基于多核和众核架构的基因组读取映射的数据并行化

赵东杰 P18206023 13121595665 zdj8023first@163.com

摘要：

目前，在解决基因组研究中蓬勃发展的数据的解决方案，由多核和多核系统组成的异构架构已经变得更加强大。本文主要介绍了基因组研究中一样的异构架构的使用。作者对众核组件的使用，比如说 Xeon Phi 加速器进行了特别的研究，该加速器可以保证为 x-86 架构的多核服务器开发的应用程序的迁移。序列对比问题是大多数基因组变异研究中基本且成本最高的计算阶段之一，本文也对该问题进行了研究。作者对 BWA，目前最流行的序列对比器之一，进行了重点的研究，同时对三种不同的异构系统进行了研究介绍，第一种是包含 Intel 多核 CPU 和加速器，第二种由多个多核服务器组成，第三种是大规模系统。每种系统在 CPU 数量，内核数量和系统组织内存方面具有不同的特性。虽然说序列对比问题适用于并行模式，但是在异构环境中实现良好的性能和良好的可扩展性可能是复杂的。通过对数据分布以及部分数据结构复制进行不同的策略分析，作者提出了 MDPR（多级数据并行化及复制），并且在前面三种结构的系统中都显示出了最好的测试结果。测试结果比文献中其他的结果都要高效，并且适用于不同的异构系统，而不用根据不同的底层架构进行特定的重构。在 MDPR 的设计中，作者还评估了不同的静态和动态策略，其中静态策略结果最好，但是有着显著的预处理的成本。而基于循环机制的数据分布的动态策略，可以获得近似的处理时间，并且没有预处理的成本。作者提出的策略是基于人类基因组数据以及 BWA 的，但是可以很容易地应用到其他类似的序列数据集以及对比工具中。

1. 介绍

近年来，由不同架构组成的异构系统已成为高性能计算领域的一个重要趋势。尽管诸如 OpenCL 和 CUDA 之类的特定编程语言需要大量的编码技能，但是诸如 GPU 之类的众核架构已经获得了相当大的普及。Intel Xeon Phi 是另一个多核协处理器，它已成为多核系统上现有程序的一个有吸引力的解决方案，可以轻松移植，因为 Xeon Phi 也基于×86 架构。

与计算进化（每 18 个月晶体管数量翻倍）相比，序列基因组数据量每 12 个月翻一番。然而，这是一个惊人的数据增长，构成了一个重要的挑战，仍然需要有效的应用程序来处理它们。

序列比对，变体调用和变体注释是基因组数据研究中的三个基本操作。序列比对是一个关键步骤，可以为其他两个操作提供主要后果。它涉及尽可能准确地将短读取映射到基因组序列。尽管近年来已经开发了许多对准器，例如 MAQ，SOAP 和 BLAST，但它们都表现出大量的执行时间和大的内存占用。 虽然它的执行时间与其他现有的对齐器类似，但 BWA（Burrows-Wheeler Aligner）[7]利用 BWT 索引技术降低了对齐过程的内存需求。 这种记忆需求的减少加上其映射精度使 BWA 成为科学界最受欢迎的对齐器之一。

在本文中，探讨了允许 BWA 使用的不同策略运行在将多核 CPU 与多核加速器或不同多核系统相结合的异构系统上运行。作者分析了文献中的一些提议策略，并提出了改进全球绩效的改进和新策略。本文扩展了之前的研究，其中作者提出了一些数据并行/复制策略，并在基于多核的系统中对基于 BWA 的对齐器进行了评估。 在这些工作中，作者的研究只关注包含加速器的异构系统。在一个案例中，我们分析了在本机模式下执行应用程序时的改进[8]，在[9]中，我们研究了在对称模式下执行 BWA 的不同变体。本文分析的 MDPR 策略源于这些以前的工作，并且已经被推广和扩展，因此它不仅可以在基于加速器的服务器中模糊地执行，而且可以在由围绕不同 NUMA 体系结构组织的多个 CPU 形成的其他类型的异构环境中执行。策略 MDPR 利用多个实例，以通过减少内存拥塞和改善内存局部性来减少内存瓶颈。作者提出了一项研究，分析了多核和多核系统上不同 BWA 变体的性能。同时作者还使用不同的动态数据分布机制扩充了 MDPR，用两个有代表性的数据集和三个异构体系结构对它们进行了评估。

2. 多核-众核架构

目前，基于 NUMA（非统一存储器访问）架构的系统是高性能计算领域中使用最多的系统。这些系统通常与多核加速器结合使用,以合理的成本增加系统中可用的核心数量。在本节中，作者简要介绍在研究中使用的基本体系结构，即具有 NUMA 节点和多核加速器的多核系统。

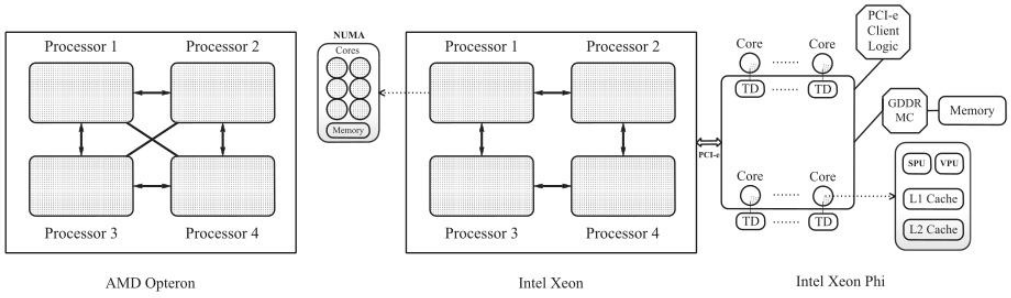

图 1 多核-众核异构架构

图 1 显示了本工作中使用的系统的主要体系结构。 两个多核系统描绘在图的左侧和中心（基于 AMD-Opteron 的服务器和基于 Intel Xeon 的服务器）。 Intel Xeon Phi 位于右侧。 我们的 AMD Opteron 和 Intel Xeon 系统具有 NUMA 节点，跨处理器具有不同的链路级别。

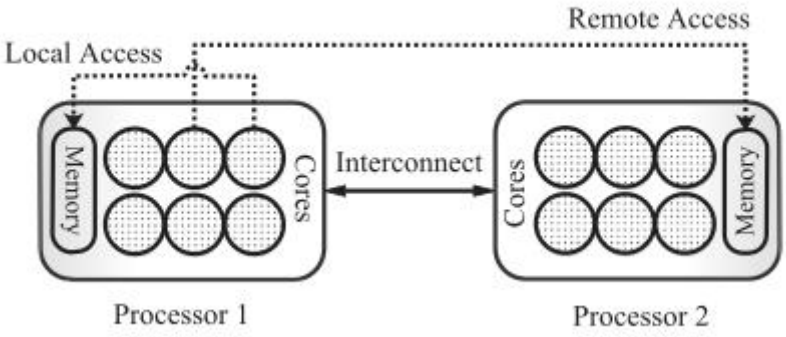

图 2 NUMP 内存架构

AMD-Opteron 和 Intel Xeon 系统都具有 NUMA 架构和四个独立插槽，每个插槽都有一个或两个多核处理器，每个处理器都连接到本地存储器组。 每个处理器及其相应的存储体被称为 NUMA 节点。 NUMA 系统最重要的特征之一是来自运行在本地核心（位于同一 NUMA 节点中）的线程的访问具有比访问位于不同 NUMA 节点中的物理内存的访问更低的延迟（如图 2 所示）。 存储器访问中的这种不对称性在执行并行应用程序时涉及额外的复杂性，因为对远程库的访问增加了执行时间，并且在基因组对齐的情况下，还引入了拥塞问题。

3. 基因组对齐器

基因组读取对准器提供参考基因组内短读取的相对位置。 尽管每个对准器存在特殊差异，但它们都具有相似的操作模式，可归纳如下：

1.有一组读数可以独立地映射到参考基因组。

2.有参考基因组数据结构（基因组索引）这是只读数据，用于映射每个单独的读取。

3.结果包括填充共享数据结构，该结构将在对齐结束时写入输出文件。

作者将研究重点放在 Li [7]编写的 BWA 上，这是基于 FM 索引的家族中最受欢迎的序列比对工具之一。 BWA 由三种算法组成，BWA-backtrack，BWA-SW 和 BWA-MEM。作者的研究侧重于已经被文献中其他工作使用的 BWA 回溯，例如 mBWA [6]，pBWA [32]和 BMIC [36]。

BWA 预先创建基因组参考索引。之后，使用固定大小的块处理短读取数据，以线程之间的循环模式。 BWA 中的每个线程处理一个数据块，默认为 256 K，独立映射到基因组参考。 与许多其他对齐器一样，BWA 具有多线程功能（使用 Pthread 库）并行解决读取映射操作，因为此操作中没有数据依赖性。 向对齐器添加多线程并不能保证有效地使用处理资源。 事实上，当线程数量增长足够大时，观察效率会降低是很正常的。

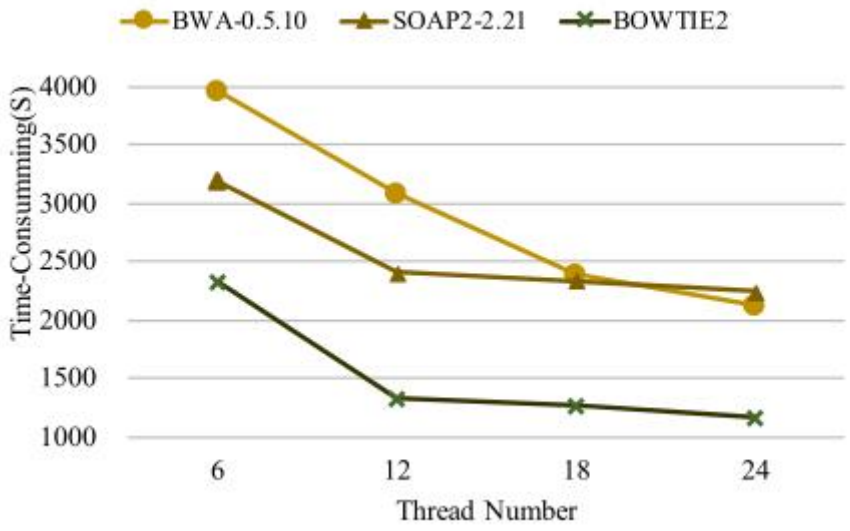

图 3 Scalability on 2-socket 2-NUMA 24-thread system.

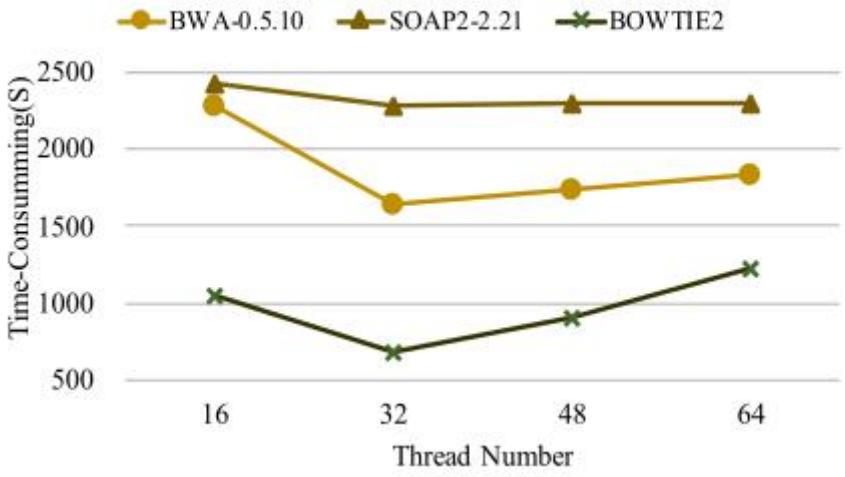

图 4 Scalability on 4-socket 4-NUMA 64-thread system.

这种现象如图 3 以及图 4 所示。 这些图说明了三个序列对齐器（BWA，SOAP2 和 BOWTIE2）获得的结果，这些序列对齐器在绘制针对人类基因组的 17.6GB 的两个大型短读取基因组实例时都是基于 FM 索引的应用。 每个图都说明了在两个不同服务器上执行对齐器时获得的结果。两台服务器都基于NUMA架构,但它们在可用的处理器,核心和NUMA

节点总数上有所不同。 第一个系统（图 3）有 2 个插座和 2 个 NUMA Intel Xeon CPU，每个 CPU 有 12 个内核。 使用 Hyper Thread 技术可以同时执行二十四个线程。 第二个系统（图 4）是一个 4 插槽 4-NUMA Intel Xeon CPU。 每个插槽包含 8 核处理器，因此核心总数为 32，并且可以同时执行 64 个线程。 在这两种情况下，每个插座都连接到存储体，该存储体构成 NUMA 节点。 NUMA 节点通过 QuickPath Interconnect 链接连接。

此外，用于处理应用程序线程的库和组织可能导致获得的执行时间的某些变化。在作者的研究论文中，观察到使用 Pthread，OpenMP 和 Cilk 的并行化在性能方面表现出明显的差异，OpenMP 和 Cilk 是 Pthread 的最佳替代品，Pthread 是最初在 BWA 中使用的库。除了线程库之外，基因组读取对齐器中的主要性能问题源于 NUMA 系统中存储器的使用。内存分配是根据第一触摸策略完成的，这意味着基因组参考索引在单个库中分配，当多个线程尝试访问它时将成为瓶颈。

4. 基于数据并行化和复制的策略

BWA 的可扩展性问题主要是由于需要多个线程访问参考基因组所在的同一存储体而产生的争用。为了缓解这个问题，作者提出了一系列基于基本通用方案的备选方案。在该方案中，存在基于处理读取组的线程组的第一级并行性。另一方面，参考基因组将在不同的实例中复制，每个实例都可由不同的线程组访问。线程组的数量和参考基因组的复制品的数量可以变化，并且可以以不同方式组织。最终目标是分析哪种变体在多核或多核架构中获得最佳结果。在众核系统中，作者专注于找到使用系统所有资源（即以对称模式执行）有效的策略。同时作者还对多核系统的解决方案感兴趣，这些解决方案可以轻松转移到由性能不同的服务器组成的多核系统。

4.1 同构体系结构的数据管理：DP，DR 和 DPR 策略

作者研究了涉及并行和复制的数据管理。 如图 5 所示，在对齐过程中涉及两个数据集。N 对应于所有短读取的集合，并且 n 是其中 N 被划分的读取子集的数量。 M 代表参考基因组，m 是由这种基因组构成的复制品的总数。 因此，读取子集称为 N_1，...，N_i，... ，N_n 和参考复制品称为 M_1，... M_j，M_m。

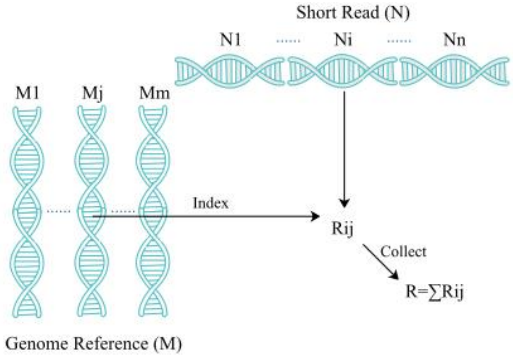

图 5 基因组序列对齐过程

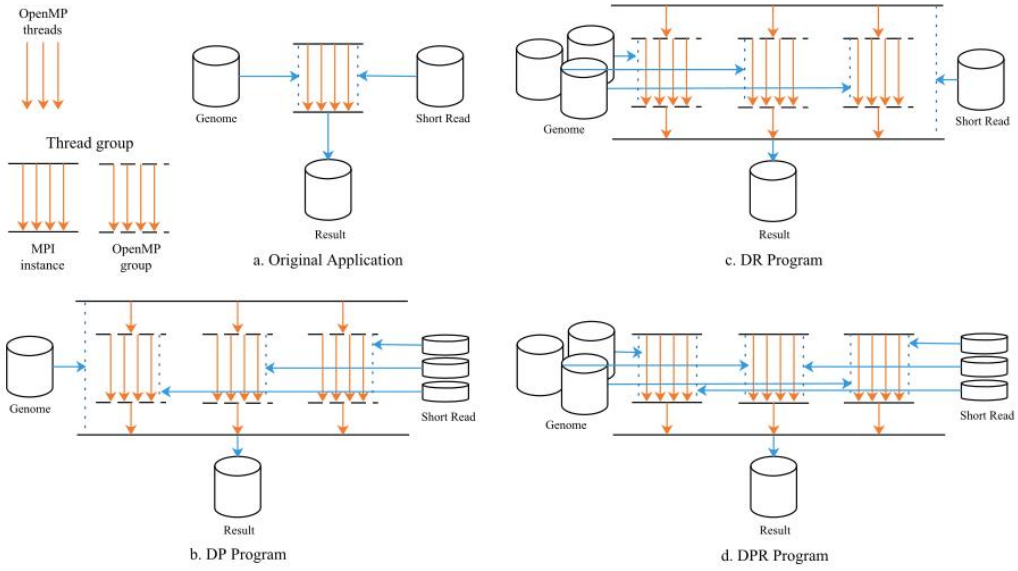

图 6 不同策略的实现

在作者提出的方法中，BWA 被组织为一组线程组（Ng），负责映射一些读取。 实际上，线程组表示 OpenMP 中的一组线程或 MPI 中的一个进程，如图 6 所示。 每个线程组使用不同数量的线程执行，具体取决于系统中可用的硬件核心。 对齐中的一个线程组在逻辑上被组织为一组线程，这些线程处理短读取的独立数据集并在所有线程之间共享基因组索引。对于潜在线程的总数 Nt 和线程组的数量 Ng，每个线程组由 Nt / Ng 线程组成。 例如，在 Intel Xeon 和 Xeon Phi 的情况下，Nt 分别为 24 和 240。 当应用两个线程组（Ng = 2）时，12（24/2）和 120（240/2）线程分别包含在 Intel Xeon 和 Xeon Phi 上的每个线程组中。

4.2 异构架构的数据管理：MDPR 策略

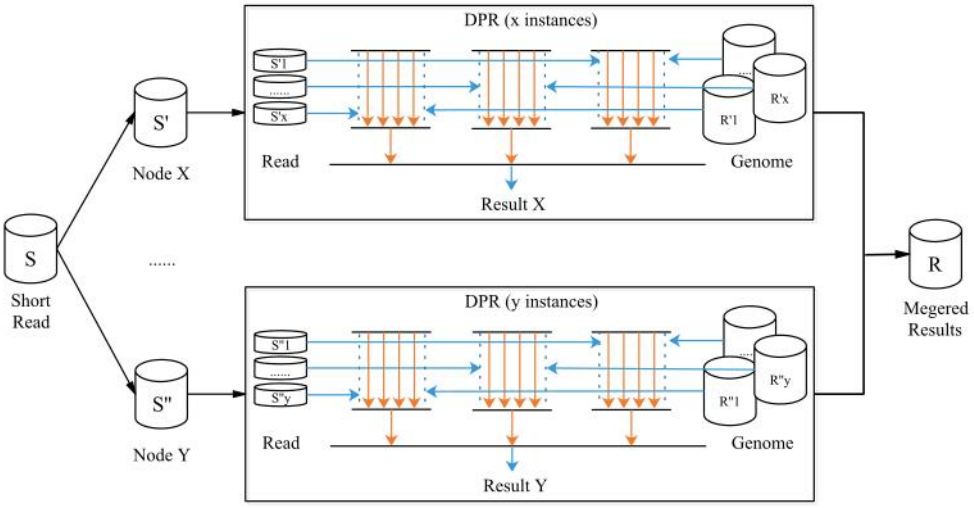

图 7 MDPR 策略

上述策略可应用于具有共享存储器的同构多核系统。 对于基于加速器的系统（在对称模式下执行）或在独立的多核系统的情况下，我们提出了一种推广 DPR 的新策略。 它被称为 MDPR（多级数据并行化和复制），如图 7 所示。 MDPR 的设计涉及与基于异构和分布式计算资源的环境的使用相关的若干挑战。

一方面，必须使用消息传递原语建立工作分配机制，这些原语不会在通信和同步方面引入过多的开销。 MDPR 最大限度地减少了通信和同步需求，因为没有中央过程负责读取和分配输入数据，以及收集输出结果。计算中的每个参与者进程直接读取其数据，并且仅需要被告知该数据所在的文件位置。另外，结果由每个过程独立产生，并且最终由中央过程通过简单的附加操作组合。另一方面，应用程序的配置和工作的分配必须以这样的方式完成，即在组成系统的每个节点中保证类似的执行时间。MDPR 的设计方式使其可以根据 NUMA 体系结构的特性（调整使用的实例数）动态配置，并结合数据分配机制（静态和动态），允许以不平衡的方式分发数据 为了使整个应用适应系统的每个节点所展示的相对性能。

短读取数据集（S）被划分为若干子集：系统中每个节点的一个子集（图 7 示出了两个节点的情况：节点 X（S）和节点 Y（S））。作者在节点 X 中有几个线程组或实例（表示为 S 1，...，S i，...，S x，1 <= i <= x）和节点 Y 中的几个线程组（表示为 S 1 ，...，S j,...，S y，1 <= j <= y）。通过考虑对准器的存储器要求（特别是参考基因组的大小）和每个系统中存储器组上的可用空间量来生成线程组。参考基因组复制品表示为 R 1 ,. 。 。 ，R x 和 R 1 ,. 。 。 ，R y。每个 BWA 实例执行多个线程以对齐分配给它的短读取。异构架构上的实例数量不需要相同。

重要的是，系统的所有节点之间的数据集的划分并不意味着通信的显着增加，因为每组线程仅提供有到所有读取所在的文件的适当指针。也就是说，数据的分发不是由读取数据文件并通过使用消息传递机制将块分发给其他进程的主进程完成的。在作者给出的例子中，主进程只将每个块的起点和终点传递给每个进程；之后，每个进程将直接读取可通过连接到所有计算节点的共享文件系统访问的文件部分。

根据此模式，如果所有线程组消耗相似的时间，则应获得最佳性能。然而，如果短读取均匀分布，则整个系统的异构性可能导致不同的执行时间。 通过以不均匀的方式划分短读数据集可以改善这种负载不平衡情况。这种划分可以通过不同的方式完成，作者评估了三种可能性：一种是将数据集分离的静态分布，另外两种是在运行时划分数据集的动态分配机制。

4.2.1 静态分布

在该模式中，预先计算数据集 S‘和 S‘‘的大小，并且在执行开始时静态地划分短读取的初始数据集。每个节点在开始时读取其块，而不需要稍后执行新的分发操作。S’和 S‘‘之间的比率是根据 DPR 在每个架构中实现的最佳结果所获得的相对性能来计算的。例如，在两个节点（X 和 Y）的情况下，对于节点 X 和节点 Y 上的一定数量的实例（分别是 x 和 y 实例），测量时间成本 Tx 和 Ty。然后，节点 X 和节点 Y 之间的 Ty / Tx 比率用于划分输入数据 S.即，根据在每个节点中实现的相对性能来计算 S‘和 S‘‘之间的比率。因此，对于节点 X，我们有 size(S’)=size(S)*Ty /(Tx+Ty)，对于节点 Y，我们有 size(S‘‘)=size(S)*Tx /(Tx + Ty)。显然，这种静态分布在运行时不会引入任何开销，但它需要先前执行几个示例来计算系统中每个节点的相对性能。

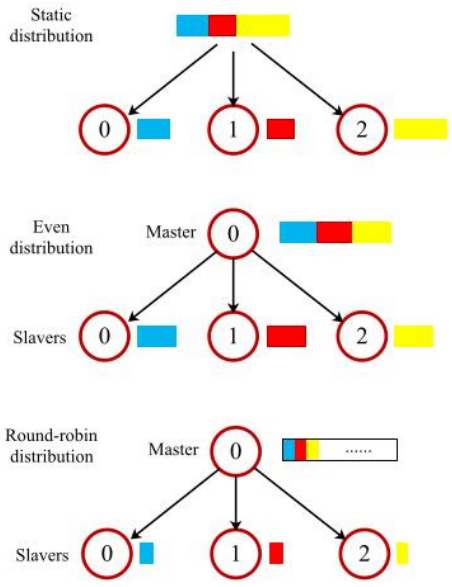

图 8 不同策略的数据分布机制

4.2.2 均匀分布

在这种情况下，MDPR 的所有实例都会收到一个数据块相同的大小。 如果实例的总数是（x + y）并且数据集的总大小是 size(S)，则每个实例处理大小的块(size(S)/(x+y))。节点 X 和节点 Y 之间的比率可以设置为 x / y。在节点 X 中，有 size(S')=size(S)*x/(x+y)，在节点 Y 中有 size(S'')=size(S)*y/(x+y)。与静态分布情况中发生的情况类似，原始数据集在执行开始时被划分，并且不会引入显着的开销。然而，如果节点的相对性能明显不同，则该方案通常不会导致良好的负载平衡。如图 8 的中心部分所示，主实例生成原始数据集的相等大小的分区，并且每个实例（从属）将处理它。

4.2.3 循环数据分布

此分发机制意味着主实例在完成以前的块时将分块读取分发给从属实例。此机制应保证比前一个更好的负载平衡，但代价是运行时的开销较大。最初，每个 BWA 线程处理一组相同大小的读取（默认为 256 K）。该大小在映射集合中所有读取的计算时间与该集合消耗的 I / O 时间之间表现出良好的折衷。在我们的实现中，这个集合大小已经被维护，因此，通过将 256 K 乘以每个实例中执行的线程数来计算 chunk 大小。例如，如果我们在具有 64 个可用线程的系统中执行 BWA 对齐器，则如果应用一个实例，则块大小应至少为 16 M（256 K * 64），如果应用了两个实例，则应为 8 M（256 K * 32）。该机制在图 8 的底部示出，其中短读取的块按照循环方式分布。实例在完成前一个实例后会收到一个新的 chunk。如前所述，块分布仅仅意味着相应块的起点和终点的通信。

5 结论

作者的研究还揭示了一个可以应用于生物信息学领域的其他问题的想法。尽管在该领域中现有数据集的数量和质量有了很大的增长，但在实践中，序列比对等问题的解决主要受参考基因组大小（通常在 3GB 和 30GB 之间）的限制，这是结构 应始终保留在主存中。具有序列的数据文件以块的形式逐渐读取，因此，它们不假设在当前系统中的存储器要求方面存在任何显着限制。因此，通过使用基于包含某些数据结构的副本的实例的体系结构来设计并行应用程序构成了具有低算法复杂度的解决方案，其可以在其存储器体系结构为 NUMA 类型的系统中提供显着改进，如 BWA 中的 MDPR。

参考文献

[1] C.F. Baes , M.A. Dolezal , J.E. Koltes , B. Bapst , E. Fritz-Waters , S. Jansen , C. Flury ,H. Signer-Hasler , C. Stricker , R. Fernando , et al. , Evaluation of variant

identification methods for whole genome sequencing data in dairy cattle, BMC Genomics 15 (1) (2014) 948 .

[2] I. Buck , Gpubench: evaluating GPU performance for numerical and scientific application, in: Proc. 1st ACM Workshop General-Purpose Computing on Graphics Processors (GP ^ 2’04), 2004 .

[3] L. Stein , Genome annotation: from sequence to biology, Nat. Rev. Genet. 2 (7)(2001) 493 .

[4] J.G. Reid , A. Carroll , N. Veeraraghavan , M. Dahdouli , A. Sundquist , A. English , M. Bainbridge , S. White , W. Salerno , C. Buhay , et al. , Launching genomics into the cloud: deployment of mercury, a next generation sequence analysis pipeline, BMC Bioinformatics 15 (1) (2014) 30 .

[5] S. Pabinger , A. Dander , M. Fischer , R. Snajder , M. Sperk , M. Efremova , B. Krabichler , M.R. Speicher , J. Zschocke , Z. Trajanoski , A survey of tools for variant analysis of next-generation genome sequencing data, Brief. Bioinformatics 15(2) (2014) 256–278 .

[6] Y. Cui , X. Liao , X. Zhu , B. Wang , S. Peng , mBWA: a massively parallel sequence reads aligner, in: 8th International Conference on Practical Applications of Computational Biology & Bioinformatics (PACBB 2014), Springer, 2014,pp. 113–120 .

[7] H. Li , R. Durbin , Fast and accurate short read alignment with burrows–wheeler transform, Bioinformatics 25 (14) (2009) 1754–1760 .

[8] S. Chen , M.A. Senar , Accelerating BWA aligner using multistage data parallelization on multi-core and many-core architectures, Procedia Comput. Sci. 80(2016) 2438–2442 .

[9] S. Chen , M.A. Senar , Improving performance of genomic aligners on intel xeon phi-based architectures, in: 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), IEEE, 2018, pp. 570–578 .

[10] S.F. Altschul , T.L. Madden , A .A . Schäffer , J. Zhang , Z. Zhang , W. Miller , D.J. Lipman , Gapped blast and psi-blast: a new generation of protein database

search programs, Nucleic Acids Res. 25 (17) (1997) 3389–3402 .

[11] B. Langmead , C. Trapnell , M. Pop , S.L. Salzberg , Ultrafast and memory-efficient alignment of short dna sequences to the human genome, Genome Biol. 10 (3)

(2009) R25 .