  (Hash,tag,key)=Get-Hash-Seed(read);
  (start, end) = Query-L1-Index (L1idx, Hash,tag);
  posset=Query-L2-Index(L2idx,key,start, end);
  foreach pos in posset do
    seq=Get-Substring(ref, pos);
    Pair-Wise-Comparison(seq,read);
  endfor
endfor
```

3.2　数据结构设计

3.2.1　基本数据结构

基本数据结构和 seed 选取方式与 PerM 保持一致. 对 ACGT 4 种碱基采用二进制编码的方式(A→00,C→01,G→10,T→11),每个碱基位用 2 b 表示,分为高、低两个 bit 位分别存储. 参考序列和查询序列信息都以两组 64 b 无符号整数分别存储高低位. 对参考序列的长度没有限制,取决于内存大小;查询序列最长支持 128 bp,seed 内容忍 2 个误差,整条序列容忍误差个数作为参数由用户运行时设定. 由于实验使用的真实测序数据长度为 100 bp,本文所涉及到的具体查询序列长度都以 100 bp 为例.

协处理器端的比对结果返回中包含查询序列标识、误差个数、匹配方向以及在参考序列中的匹配位置,用 4 个 32 b 无符号整数表示. 主处理器对返回的结果进行后处理,采用标准格式(如 sam 等)进行输出.

3.2.2　索引结构设计

考虑到基因序列中重复片段会较多,发生冲突的概率较大,采用两级 Hash 索引在保证精度的同时可以有效降低冲突率. 在第 2 级索引中,通过比较 Hash 地址延伸扩展部分的 *key* 值,可以进一步缩小范围,从而节省后续的进一步比对时间.

Hash 索引的结构设计需要考虑的主要因素是消耗的内存空间. 一方面,Hash 地址部分选取越长,Hash 表的空间开销越大,如在 100 bp 中选取 20 bp(40 b)作为 Hash 地址,需要 32 TB 的内存空间;另一方面由于在 seed 中容忍的误差数量有限,Hash 地址部分选取越长,所需的 seed 也越长,全序列的比对精度会有所降低. 因此需要选取合适长度的 Hash 地址,在保证一定比对精度的同时,减少内存空间需求,如 PerM 中默认使用 26 b 作为 Hash 地址的长度.

一种可以改进索引结构的方法是增加标识位. 假设选取长度为 A 的字段作为第 1 级 Hash 表的索引地址,则共有 2^A 项 Hash 表项,每项以长度为 32 b 的字段记录第 2 级索引的入口 *StartPos*,第 1 级 Hash 表大小为 $2^A\times 32$ b,若从 A 中取 T 作为标识位,则对每个 Hash 表项共有 2^T 种标识. 将不同标识位对应的第 2 级索引中〈*pos*,*key*〉二元组的个数 $Tag[i]$ 添加到第 1 级 Hash 表项中 *StartPos* 字段之后,同样可以通过 $StartPos+Tag[i]$ 确定第 2 级索引的入口,如图 2 所示. 由于同一个 Hash 桶内的元素个数有限,通过缩小 $Tag[i]$ 的位宽可以减小第 1 级 Hash 表的大小. 例如,若取 A 为 26,则第 1 级 Hash 表为 2 GB;若取 A 为 22,T 为 4,$Tag[i]$ 的位宽 t 为 14,则第 1 级 Hash 表大小为 $2^A\times(32+2^T\times t)=2^{22}\times(32+2^4\times 14)=1$ GB,相比于前一种格式的 Hash 表减少了一半的空间需求. 改进之后的第 1 级 Hash 表项长度为 256 b,由于 Hash 表访问的随机性以及 DRAM 的 burst 访问模式,相比于改进前的索引结构并不会带来访存性能的损失.

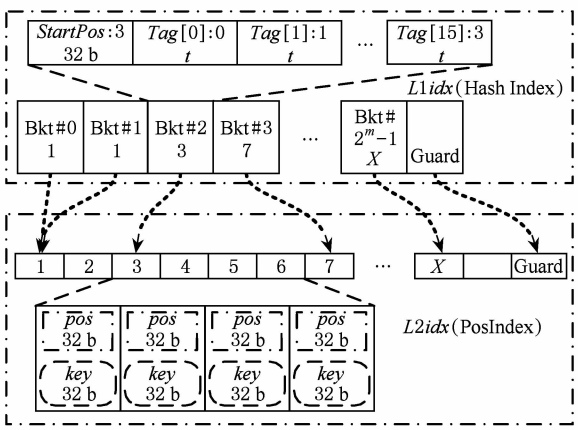


Fig. 2　Structure of 2 level Hash index.

图 2　Hash 索引结构

与传统的 DRAM 通过 burst 模式获得较高的理论吞吐率不同,Convey 平台采用了支持 scatter-gather 的内存条,可以在 64 b 的访存粒度下获得 80 GBps 的理论峰值带宽. 在 Convey 平台上,我们沿用了这种改进的索引结构,并根据平台的访存特性作了微调,从而提高了访存获得的有效数据的比率.

3.3　数据处理单元结构设计

为了充分利用协处理器的访存性能,需要在 AE 上放置多个并行的处理单元(processing element, PE),每个 PE 占用一个内存端口,能极大地提高访存的并发性. 同时,在 PE 内部集成一个内存控制模块(PE memory controller, PEMC)统一管理对不同数据结构的访存请求,如图 3 所示.

PE 主要负责从内存中获取查询序列信息,最终输出比对结果,采用全流水结构设计,整个处理过程可以分为 4 级:查询序列 seed 生成、Hash 索引查询、位置索引查询以及序列比对,每级之间使用 FIFO 隔离. 这 4 级都需要进行内存的读操作,仅序列比对部分需要进行比对结果的写操作. 读写操作请求均通过 PEMC 进行仲裁,然后发送到访存端口. 每次读写请求的数据位宽均为 64 b.

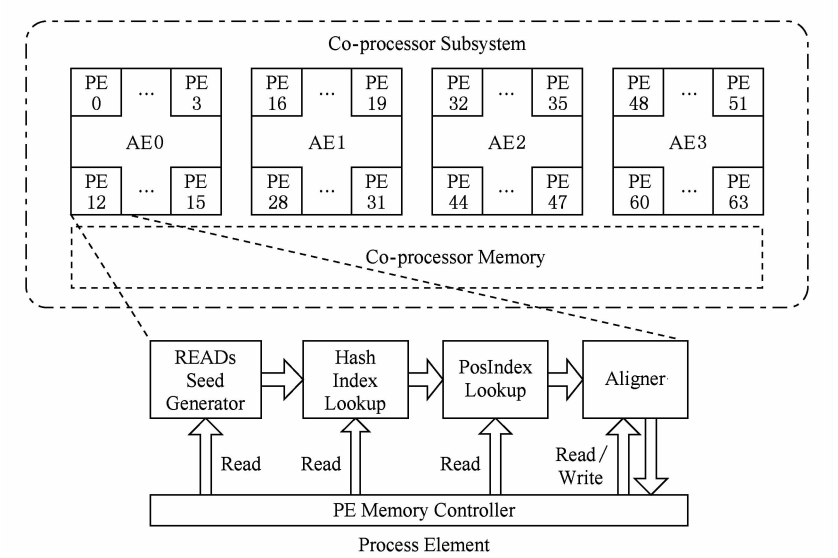


Fig. 3 Design of co-processor.

图 3 协处理器结构设计

3.3.1 查询序列 seed 生成

查询序列 seed 生成包括查询序列获取模块、逆序模块、移位模块、Hash 值计算 4 个子模块，如图 4 所示. 查询序列获取模块用于发起访存请求读取查询序列信息，添加查询序列标识写入查询序列 FIFO 供序列比对使用，添加辅助控制信息字段发送给逆序模块. 辅助控制信息字段包括查询序列结束标识、查询序列边界标识、逆序标识、移位计数. 由于查询序列的方向不确定，在正反两个方向上都有可能匹配，故需要逆序模块是进行逆序处理，并将逆序标识置为 1. 移位模块则是从查询序列的起始位置开始，逐位生成 seed，并设置移位计数信息.

Hash 模块如图 5 所示，采用一个 6 级流水线结构，每个周期能给出一个 Hash 函数的计算结果.

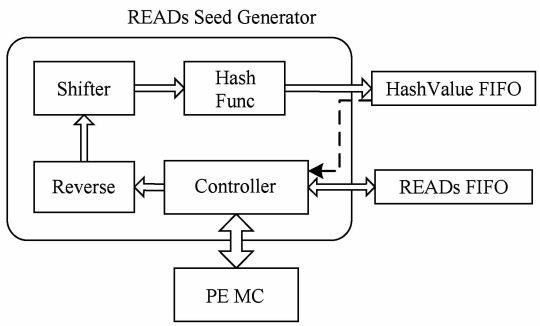


Fig. 4 The READs seed generator.

图 4 查询序列 seed 生成模块

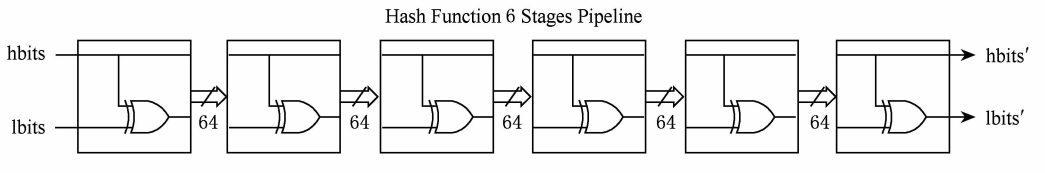


Fig. 5 Hash function pipeline.

图 5 Hash 函数及 6 级流水线

Hash 函数如算法 2 所示. Hash 值连同辅助信息字段写入 HashValue FIFO，供下一步查询 Hash 索引使用.

算法 2. Hash 函数.

输入：v, value before Hash; $lmask$, lower half bits mask; $hmask$, higher half bits mask;

输出：hv, value after Hash.

$p[0]=v$;

for i from 1 to 6 do

$h[i]=p[i-1]$ AND $hmask$;

$l[i]=p[i-1]$ AND $lmask$;

$p[i]=h[i]$ OR ($v[i]$ XOR $l[i]$);

endfor

$hv=p[6]$;

3.3.2 Hash 索引查询

Hash 索引查询模块如图 6 所示，包括 Hash 索引访问模块和重组模块两个子模块以及一个 HashKey FIFO. Hash 索引访问模块根据上一个模块计算获

得的 Hash 值的生成查询地址检索 Hash 索引,将剩余部分作为 *hashKey* 连同辅助控制信息写入 HashKey FIFO. 访存请求返回一个二级位置索引中的偏移量 *posOffset* 和冲突表项个数 *count*,交由重组模块与对应的 *hashKey* 以及辅助控制信息进行重组,然后写入 PosPointer FIFO 中,供检索 2 级位置索引使用.

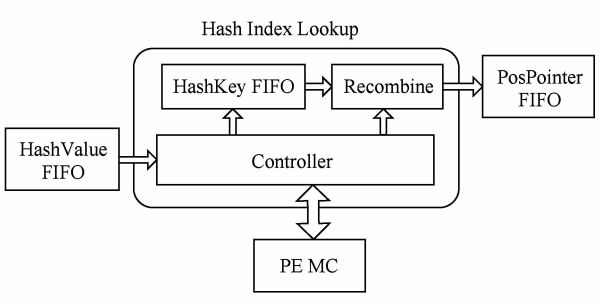


Fig. 6　Hash index lookup module.

图 6　Hash 索引查询模块

3.3.3　位置索引查询

位置索引查询模块如图 7 所示,包括位置索引访问和 *Key* 比较两个子模块,以及一个 PosKey FIFO. 位置索引访问模块从位置偏移 FIFO 中获取 *posOffset*,*count*,*hashKey* 以及辅助控制信息,后三者作为一组写入 PosKey FIFO 中暂存,同时根据 *posOffset* 生成访存地址,从位置索引中获取数量为 *count* 的 64 b 二进制数,该 64 b 二进制数由参考序列位置 *pos* 和 *key* 组成. 比较模块从 PosKey FIFO 中取出一组信息,对访存返回值的个数进行计数,达到 *count* 个则取下一组信息. 同时将访存返回的 *key* 与 *hashKey* 进行比较,若相同则把参考序列位置写入 Position FIFO 中,供序列比对使用.

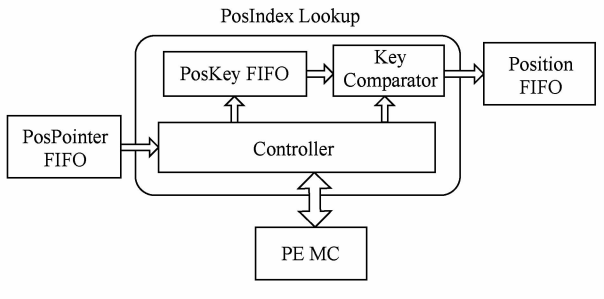


Fig. 7　PosIndex lookup module.

图 7　位置索引查询模块

3.3.4　序列比对

序列比对模块如图 8 所示,包含参考序列访问、比较器、结果输出 3 个子模块和 PosInfo FIFO、比对结果 FIFO. 参考序列访问模块根据 Position FIFO 中的参考序列位置信息生成访存地址,获取对应位置与查询序列相同长度的序列进行比对.

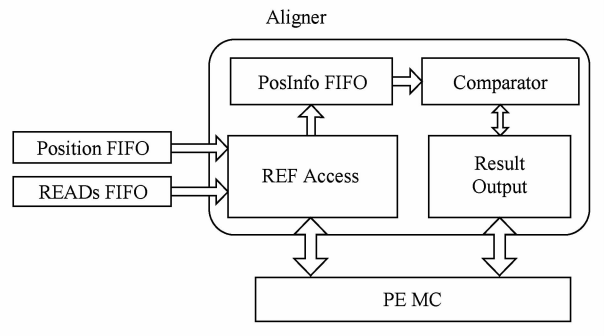


Fig. 8　Aligner.

图 8　序列比对模块

为使每个时钟周期内能完成一次序列比对操作,比较器实现为一个 3 级流水线结构,如图 9 所示:

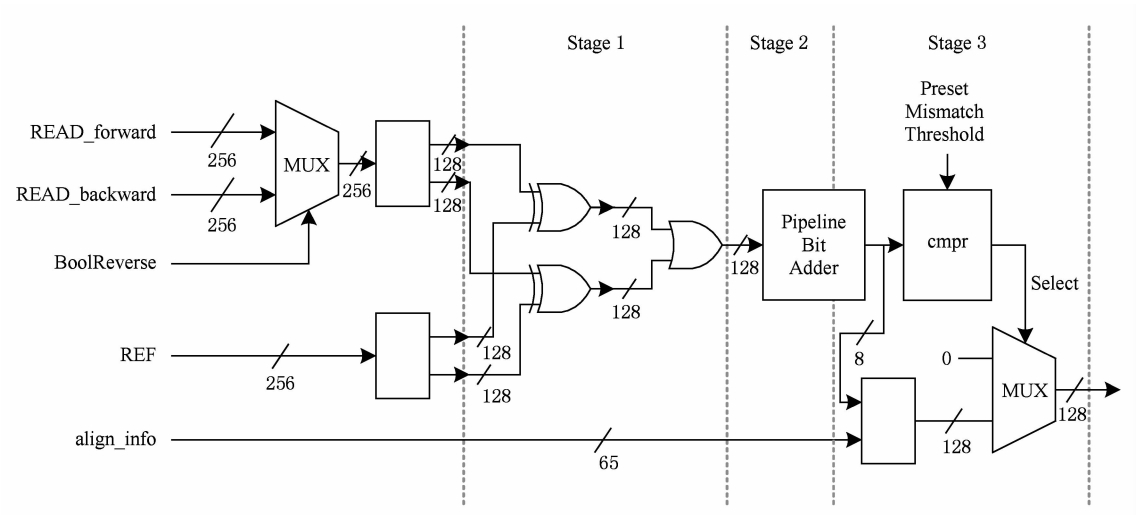


Fig. 9　Pipeline of comparator.

图 9　比较器 3 级流水线结构

第1级流水线通过先对高低两组数据作异或运算,然后对结果作或运算来实现序列的比对.后2级流水线用于统计比对结果中"1"的出现次数(即mismatch出现的次数).当"1"的出现次数小于阈值时,表示输入序列对在允许的误差范围内匹配成功,将比对结果写入结果FIFO中.虽然目前的比较器实现只考虑字符误差,但可以通过用动态规划算法模块替换异或模块来实现各种复杂序列匹配功能扩展.结果输出模块从结果FIFO中取出比对结果,生成写地址,将写内存请求发送给PEMC.

3.4 PE访存控制器结构设计

由于DRAM返回数据需要的周期数具有不确定性,为了保证访存通道的高效使用,需要采取一定的缓存机制将处理单元和访存单元隔离.读写请求的地址和数据都通过FIFO与数据处理单元交互.仲裁模块在发送访存请求的同时,根据来自请求的具体处理模块不同,添加对应的访存标识.访存端口在返回请求数据时会连同访存标识一起返回.读写控制模块根据返回的不同标识,将数据分发至不同的FIFO中,供数据处理模块读取.PEMC结构如图10所示:

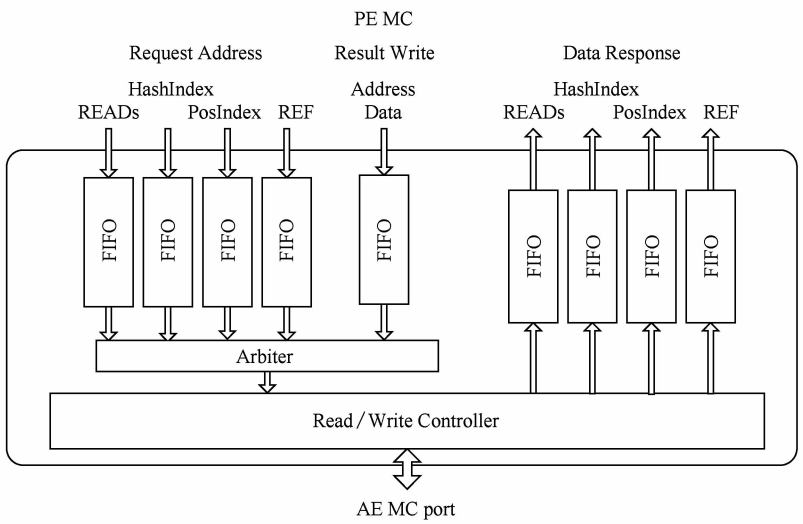


Fig. 10 Memory controller of PE.

图10 PEMC模块

由于一个时钟周期内一个访存端口只能接收一个读或写请求,而4个计算模块中共有4种不同数据类型的读请求和一种写请求,因此需要使用一定的仲裁策略将数据处理单元的读写请求映射到访存端口上.考虑到5种访存请求量较大,且具体分布与实际运行时的数据特征相关,故采用可配置的时间片轮转的仲裁策略.仲裁策略如算法3所示,根据预先设定的时间片分配比例(目前为1:3:3:3:2,可根据实际需求设置),轮流将5种访存信号发送到访存端口.若当前时间片对应的读数据类型没有访存请求,则将该时间片分配给下一类型的当前存在的读请求.

算法3. PEMC仲裁策略算法描述.

输入: *read_rd*, *ref_rd*, *Hashidx_rd*, *posidx_rd*, *result_wr*; /* memory access request from top modules */

输出: *ld_req*, *st_req*. /* memory request send to MC */

```
while 1 do
  switch time_slice do
    case Hit-Read_Rd
        ld_req=read_rd;
        st_req=NULL;
    case Hit-Hash-Rd
        ld_req=Hashidx_rd;
        st_req=NULL;
    case Hit-Pos-Rd
        ld_req=posidx_rd;
        st_req=NULL;
    case Hit-Aligner-Rd
        ld_req=ref_rd;
        st_req=NULL;
    case Hit-Result-Wr
        ld_req=NULL;
        st_req=result_wr;
    otherwise
        ld_req=Next-Non-Null-Requset;
        st_req=NULL;
  endswitch
endwhile
```

4　实验与性能分析

4.1　实验环境与测试用数据

本文实现的系统设计是基于 Convey 的 HC-1ex 平台.该平台共有 128 GB 内存,主 CPU 和协处理系统各占一半.主 CPU 为 2 个双核 Xeon L5408 处理器,协处理系统包含 4 块 Xilinx Virtex-6 LX760 组成的 AEs,2 块 Xilinx Virtex-5 LX110 组成的 AEH,以及 8 块 Xilinx Virtex-5 LX155 独立构成的 8 个内存控制器.协处理系统通过 FSB 总线与主 CPU 通信.

在并行性测试方面,协处理器将查询序列的数量均匀分配给所有 PE,当所有 PE 完成计算任务后才结束调用.对于纯 CPU 版本的软件,采用了一款 64 核 SMP 系统进行扩展性分析.该系统包含 8 个 Intel Xeon X7550 处理器.每个处理器包含 8 个工作频率为 2 GHz 的处理核心,共享 18 MB 三级缓存.8 个处理器分为两组,组内 4 个处理器通过 QPI (QuickPath Interconnect)全直连,组间的处理器通信则需要一次 QPI 中继.并行算法通过 Open MP 实现并行,任务划分也采用平均分配查询序列的方法.

考虑到实际应用的广泛性,实验数据使用 Sanger 实验室公布的人类全基因组数据作为参考序列,长度约为 3 Gbp.待查询序列来自于 Novogene 公司①,由 Illumina 公司的 HiSeq 2000 测序仪所产生的长度为 100 bp 的短序列.在整个处理流程中,查询序列的加载和结果的输出处理由主 CPU 完成,可以和协处理器的计算时间相互重叠.此外,由于 Hash 索引的构建是一次性的,在批量处理查询序列时,Hash 索引仅需载入一次,Hash 索引的建立和加载时间不计算在内.因此,以下的时间统计仅包含从主 CPU 调用协处理器开始,到协处理器完成计算任务发出返回信号被主 CPU 检测到为止.同时,对纯 CPU 版本的软件,也仅统计实际比对计算消耗的时间,不包含数据的预处理和结果的输出的时间.

4.2　硬件资源开销

以 Xilinx ISE 12.4 工具经布局布线步骤后生成的数据为依据,表 3 列出了单个 PE 和一个 AE (16 个 PE)的资源开销、资源开销百分比和综合频率.其中 FPGA 片内存储资源(BRAMs)主要用于实现 PE 和 PEMC 中的 FIFO 以及访存端口模块中的缓存.

Table 3　FPGA Resources Consumption

表 3　系统实现的资源开销和综合频率

Module	Slice LUTs	36 Kb BRAMs	Frequency/MHz
Single PE	75 190(15%)	69(9%)	150
Single AE(16PE)	353 000(74%)	115(15%)	150

4.3　加速效果及访存带宽

图 11 表明了在异构平台上的性能提升与使用 PE 个数的关系.加速比计算以 CPU 单线程的计算时间为基准.从图 11 可以看出,当 PE 使用数量增多时,查询序列的条数在达到一定规模后加速比才会趋于稳定.这是由于 PE 完成计算任务的时间与其所分配到的实际查询序列的数据有一定的相关性.有的查询序列可匹配的位置比较多,消耗的时间也会增加,反之则消耗的时间比较少.从图 11 还可看出,PE 个数相同时,查询序列的数量变化对性能的影响不大.当每个 PE 分配到的查询序列数据规模到达一定程度的情况下,性能会趋于稳定.当数据充足时,各个规模情况下的单个 PE 的加速比都稳定在 3.5 以上,体现了优秀的可扩展性.在比对精度方面,与纯 CPU 的串行软件版本保持一致.

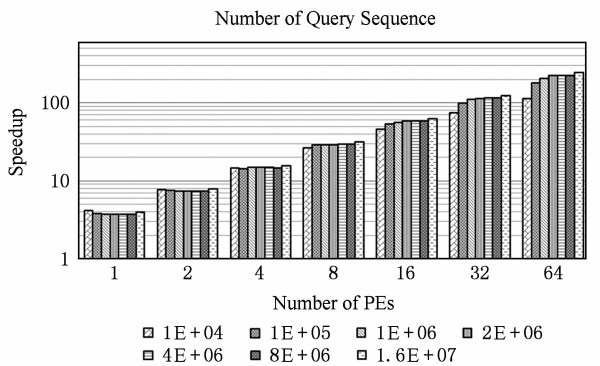


Fig. 11　Performance improvement of co-processor base on 1 CPU core.

图 11　使用协处理器异构结构的性能提升

图 12 将 PE 的可扩展性与 CPU 的可扩展性进行了对比.加速比都是以单 PE 或单个 Open MP 线程作为基准.由于短序列比对应用本身具有的潜在的高并发性,在当前的硬件平台上两种并行方式都具有较好的可扩展性.在纯 CPU 平台上,加速效果在并行低于 16 路的情况下趋近于线性,高于 16 路

① http://www.novogene.cn/

之后逐渐下降,在 64 路并行时仅有 32.7 倍的加速效果.在异构平台上,加速效果始终接近线性,在 64 路并行时仍有高达 60.02 倍的平均加速效果.考虑到 16 M 条查询序列分配到 64 个 PE 上,每个 PE 上仅有 25 万条查询序列,PE 执行任务的结束时间相差仍较大,其并行加速效果仍有提升空间.

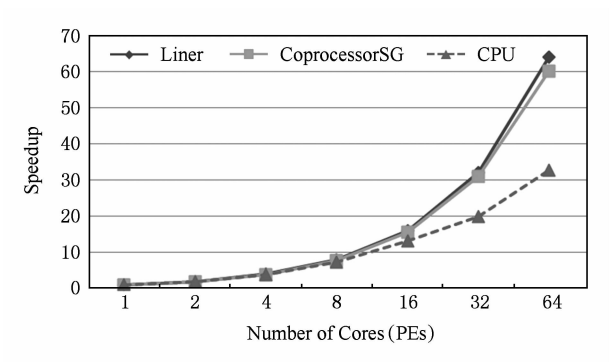


Fig. 12 Comparison of scalability.

图 12 可扩展性对比

异构平台的之所以能取得较好的加速效果,得益于其所支持的 scatter-gather 内存和高并发访存带宽.由于实际执行中写内存的数据量不到读的 1/1 000,在带宽测试和分析中仅考虑读内存带宽.图 13 展示了 AEs 在不同情况下的读内存带宽.由于 FPGA 的高度可定制性以及细粒度访存的支持,使得实际访存所获得的数据具有极高的利用率,这一特性是通用处理器平台所不具备的.由于序列比对应用的访存行为与数据较为相关,为了对比参考序列的不同对访存性能的影响,我们测试了单 PE 以包含人类 1～22 号染色体、X 染色体、Y 染色体共 24 条不同染色体作为参考序列的读内存带宽,如图 14 所示.当参考序列规模较小时,带宽的波动较大,规模超过 80 MB 后趋于平稳.耗时总体随着参考序列增大或增加,同时也与查询序列与参考序列的数据特征相关.不将染色体分开直接使用全基因组作为参考序列时的读内存带宽约 305 MBps,与单染色体相当.

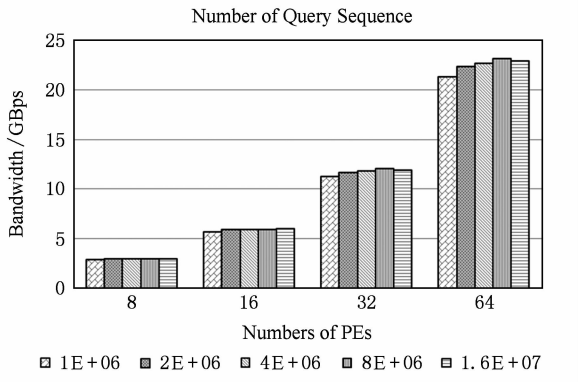


Fig. 13 Bandwidth of co-processor.

图 13 协处理器有效读数据带宽

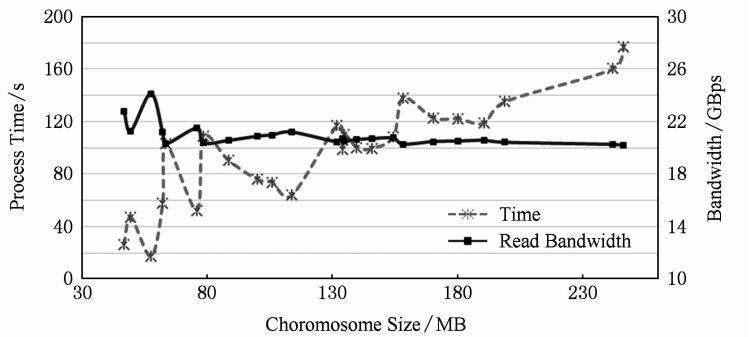


Fig. 14 Bandwidth of single PE for different chromosome.

图 14 不同规模的染色体作为参考序列时读数据带宽

4.4 性能分析

在基于 Hash 索引的重测序算法中,读内存的数据总量远超写的数量,决定性能的主要因素是内存系统的随机读带宽.注意到在 64PE 的情况下的有效数据读带宽为 22.59 GBps,仅占理论峰值带宽 80 GBps 的 28.88%.由于计算模块都采用流水线结构,只要数据供给不出现停顿,计算流水线不会成为瓶颈.可能影响访存性能的因素有:4 个计算模块间的 FIFO 深度、PEMC 内读地址和数据返回 FIFO 深度以及读请求的发起频率.

由于片上资源受限,我们使用相同的数据对计算模块间的 FIFO 深度分别为 128,256 和 512,以及 PEMC 内 FIFO 深度分别为 256,512 和 1 024 的单 PE 读数据带宽进行了测试,如表 4 所示.在各种情况下的带宽变换较小,可见系统性能与处理单元内的缓存大小关系不大.

在查询序列、Hash 索引、位置索引、参考序列 4 种读请求中,每一条查询序列会产生 4 次读请求,继而会产生 28 个 Hash 值,对应了 56 次 Hash 索引读请求;每个从位置索引返回的参考序列偏移量平均

Table 4　Read Bandwidth of Single PE with Different FIFO Depth

表 4　单 PE 不同 FIFO 深度的读数据带宽　MBps

Type	PEMC_FIFO-256	PEMC_FIFO-512	PEMC_FIFO-1024
PE_FIFO-128	377.95	379.23	380.12
PE_FIFO-256	376.13	377.91	379.04
PE_FIFO-512	378.34	380.91	380.03

对应了 6 次参考序列的读请求. Hash 索引读请求数量和位置索引读请求数量的比例关系与实际的数据相关,通过对 1 600 万条查询序列的统计约为4∶7. 在处理单元中,由于边界条件的差异,4 种读请求的产生状态机各不相同,若对应的 FIFO 读写都无需等待即为理想状态. 实际的读请求比例和理想状态下发出读请求的周期数占整个状态机周期数的比例如表 5 所示,其中位置索引的 c 为当前扩展 key 值匹配的个数,与实际数据相关,在当前数据集的情况下平均为 1.75.

Table 5　Statistics of Read Request

表 5　读请求次数统计

Type	READs	Hash Index	PosIndex	REF
Actual Access	4	56	98	600
Read Cycle in State Machine	1/1	2/4	$c/(c+3)$	6/11
Theoretical Cycle	4	112	266	1100

根据表 5 数据可以看出,由于 4 个模块并行执行,在理想状态下,平均一条查询序列需要发出 758 个读请求,占用 1 100 个周期,占空比约为 68.9%. 而在实际执行中,由于数据特征的不均匀性,某些查询序列位置索引和参考序列的访问数量会远高于其他序列,从而会导致数据流的拥塞,且后两种读请求的占空比较低,对整体性能影响较大. 一种解决方案是根据读请求的比例不同,实现不同数量的功能模块,增加单位周期内的访存请求次数,以提高 PE 对访存端口的使用效率. 如使用 1 个查询序列 seed 生成模块、2 个 Hash 索引查询模块、3 个位置索引查询模块以及 2 个序列比对模块,以提高读请求的发起频率,这将是我们的未来工作之一.

5　结　　论

通过对当前短序列比对算法的细致分析,同时考虑到未来的发展趋势,本文选择了以 Hash 索引为基础的短序列比对算法作为研究对象,并分析了此类应用在体系结构方面的需求. 由于以 Hash 索引为核心的应用包含大量的随机内存访问,计算操作主要为字符串比对操作,访存需求较高而计算强度较低. 传统的通用处理器平台在以随机访存为瓶颈的应用上效率较低. 本文在 Convey 的 HC-1ex 异构平台上实现了 64 路并行处理单元的体系结构,读内存带宽平均可达 22.59 GBps,平均计算性能相对于 8 核 Xeon X7550 处理器具有 28.5 倍的提升.

随着测序技术的进一步发展,以高通量重测序应用为代表,基因数据的处理与分析将成为计算需求的新兴增长点. 本文的主要贡献在于从体系结构的角度探寻了这一类应用的特殊需求,并通过对协处理器体系结构的设计与实现进一步验证了其可行性. 此外,这种异构的体系结构对于以随机访存为瓶颈的算法具有较强的适用性. 我们的后续工作是通过提取 Hash 操作的共性,设计并实现协处理器的体系结构,使之能应用于以 Hash 索引为核心的其他应用.

参　考　文　献

[1] National Human Genome Research Institute. All about the human genome project [OL]. [2010-12-20]. http://www.genome.gov/10001772

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

[3] Kent J W. BLAT—The BLAST-like alignment tool [J]. Genome Research, 2002, 12(4): 656-664

[4] Smith T F, Waterman M S. Identification of common molecular subsequences [J]. Journal of Molecular Biology, 1981, 147(1): 195-197

[5] Zhang Peiheng, Liu Xinchun, Jiang Xianyang, et al. Animplementation of reconfigurable computing accelerator card oriented bioinformatics [J]. Journal of Computer Research and Development, 2005, 42(6): 930-937 (in Chinese)
(张佩珩, 刘新春, 江先阳, 等. 一种面向生物信息学的可重构加速卡的设计与实现[J]. 计算机研究与发展, 2005, 42(6): 930-937)

[6] Zhang Yang, Dou Yong, Xia Fei, et al. Design and implementation of bioinformatics pare-wise alignment accelerator [J]. Journal of Frontiers of Computer Science & Technology, 2008, 2(5): 519-528 (in Chinese)

(张阳, 窦勇, 夏飞, 等. 生物信息学双序列比对算法加速器设计与实现[J]. 计算机科学与探索, 2008, 2(5): 519-528)

[7] Li Ruiqiang, Yu Chang, Li Yingrui, et al. SOAP2: An improved ultrafast tool for short read alignment [J]. Bioinformatics, 2009, 25(15): 1966-1967

[8] 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): 1-10

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

[10] Rumble SM, Lacroute P, Dalca AV, et al. SHRiMP: Accurate mapping of short color-space reads [J]. PLOS: Computational Biology, 2009, 5(5): 1-11

[11] Chen Y, Souaiaia T, Chen T. PerM: Efficient mapping of short sequencing reads with periodic full sensitive spaced seeds [J]. Bioinformatics, 2009, 25(19): 2514-2521

[12] Illumina. HiSeq 2500 sequencing system [OL]. [2012-04-25]. http://www.illumina.com/

[13] Li H, Homer N. A survey of sequence alignment algorithms for next generation sequencing [J]. Briefings in Bioinformatics, 2010, 11(5): 473-483

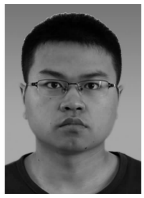
[14] Bao S, Jiang R, Kwan W, et al. Evaluation of next-generation sequencing software in mapping and assembly [J]. Journal of Human Genet, 2011, 56(6): 406-414

[15] Helicos Heliscope. HeliScope single molecule sequencer [OL]. [2011-03-03]. http://www.helicosbio.com

[16] Pacific Biosciences. SMRT technology [OL]. [2011-03-15]. http://www.pacificbiosciences.com/products/smrt-technology/

[17] Intel. Intel® VTune™ Amplifier XE 2013 [OL]. [2013-01-11]. http://software.intel.com/en-us/intel-vtune-amplifier-xe/

[18] Nallatech. Intel Xeon FSB FPGA accelerator module [OL]. [2010-07-05]. http://www.nallatech.com/Intel-Xeon-FSB-Socket-Fillers/fsb-development-systems.html



Tang Wen, born in 1987. PhD. Received his BSc degree from the University of Science and Technology of China. His research interests include parallel algorithm, reconfigurable computing and computer architecture.

Zhang Chunming, born in 1980. Associate engineer. Received his BSc degree from Civil Aviation University of China, and master degree from Tsinghua University. His main research interests include reconfigurable computing and ASIC.

Tan Guangming, born in 1980. Associate professor in the High Performance Computer Center, Institute of Computing Technology, Chinese Academy of Sciences. His main research interests include parallel algorithm and programming, performance modeling and evaluation, and computer architecture.

Zhang Peiheng, born in 1968. Professor in the High Performance Computer Center, Institute of Computing Technology, Chinese Academy of Sciences. Senior member of China Computer Federation. His main research interests include computer architecture, hardware design and reconfigurable computing.

Sun Ninghui, born in 1968. Professor and PhD supervisor. Senior member of China Computer Federation. His main research interests include high performance computer systems.