SALoBa：最大限度发挥数据局部性与工作负载均衡，实现 GPU 上的快速序列比对

Seongyeon Park†、Hajin Kim†、Tanveer Ahmad§、Nauman Ahmed§、Zaid Al-Ars§、H. Peter Hofstee§‡、Youngsok Kim† 和 Jinho Lee†*
†延世大学 §代尔夫特理工大学 ‡*IBM*{syeonp, kimhajin},@yonsei.ac.kr, {t.ahmad, n.ahmed, z.al-ars}@tudelft.nl, hofstee@us.ibm.com, {youngsok, leejinho}@yonsei.ac.kr

arXiv:2301.09310v1 [cs.DB] 23 Jan 2023

摘要——**序列比对是许多测序应用的重要基础。序列比对中一种常用策略是采用二维动态规划方法进行近似字符串匹配。尽管此前已有一些关于序列比对 GPU 加速的研究，但我们发现其中仍存在若干不足，限制了现代 GPU 全部计算能力的发挥。本文介绍 SALoBa，这是一种专注于种子延伸的 GPU 加速序列比对库。基于对先前研究及真实测序数据的分析，我们提出了利用数据局部性并改善工作负载均衡的技术。实验结果表明，与最先进的基于 GPU 的方法相比，SALoBa 显著提升了种子延伸内核的性能。**

索引词——**基因组测序、序列比对、史密斯-沃特曼算法、GPU加速**

I. 引言

随着新一代测序（NGS）技术的快速发展，DNA测序的成本正以超过摩尔定律的速度下降 [1]。然而，事物都有两面性，测序吞吐量的迅速提升意味着数据处理已成为一个更严峻的瓶颈。例如，在英特尔至强处理器上对人类基因组进行读段比对，如今所需时间已超过 20× 测序时间 [24]。

读段比对是序列比对中的一个过程，它将一段查询DNA（由测序生成）映射到参考DNA中的匹配位置。然而，由于测序错误和/或突变，参考DNA与查询DNA并不完全匹配。因此，该问题属于近似字符串匹配范畴。

解决这一问题的一种常用策略是种子生成与延伸：种子阶段定位若干精确匹配，而延伸阶段则基于动态规划（DP）执行近似匹配，例如采用史密斯–沃特曼算法 [57]。遗憾的是，史密斯–沃特曼算法的复杂度与输入长度的平方成正比；随着NGS技术不断发展，输入长度也在持续增长。

在这项工作中，我们重点关注扩展过程，它占据了读取映射执行时间的很大一部分 [12]。为解决这一计算瓶颈，人们提出了许多采用专用集成电路 [22], [23], [58] 或现场可编程门阵列 [10], [25], [26], [28], [62] 的专用硬件加速器方案问题。然而，这类加速器尚未在实践中得到广泛应用。专用集成电路的生产成本高昂，除非能够保证大规模生产，否则很难进入市场。现场可编程门阵列受这一问题的影响较小，因为市场上已有许多 FPGA 加速卡。然而，FPGA 加速卡尚未成为主流选择，因为它们价格昂贵且难以编程。

在这种情况下，基于 GPU 的加速方案是可行的选择，因为成本更低、市场上更容易获得，而且软件开发所需的工作量也更少。因此，许多用于生物信息学的 GPU 库应运而生 [3], [9], [48]。随着 GPU 计算能力的快速提升 [17]，GPU 很可能会继续成为加速序列比对的首选方案。

然而，通过分析最先进的 GPU 加速种子延伸技术，我们发现了若干尚未利用的性能提升机会，这些机会已变得至关重要，尤其是在字符串查询较长时。考虑到第三代测序仪生成的测序读段长度正在迅速增加 [55], ，理想情况与现有软件之间的性能差距将会扩大。具体而言，我们发现现有的 GPU 内核存在以下问题：1）内存利用效率低；2）受到负载不均衡的影响。

本文提出 SALoBa（**序**列**比**对与数**据局**部性和工作负载**均**衡），旨在解决上述问题并实现卓越性能。借鉴若干 ASIC/FPGA 加速器的经验 [10], [23], [62]，SALoBa 集成了一个能够更好地利用现代 GPU 计算能力的内核。

SALoBa 通过为一个查询-参考序列对分配多个 CUDA 线程来利用查询内并行性。尽管已有一些研究探讨过这种方案 [13], [35], ，但这些方案存在内存访问模式低效和资源利用率不足的问题，性能不及当前最先进的方法；后者仅使用查询间并行。SALoBa 通过引入两种新技术来解决这些问题：向全局内存延迟溢出和子 warp 调度。在延迟溢出中，数据首先累积到 CUDA 共享内存中，随后以合并访问方式溢出到全局内存。通过轮换使用双缓冲共享内存区域，可大幅减少全局内存中的冗余，且不会给共享内存带来太多额外开销。

*通讯作者

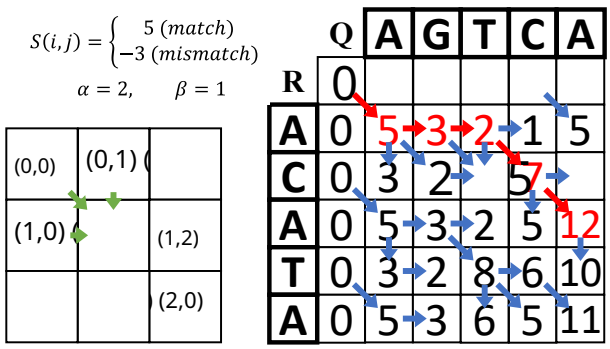

图 1：种子延伸算法示例。

此外，子 *warp* 调度 将一个线程束划分为多个子束，以缓解资源利用率不足的问题。虽然这会略微加剧工作负载不平衡，但更高资源利用率带来的收益通常足以抵消工作负载不平衡所产生的开销。

根据实验结果，在多种序列长度下，SALoBa 在种子延伸内核上的性能显著优于最先进的 GPU比对器。在长度各异的真实世界数据集上进行测试时，由于工作负载均衡更佳，我们取得了更优的结果。我们的贡献总结如下：

- 我们发现当前的库中存在若干尚未利用的性能提升机会。
- 我们提出了 SALoBa，与当前最先进的基于图形处理器的比对库相比，它能够显著提升速度。
- 我们进行了大量评估，包括使用一种流行的读段比对软件中的合成数据和真实世界数据，以展示 SALoBa 的效率。

II. 背景

A. 种子延伸

序列比对问题的核心是近似字符串匹配。每当发生错配时，都可能有多条路径可供选择，因此其复杂度会随着输入序列对的长度增加而迅速上升。

种子-延伸策略是一种高效执行比对的方法。基于良好的比对通常包含许多匹配这一观察，该策略首先寻找多个精确匹配，称为 种子。然后以找到的种子为基础，向两个方向进行延伸，并计算近似匹配得分。

种子延伸通常使用动态规划算法来填充二维动态规划表，其复杂度为 $O(N^2)$。常用算法包括史密斯–沃特曼算法 [57] 和尼德曼–翁施算法 [51]。每个单元格的得分根据预先设定的差异类型（即插入、删除和错配）得分计算。针对频繁出现的长间隙进行调整后，仿射间隙函数按如下方式计算动态规划单元格 $H(i,\ j)$ 的得分：

$$H(i,j) = max \begin{cases} 0 \\ E(i,j) \\ F(i,j) \\ H(i-1,j-1) + S(i,j) \end{cases}, \quad (1)$$

$$E(i,j) = max \begin{cases} H(i,j-1) - \alpha \\ E(i,j-1) - \beta \end{cases}, \quad (2)$$

$$F(i,j) = max \begin{cases} H(i-1,j) - \alpha \\ F(i-1,j) - \beta \end{cases}, \quad (3)$$

其中，$S(i,\ j)$是一个评分函数：当 i 处的参考序列与 j 处的查询序列匹配时返回正值，否则返回负值。此外，E 和 F 是用于跟踪连续间隙的辅助变量。最后，α 和β 分别根据新间隙或延续间隙计算不同的间隙罚分。

图 1 展示了一个动态规划表示例，其中 Q 表示查询字符串，而 R 表示参考字符串。计算一个单元格时，需要用到与其相邻的三个单元格：上方、左侧和左上方的单元格，如绿色箭头所示。动态规划表中最高得分的轨迹表示算法找到的最佳匹配，以红色箭头和数字标示。

B. 基于 GPU 的种子延伸基线

此前已有多种尝试使用 GPU 加速种子延伸。测序读段数据中的每个碱基都以八位字符数据类型表示。然而，序列中只有五种碱基：A、C、G、T/U 和 N，其中 T 用于脱氧核糖核酸，U 用于核糖核酸。N 表示未知碱基。由于存在五种碱基，因此每个碱基至少需要三位来表示。三位表示在现代架构中处理效率低下，因此通常采用四位打包表示 [3],[9]，也有一些工作采用八位表示 [35]。

现有的基于 GPU 的种子延伸方法，大致可按所利用的并行性分为两类：查询内并行和查询间并行。查询内并行是指同时处理动态规划表中的多个单元格。如图1所示，动态规划表中存在反对角线形式的并行性。例如，处理完单元格 (0,0) 后，可以并行处理单元格 (0,1) 和 (1,0)。随后，可以并行处理单元格 (0,2)、(1,1) 和 (2,0)。许多 ASIC/FPGA 加速器都成功利用了这一特性，通常采用一维脉动阵列结构 [10], [23], [62]。一些基于 GPU 的方法也利用了这种并行性 [13],[35]。然而，由于资源利用不足和内存访问效率低下，这些方法往往无法获得足够的加速 [13]。另一方面，查询间并行是指直接并行处理多个查询-参考序列对。由于后者更容易优化资源利用率，许多成功的库都采用了这种方法 [3], [9], [39],[45]。

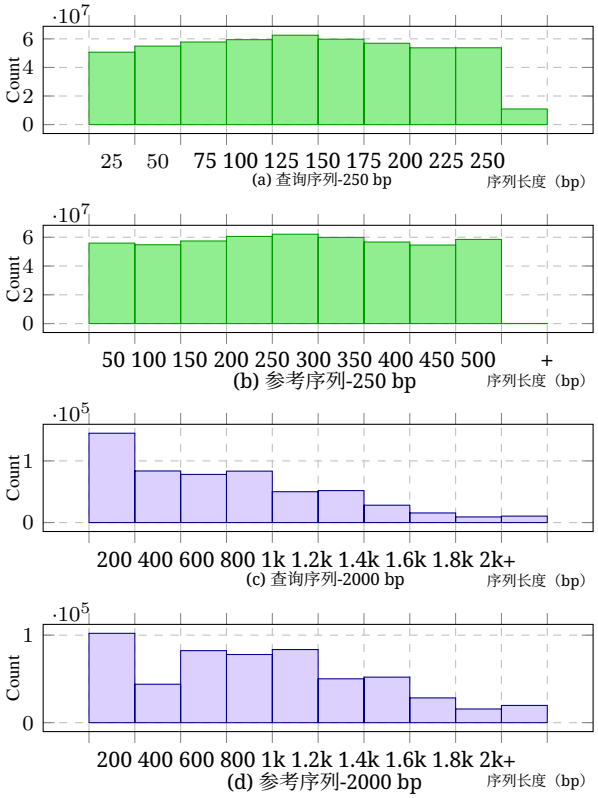

图 2：作为 BWA-MEM 种子延伸输入的序列分布 [37]。

在采用查询间并行的库中，GASAL2 [9]以当前最先进的性能表现而闻名。本节将详细介绍其策略。GPU寄存器的宽度为 32 位，因此每次可获取查询序列和参考序列中的 8 个碱基。因此，一次处理 8×8 个单元格是很自然的做法，这些单元格对应一个单字参考序列和一个单字查询序列。处理完 8×8 个单元格组成的块后，线程向右移动。为此，必须存储最右侧一列的 8 个单元格，以满足依赖关系。通过将这些动态规划单元格的值保存在寄存器中，即可满足依赖关系。当线程到达表格最右端时，会移至动态规划表的底部。此时，为满足依赖关系，必须访问顶部的单元格；线程会将每个 8×8 块底部的 8 个单元格存储在全局内存中。因此，考虑到矩阵大小为 $N \times N$，用于读取和写入依赖单元格值的全局内存访问量为 $N \times N/4$ 。

III. 动机——诊断

在本节中，我们分析最先进的比对器 GASAL2 [9]，并探讨进一步提升性能的潜在机会。首先，我们揭示了各个查询序列之间存在显著的负载不均衡。其次，我们发现全局内存的使用未实现合并访问，因此需要进行大量冗余内存访问。

表 I：现有 GPU 比对器存储和访问的数据量

Data	数量
必要	$2N$
存储的	$2N + N^2/4$
访问（截至 Pascal [4]）	$128N + 16N^2$
访问（Volta [5]之后）	$32N + 4N^2$

A. 负载不均衡

如第 II-B 节所述，GASAL2 的主要并行化方案是查询间并行。在让每个 CUDA 线程处理一个查询序列的策略中，其主要优势在于无需复杂的线程间同步或开销巨大的通信。

然而，这种方法的一个缺点是忽略了种子延伸工作负载的差异。种子生成步骤会将查询序列和参考序列的长度作为种子延伸的输入。因此，单个输入的长度各不相同，导致显著的负载不均衡。

图2 绘制了两种类型读段的查询序列和参考序列长度分布（详细设置见第 V 节），这些读段来自广受欢迎的比对软件 BWA-MEM [37]。如图所示，两种分布的取值范围均从零到数百或数千，且没有明显的聚集趋势，这意味着 GPU 中存在大量线程束分歧。如图所示，最短字符串和最长字符串之间的长度差，查询序列和参考序列均可达到 10×。鉴于计算复杂度与两个字符串长度的乘积成正比，实际中的工作负载不平衡可能非常严重。

B. 内存低效

采用四位序列打包时，一次处理 8×8个动态规划单元是合理的。转到下一个单元格时，[1] 当前单元格的内容可以保存在寄存器中。然而，种子延伸的依赖结构还要求长期存储单元格信息，以便处理下一行的 8×8 单元格。这些中间数据远超寄存器文件的容量，通常会存储在全局内存（即动态随机存取存储器）中。

然而，这种方案会造成两种低效。首先，内核结束时，中间数据无需继续保留在全局内存中，因此属于开销。减少这项开销有助于提升性能。其次，图形处理器全局内存的最小访问大小为 128 B（而 Volta [5] 架构为 32 B，如 [32] 所示），而单个单元格的数据大小仅为 4 B。如果未被二级缓存捕获，就会针对相同的中间数据产生多次冗余访问，导致内核效率低下。

[1]不失一般性，我们假设单元格按从左到右、从上到下的顺序处理。

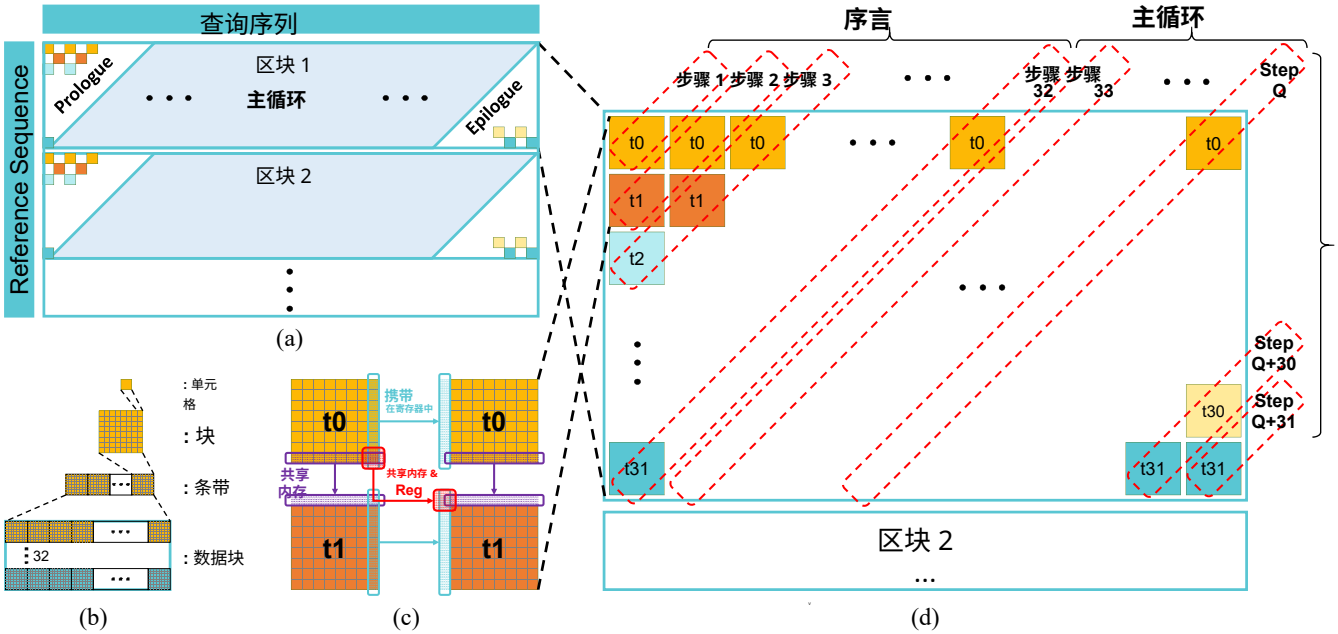

图 3：查询内并行。（a）展示了使用一个线程束的并行化策略，（b）说明了本图中使用的术语。（c）演示了如何处理单元格之间的依赖关系，（d）描绘了反对角线结构的使用。

表I 列出了数据量、扩展所需的输入数据（必要）、为中间数据存储的数据（已存储），以及由于访问粒度而访问的数据（已访问）。该表表明，即使在现代架构上，效率低下的问题也有多方面的表现，这也为进一步改进提供了机会。在这项工作中，我们展示了 SALoBa 能够减少大量此类访问，从而带来更优异的性能。

IV. SALOBA 设计

A. 查询内并行性

由于动态规划表中的每个单元格都依赖于上方、左侧和左上方的单元格，因此已知查询内并行性以反对角线形式存在，如图 3 所示。基于这一思路，我们使用多个 CUDA 线程并发处理反对角线上的元素。略有不同的是，单元格并非由各个线程每次计算一个，而是考虑到参考序列和查询序列采用了 4 位打包，将单元格分配给 8×8 块。

在该库的第一个版本中，我们决定最多由 32 个线程——即一个线程束——协同处理一个查询序列。由于线程束内同步的开销相对较低，甚至可以忽略不计（对于引入独立线程调度之前的图形处理器而言 [5]），因此这是一个很有吸引力的设计选择。虽然理论上可以使用超过 32 个线程，但这需要线程块级同步（即 `__syncthreads()`），从而带来不可忽略的开销。

在线程之间进行通信时，我们使用了 CUDA 共享内存。将共享内存用作通信通道并在每次迭代中重复使用，只需要固定数量的共享内存，并且对共享内存的所有访问都不会产生冲突（见第 IV-B节）。

图 3 (a) 和 (b) 展示了 SALoBa 的处理过程。最大的计算单元称为数据块，它们是动态规划表的水平分区。数据块高为 32 个块，宽度则覆盖整个查询序列。

如图所示，每一步中，一个线程处理数据块中一个条带（即一行块）里的一个块。第一步中，第一个线程通过从查询序列和参考序列中各读取一个 32 位字来启动处理，这足以处理由 8×8块 单元格组成的区域。

在接下来的步骤中，已准备好处理的单元格数量每步增加一个。因此，参与计算的线程数量不断增加，直到第 32 步，从而形成一个持续 31 步的序言。在主循环阶段，[2] 线程束中的所有线程同时以反对角线方式处理块，每步向右推进一个块。此过程持续 $Q-31$ 步，其中 Q 是查询序列中的块数（即 $\lceil query\ length/8 \rceil$）。当第一个线程到达行末时，尾声开始，参与的线程数量每步减少一个，直到最后一个线程到达行末。尾声结束时，处理整个数据块总共耗费的步数为 $Q+31$。

每当线程推进到下一步时，就会访问顶部、左侧和左上方单元格中的依赖数据（公式3）。图 3 (c) 展示了具体做法。计算一个块时，左侧八个单元格中的数据是当前线程在上一步计算得到的。因此，

[2]这种模式也称为“内核”，但我们决定称其为主循环阶段，因为若使用“内核”一词，容易与 GPU 内核混淆。

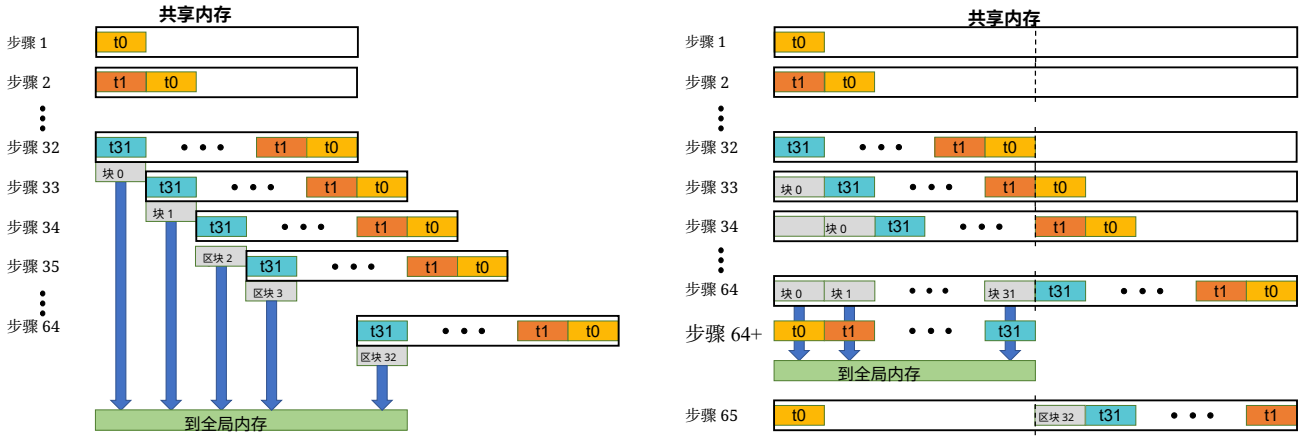

图 4：惰性溢出优化。（左）：朴素的全局内存访问。（右）：提出的优化访问方式。每个小矩形表示为依赖关系存储的 8×1 个单元格。标记为 't#' 的彩色矩形表示线程 # 正在访问该单元格，标记为 'block #' 的灰色矩形表示该单元格来自块 ID #，并已准备好溢出到全局内存。

这些数据可以存储在寄存器中。顶部八个单元格中的数据由相邻线程在上一步计算得到。处理完一个块后，每个线程都会将底部的 8×1 个单元格写入共享内存。在下一步中，每个线程会读取共享内存中下一个位置的数据，以便接收上方块的 8×1 个单元格。最后，左上方的一个单元格依赖于当前块，而线程在上一步中从共享内存接收到了该单元格的数据。幸运的是，这个数据也可以通过寄存器传递。因此，存储在寄存器中的单元格数量变为九个，而不是八个。

该方案不仅减少了线程束分歧，还改善了内存访问。在先前的技术中，每个条带的所有底部单元格都必须存储到全局内存中，并从全局内存读回。相比之下，采用查询内并行时，只需将每个数据块底部的单元格（而非条带底部的单元格）存储到全局内存中并从中读取。对于一个包含 32 个线程的线程束，这使中间数据访问量减少到 1/32。

B. 惰性溢出至全局内存访问

处理一个数据块时，必须存储最底部单元格中的数据，以便处理下一个数据块最顶部的单元格。通常，这很容易超出共享内存容量，因此必须存储在全局内存中。在图中所示的朴素方案里，4 （左），最后一个线程（$t31$）将一个块底部的单元格存储到全局内存中，而第一个线程（$t0$）在处理下一个数据块时从全局内存中读取这些单元格。然而，正如第III节所述，这会造成大量非合并访问，成为效率低下的原因。

为了解决这个问题，我们采用了延迟溢出，通过为每个线程束使用双缓冲共享内存来实现，从而减少全局内存访问量。图4 （右）标出了每一步线程写入的共享内存位置。为每个线程束分配一个大小为 $2 \cdot dim(block) \cdot \#threads$的共享内存区域。如第IV-A节所述，第一个线程（$t0$）从共享内存的第一个位置开始写入，并在下一步之前将该位置传递给第二个线程。在每一步中，线程从共享内存读取数据，处理一个块，并将块底部的单元格覆盖写入共享内存。然后，所有线程向后移动一个块位置，以进行下一步。

序言之后，线程会留下数据轨迹，由最后一个（$t31$）线程写入。幸运的是，这些正是需要存储到全局内存中的数据。当第一个线程（$t0$）在步骤 64 到达缓冲区末尾时，共享内存左侧已有一段连续数据，其大小足以由一个线程束处理。线程停止处理并聚集起来，以合并写入的方式将数据溢出到全局内存。随后，线程在共享内存中循环，块数据的轨迹开始在共享内存右半部分形成，之后会被溢出到全局内存。

要处理下一个数据块，必须读取前一个数据块溢出的数据，以便第一个线程利用这些数据计算单元格值。这个过程也以相同的方式进行，使用双缓冲共享内存，方向与写入相反。

C. 子 *warp* 调度：并行性与工作负载均衡之间的权衡

所提出的策略——每个查询一个线程束——可以消除线程束内线程之间的工作负载不平衡，因为它们协同工作。然而，正如第IV-A节所示，序言和尾声阶段存在一个问题。这些阶段的平均线程利用率为 50%（因为其形状为三角形），且其规模与线程束大小成正比。动态规划表必须显著大于线程束中的线程数，才能摊销这一开销。然而，根据图 2，表维度的大小仅为数百左右，只比 32 大几倍，有时甚至更小。

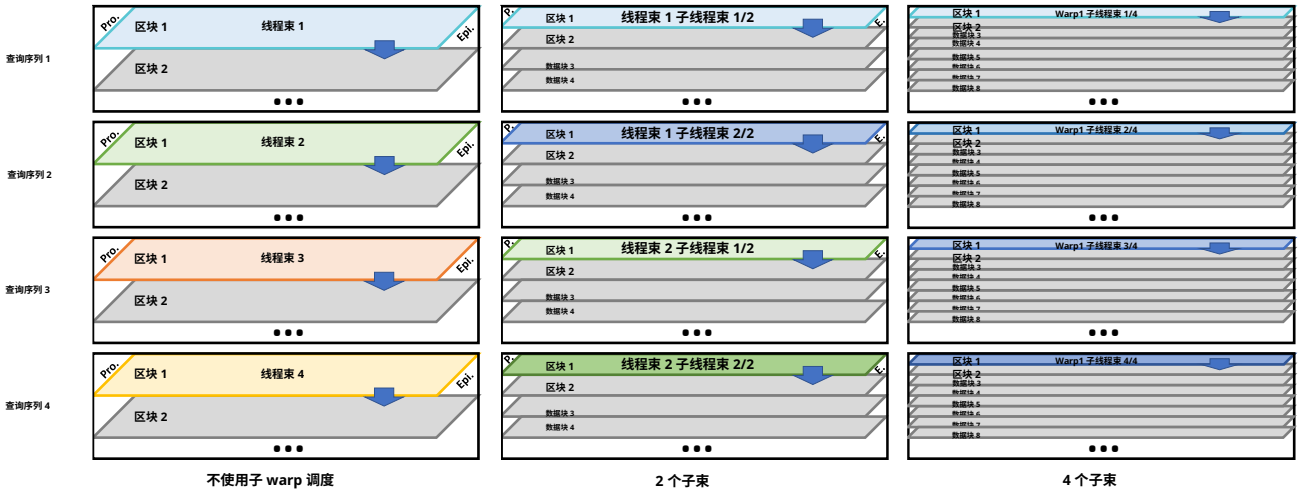

图 5：子 warp 调度。

我们选择将线程束拆分为多个子束，并让它们处理单个查询序列，如图 5所示，这与图处理领域采用的 VWC [27]理念类似。其本质是在工作负载均衡与资源利用率之间进行权衡。

使用较多子束时（即子线程束大小较小时），序言和尾声的规模会变小，因此平均利用率会提高。然而，由于工作负载不平衡，使用多个子束的方案更容易出现线程束分歧。相反，使用较少子束时（即子线程束大小较大时），出现负载不均衡的可能性较低，但序言和尾声所占比例增加会导致利用率降低。根据经验，我们发现，在我们的设置中使用两个或四个子束效果最佳（参见第 V 节）。

一个潜在的缺点是，使用子束会使内存访问优化变得复杂。如果采用相同的双缓冲技术，访问连续内存地址的线程数将少于 32。对于 NVIDIA Volta [32]之后的架构，使用大于八个线程的子束不会造成明显问题。对于较旧的架构，可以通过分配 $N+32$个共享内存槽位，而非 $2N$ 来解决这一问题。通过使活动区域循环移动，并让整个线程束中的所有线程共同将 32 个数据槽写入全局内存，可以实现完全合并访问，但共享内存容量需求会略有增加。

V. 评估

A. 实验设置

我们在两个平台上评估了 SALoBa。首先，在一套“经济型”系统上，我们使用了基于 Turing [6] 架构的 GTX1650 GPU 卡。服务器配备六核 Intel i5-9600K 中央处理器和 16 GB 内存。我们还在一套“高端”系统上进行了实验，该系统配备基于 Ampere [17]架构的 RTX3090 GPU 卡。服务器采用单路配置，配备 12 核 AMD EPYC 7272 中央处理器和 128 GB 内存。两台机器均运行 Ubuntu 18.04，使用 CUDA 11.2 版本和 NVIDIA 驱动程序 460.27.04。

作为比较，我们使用以下库中的种子延伸内核作为基线，列于表 II。**SOAP3-dp** [50] 是较早的一种 GPU 辅助短读段比对算法，采用查询间并行。**CUSHAW** 是一系列 GPU 辅助短读段比对算法。其大多数成员 [43], [44], [48] 仅在种子阶段使用 GPU，**CUSHAW2-GPU** [45] 则采用与 SOAP3-dp 类似的策略，利用 GPU 加速种子延伸部分。通过精简全局内存存储格式并使用 CUDA 纹理内存，可获得更好的性能。**NVBIO** [3] 是 NVIDIA 设计的可复用组件库，旨在利用 CUDA 加速生物信息学应用。**SW#** [35] 是一种采用查询内并行的算法。它将动态规划表划分为多个反对角线分区，并在每次内核启动时处理一个分区。由于它会针对每条查询序列多次启动内核，因此适用于超长序列。**ADEPT** [13]是采用查询内并行的算法中最新的一项工作。它结合使用洗牌指令和二进制掩码，以利用反对角线并行。最后，**GASAL2** [9] 是最先进的库。它进一步在 GPU 设备上执行序列打包，从而实现了出色的性能。

表 II：参与比较的基线内核

内核	并行性	位宽	映射
SOAP3-dp [39]	查询间	2 位	一对一
CUSHAW2-GPU [45]	查询间	2 位	一对多
NVBIO [3]	查询间	2、4、8 位	一对多
GASAL2 [9]	查询间	4 位	一对一
SW# [35]	查询内	8 位	一对多
ADEPT [13]	查询内	8 位	一对一

*(所有内核均已修改为至少支持四位的 GPU 辅助打包，并支持一对一映射。)

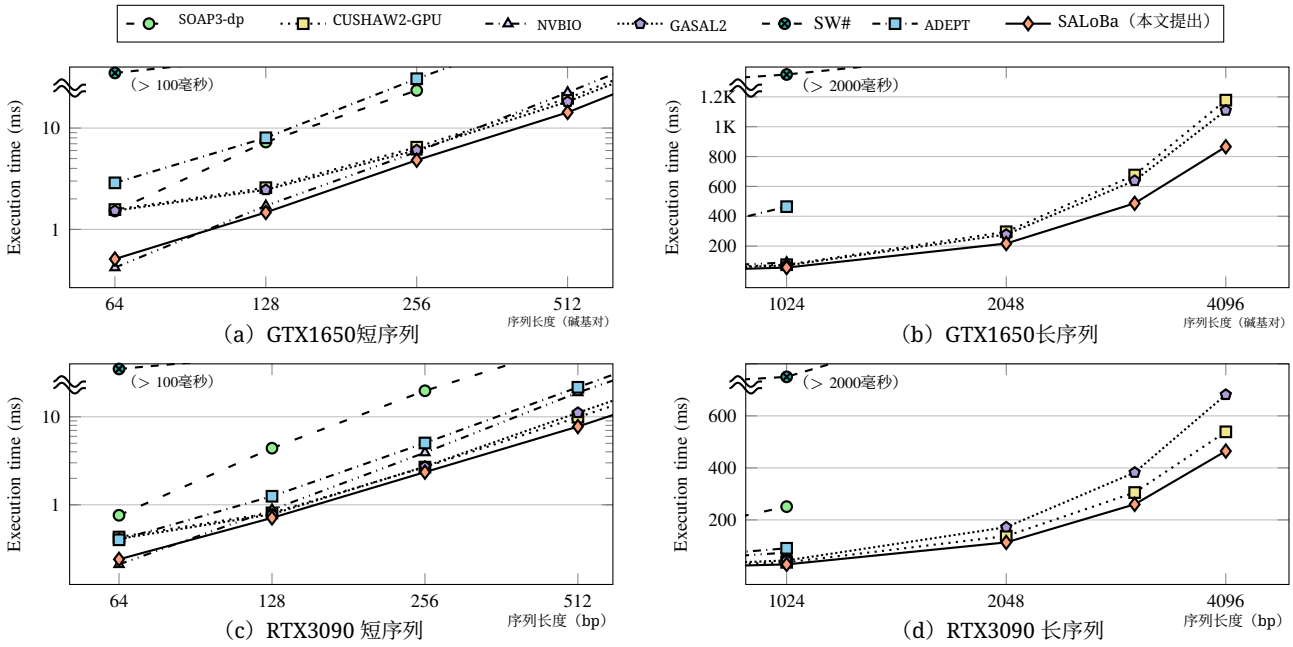

图6：延伸内核的性能比较。(a) 和 (c) 的 y 轴采用对数刻度，以便更清晰地显示。

为进行公平比较，我们尽最大努力在相同环境下优化现有方法。最重要的是，我们假设所有方法都在 GPU 上进行序列打包。众所周知，GASAL2 通过在 GPU 上执行自定义序列打包内核，实现了最先进的性能。尽管这一策略效果显著，但序列打包与种子延伸本身是正交的，因此同样的方法也可应用于所有其他内核。我们修改了 SOAP3-dp、CUSHAW2-GPU 和 ADEPT 内核，使其通过统一采用四位打包来支持五种可能的字面量类型。SW# 则保留原有的八位打包，因为修改它以支持打包会使内存访问行为变得复杂。此外，我们还修改了部分仅支持一对多比对模式的内核（CUSHAW2-GPU、SW# 和 NVBIO），使其支持一对一比对。

B. 内核性能测量

在本小节中，我们使用等长输入读段，对不存在负载不均衡的种子延伸内核进行性能比较。为测量不同输入长度下的性能，我们使用了一个内部开发的序列读段模拟器，其功能类似于 Wgsim [7]，为长度介于 64 到 4096 之间的每种长度生成合成读段。每个内核每次调用处理 5,000 条读段，共调用 200 次，并对结果取平均值，得到图6所示的结果。执行时间使用 $cudaEventElapsedTime()$ API 测量。

图 6 (a) 和 (b) 展示了各方法在 GTX1650 显卡上的性能比较，(c) 和 (d) 展示了各方法在 RTX3090 显卡上的性能比较。对于长度为 128 碱基对或更长的序列，SALoBa 的性能优于所有其他方法。对于 64 碱基对这样很短的序列，NVBIO 的执行时间略短于 SALoBa（GTX 1650 上分别为 0.42 毫秒和 0.51 毫秒，RTX 3090 上分别为 0.21 毫秒和 0.24 毫秒）。这是合理的，因为采用查询内并行时，序言和尾声造成的资源利用不足影响相对显著。然而，盈亏平衡点出现在 128 碱基对，此时 SALoBa 减少全局内存访问的优势开始占据主导。与其他方法相比，除 SALoBa 外采用查询内并行的方法（SW# 和 ADEPT）表现较差。SW# 尤其缓慢，因为它将单个动态规划表拆分为多个内核调用，每次内核调用处理一个反对角线组，导致资源利用率极低。

一个颇令人意外的发现是，GASAL2 并非总是优于其他基线方法。虽然 GASAL2 报告称，与其先前的工作相比实现了数倍加速，但其加速主要来自 GPU 端打包，而非扩展策略。由于我们将 CUSHAW2-GPU 与取自 GASAL2 的 GPU 端打包模块相结合，因此在 GTX1650 上，各种长度下的结果都与 GASAL2 相当；在 RTX3090 上，对于较长的序列长度，性能略好。

SALoBa 在短序列长度下取得的最佳加速比出现在 512 碱基对时。与 GASAL2 相比，在 GTX1650 上加速 27.7%，在 RTX3090 上加速 43.6%。对于较长的序列长度，一些基线内核因结构限制（ADEPT）或设备内存有限（NVBIO 和 SOAP3-dp）而无法运行。随着全局内存访问开销在执行时间中所占比例降低，较长序列长度下的性能趋势变得更加一致。另一原因可能是，与序列长度相比，序言/尾声的相对大小有所减小。对于长度等于或超过 1,024 碱基对的输入，SALoBa 相较于 GASAL2 的加速比稳定在 GTX 1650 上约 30%、RTX 3090 上约 50%。与 CUSHAW2-GPU 相比，

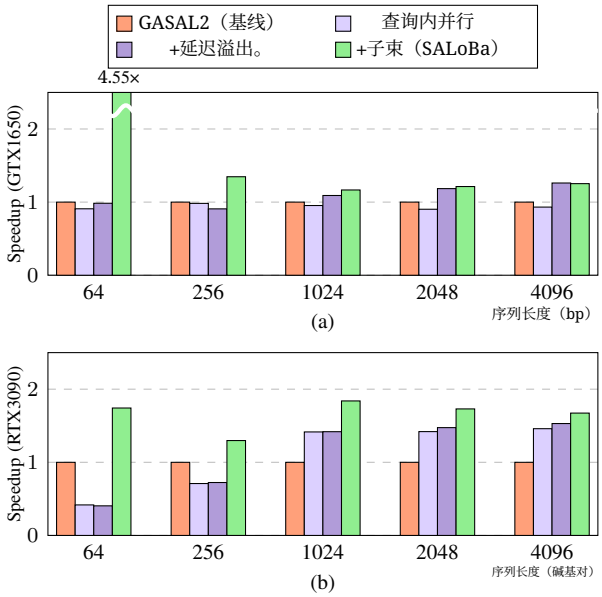

图 7：GTX1650（a）和 RTX3090（b）的消融研究。

在 GTX1650 上加速约 40%，在 RTX 3090 上加速约 20%。

C. 消融研究

图 7 展示了构成 SALoBa 的三种技术各自的贡献。加速比以 GASAL2 为基准进行归一化；在我们关注的各种序列长度下，GASAL2 的表现都相当不错。对于较短序列（≤1024），效果最显著的技术是子 warp 调度，这符合预期，因为输入较短时，序言和尾声所占的比例都很大。因此，切换到查询内并行会导致性能下降。直接采用子 warp 调度可减少开销，并带来显著的加速效果。

在 GTX1650 和 RTX3090 上，64 碱基对时都观察到了看似很大的加速比，这有悖直觉，因为 SALoBa 可利用的查询内并行性更少。然而，进一步分析图 6 （a）和（c）可知，问题在于 GASAL2 效率低下，而非 SALoBa 加速效果显著。GASAL2 在开始时的内存初始化开销相对较大，序列长度较短时无法将其摊销。与其在 64 碱基对时同 GASAL2 比较，不如同 NVBIO 比较；后者正确地体现了查询间并行更适用于极短查询序列这一规律。

随着输入变长，使用子束带来的收益会变得微乎其微，而另外两种技术的效果则愈发显著。查询内并行和延迟溢出都能减少全局内存访问。前者减少存储在全局内存中的数据量，后者则通过更好地合并访问，减少冗余访问。因此，在较长的输入范围内，这两种技术成为性能提升的主要驱动力。有趣的是，在 RTX3090 上，查询内并行的效果更明显；而在 GTX1650 上，延迟溢出则更为有效。我们认为，这可以用 RTX3090 具有更高的计算/内存带宽比来解释。RTX3090 的峰值性能为 35.58 太 FLOPS，使用 GDDR6X 时内存带宽为 936.2 GB/s。相比之下，GTX1650 的峰值性能为 2.98 太 FLOPS，内存带宽为 128.1 GB/s。RTX3090 的比值为 38.91 浮点运算/字节，而 GTX1650 为 23.82 浮点运算/字节。这意味着 RTX3090 通常更受内存限制。查询内并行的主要缺点是 CUDA线程利用率低。不过，由于 RTX3090 更受内存带宽限制，即使部分计算资源处于空闲状态，也很可能仍有更多计算资源可用于处理数据。这在一定程度上缓解了 RTX3090 上的利用率问题。

D. 真实数据实验

为了在真实工作负载大小分布下测试 SALoBa 应对工作负载不平衡的能力，我们使用真实世界数据集，通过 BWA-MEM 生成种子 [37] 。该过程需要一条完整的参考基因组序列和许多测序读段，以生成由多个配对组成的种子，供延伸内核处理。

对于参考基因组序列，我们采用了人类基因组组装项目的最新版本，即 Build 38 第 13 次补丁发布版 (GRCh38.p13) [2] ，其长度为 3.1G 碱基对 (bp)。对于输入测序读段，我们使用了从序列读取档案库下载的两个数据集 [36]。数据集 **A** (SRR835433) 来自第二代测序仪 Illumina MiSeq，代表短读长，每条序列长度为 250 碱基对。该数据集包含 830 万条序列，这些序列是从人类基因组中随机读取的片段。每条序列读取两次，因此碱基对总数为 250×8830 万×2=4.1 Gbp。数据集 **B** (SRP091981) 来自第三代测序仪 PacBio RS，代表长读长。该数据集同样包含从人类基因组中随机读取的片段，其中有 8.2 万条长度不一的序列，平均长度约为 2,000 碱基对，总计 182.4 Mbp。BWA-MEM 处理的种子分布 [37] 如图2所示。

结果绘制于图8。图8 (a) 展示了短读长数据集 A 的性能。相较于基线 GASAL2，SALoBa 在 GTX1650 上的最佳加速比为 32.5%，在 RTX3090 上为 20.2%。SOAP3-dp 无法在 GTX1650 上完成该工作负载，因为部分输入的长度超出了它能够处理的范围。SALoBa 测得的加速比略高于图6 (a) 和 (c)，这是因为 SALoBa 受到工作负载不平衡的影响较小。SOAP3-dp 和 NVBIO 的性能不及 GASAL2，而 CUSHAW2-GPU 则比 GASAL2 有一定加速（这得益于 GASAL2 的 GPU端打包内核）。不过，其加速比低于 SALoBa。ADEPT 仅在 RTX3090 上实现了与 SALoBa 相近的加速比，但这是通过将所有中间值存放在共享内存中（不进行全局内存访问）实现的，这从根本上将 ADEPT 的处理能力限制在 1024 碱基对以内。

GTX1650 和 RTX3090 之间一个有趣的差异是，GTX1650 的最佳子线程束大小为 16，而 RTX3090 的最佳子线程束大小为 8（见图 8 (c)）。这一结果是由于

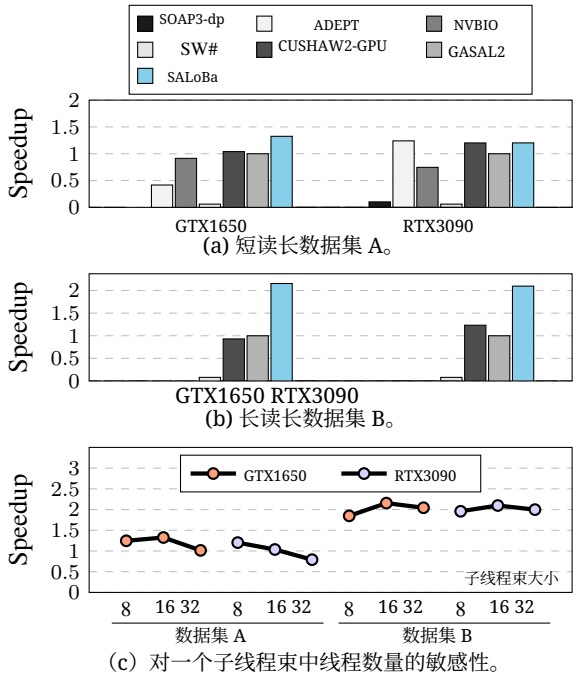

图 8：SALoBa 在真实数据上的性能。加速比以 GASAL2 的性能为基准进行归一化。

各项技术的效果不同。根据图 7，对较短序列应用子 warp 调度的收益为 2。26× （GTX1650）和 2。85× （RTX3090），几何平均值亦如此。在 GTX1650 上，使用子束带来的收益部分被工作负载不平衡的开销抵消，因此最优配置变为每个子线程束包含 16 个线程。

图8（b）展示了数据集 B 上较长读段的性能。在该数据集上，SOAP3-dp、ADEPT 和 NVBIO 因输入长度限制而无法运行。与图 6 （b）和（d）相比，SALoBa 的加速比大幅提升，这是因为工作负载不平衡程度增加，有利于 SALoBa。GTX1650 和 RTX3090 上的最佳加速比均约为 2。1× 每个子束包含 16 个线程时性能最佳，这是兼顾利用查询内并行性与减少工作负载不平衡的最佳选择。

VI. 相关工作

A. 基于 *GPU* 的 *Smith–Waterman* 算法实现

多年来，人们一直在研究如何利用 GPU 加速史密斯–沃特曼算法 [57] 。一些早期基于 OpenGL 库的工作 [40], [41] 解决了将该算法映射到图形流水线上的问题。一种流行的 CUDA 实现见于 CUDASW++ 系列 [42], [47], [49]。他们建议根据预先确定的序列长度阈值，利用查询内和查询间并行性 [42]。GSWABE [46]和 gpu-pairalign [14]也采用了类似的方法。此外，针对蛋白质数据库比对中的全对全模式，还提出了查询剖析优化 [47] 和 CPU-GPU 混合并行 [49] 。然而，这些优化往往过于针对蛋白质比对，难以轻易推广到脱氧核糖核酸等其他领域。

在这方面，CUDAlign 系列 [20], [21], [52], [54]专注于采用查询内并行性的通用 Smith–Waterman 算法。CUDAlign [54] 将动态规划表划分为多个块，并将其分配给线程块。随后，线程块之间通过全局内存中的专用区域进行通信，这些区域称为“水平总线”和“垂直总线”。之后，该算法扩展为支持线性空间算法，从而减少内存占用 [52]。其近期版本支持多 GPU 比对、执行时间预测 [21] 和回溯 [20]。当动态规划表过大时，MultiBP [18], 、MASA [19],和 SW# [35] 等方法会采用块剪枝 [53] ，移除那些不可能获得最优得分的块。

然而，这些算法针对超长序列进行了优化，而这类序列有时甚至无法装入单张 GPU 卡 [33]。对于采用种子-延伸策略的 DNA 比对软件，即使处理较长的 DNA 读段，延伸阶段的输入序列长度也只有数百个碱基左右。因此，依赖查询间并行性的库 [3], [9] 通常能在 DNA 比对中取得更好的表现，正如[13]所示。另一方面，SALoBa 通过采用多项精心设计的优化，在 DNA 比对场景中优于先前的方法。

B. *GPU* 加速的序列比对软件

一些方法针对 GPU 架构设计了适用的端到端 DNA 比对软件。SARU-MAN [15] 和 GPU-RMAP [11] 是较早采用 GPU 执行哈希表查找以进行短读段比对的方法。随着基于后缀树的 Burrows–Wheeler 变换 (BWT) [38] 得到越来越广泛的应用，BarraCUDA [34],、SOAP3 [39] 和 CUSHA W1 [48] 相继推出了 BWT 的 GPU 实现。SOAP3-dp [50],、CUSHAW2-GPU [45], 和 LOGAN [60] 是后来采用种子-延伸策略的方法，而 Arioc [59] 则进一步将该策略扩展到多个 GPU。然而，这些方法往往以牺牲比对质量为代价来提高吞吐量。例如，SOAP3 和 LOGAN 仅支持有限数量的错配，而 CUSHAW2-GPU 将序列打包为每个碱基占两位，并将第五个“N”碱基转换为随机碱基。

尽管上述方法在质量仅有轻微下降的情况下实现了良好的吞吐量，但业界如今更看重质量而非速度。随着 BWA-MEM [37] 等高质量算法成为事实上的标准，GPU 感知型比对软件的实用性受到了限制。一些方法 [13], [29]–[31]着手解决这一问题，设计了通用性足够强的种子延伸内核，使其能够用于 BWA-MEM，并利用查询内性能。然而，后来基于查询间并行的方法超越了这些内核；当前最先进的方法（如 NVBIO [3] 和 GASAL2 [9]）采用的正是这一策略。

最后值得一提的是 cuBLASTP [61], ，这是一种 GPU 加速的蛋白质搜索算法，采用了种子生成与延伸策略的变体。然而，cuBLASTP 仅加速了

种子生成部分，并将延伸部分留给中央处理器，以利用CPU-GPU 混合并行。

C. 现场可编程门阵列/专用集成电路加速

随着人们对基因组学流程的兴趣日益浓厚，越来越多的研究人员开始考虑使用现场可编程门阵列或专用集成电路设计专用加速器。 [62] 提出了一种在 Altera 现场可编程门阵列上实现 Smith–Waterman 算法的方法，随后又有一些研究采用带状算法 [16], [26] 或灵活的脉动阵列 [10], [28]。此外，DRAGEN [56]是 Illumina 推出的基于现场可编程门阵列的平台，可在现场可编程门阵列上实现完整的端到端分析。Darwin [58] 是一种专用集成电路加速器，通过软硬件协同设计并采用带空位过滤，加速全基因组测序。GenAx [22]是一种用于种子生成和延伸的基于自动机的加速器。

尽管这些方法能带来显著的加速效果，但现场可编程门阵列或专用集成电路的设计难度要大得多，而且产品上市需要很长时间。相比之下，基于GPU的比对软件可快速部署，任何配备 GPU 的系统都能轻松采用。

VII. 讨论

A. CUDA 洗牌指令

CUDA 洗牌指令是共享内存的一种替代方案，可直接在寄存器之间交换数据，用于线程间通信。然而，在 SALoBa 的基础上采用基于洗牌指令的通信并未带来额外的加速效果。这一结果与 CUDA 规范一致：洗牌指令的吞吐量几乎与无冲突的共享内存读取相同 [8]。

洗牌指令的一个潜在优势是减少共享内存的使用量。然而，在当前场景中，缓存在共享内存中的数据所占比例可以忽略不计，因此并未带来明显的加速效果。

B. 带状算法

许多种子延伸方法使用带状算法来降低计算负担。由于动态规划表中最佳匹配路径通常靠近对角线，因此只处理预定宽度范围内的单元格（即带状区域）通常就能获得足够高质量的解。然而，大多数基于图形处理器的延伸库并未利用这一方法。

一个问题是，每个查询序列的带宽通常各不相同，这会加剧负载均衡问题。不过，我们认为，对于更长的读段，带状算法将带来更多优势，并可用于提升性能。

C. 多个GPU

使用多个图形处理器通常很有益，尤其是在单台机器配备多个图形处理器时。我们认为，只需将查询划分为数量相等的部分，并将其分配给多个GPU，就能轻松扩展这项工作。这种策略的一个潜在缺点是GPU之间可能出现负载不均衡。不过，与线程级不平衡问题相比，这种代价很小。如果这种情况变得严重，可以通过动态分配或采用近似排序进行预处理来解决。

VIII. 结论

我们提出了 SALoBa，这是一种新型的基于 GPU 的种子延伸库，其性能优于现有工作，达到了最先进的水平。实验表明，该软件在现代设备上表现良好，并且随着新一代 GPU 的推出，性能提升幅度也随之增大。我们认为，在序列比对耗时对致命疾病诊断至关重要的领域，这款软件将大有用处。

致谢

本研究得到韩国政府（科学技术信息通信部）资助的韩国国家研究基金会（NRF）项目（2022R1C1C1008131、2022R1C1C1011307）以及韩国政府（科学技术信息通信部）资助的信息通信技术规划与评估院（IITP）项目（2020-0-01361，人工智能研究生院项目（延世大学）；2021-0-00853，PIM编程软件平台开发）的支持。

参考文献

[1] “DNA测序成本：数据。” [在线]。网址：https://www.genome.gov/about-genomics/fact-sheets/DNA-Sequencing-Costs-Data [2] “GRCh38.p13。” [在线]。网址：https://www.ncbi.nlm.nih.gov/assembly/GCF 000001405.39/ [3] “NVBIO。” [在线]。网址：https://developer.nvidia.com/nvbio [4] “NVIDIA Tesla P100。” [在线]。网址：https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf [5] “NVIDIA Tesla V100 GPU架构。” [在- 线]。网址：https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf [6] “NVIDIA Turing GPU架构。” [在-线]。网址：https://images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf [7] “读段模拟器。” [在线]。网址：https://github.com/lh3/wgsim [8] “原生算术指令的吞吐量。” [在线]。网址：https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#arithmetic-instructions throughput-native-arithmetic-instructions [9] N. Ahmed, J. L´evy, S. Ren, H. Mushtaq, K. Bertels 和 Z. Al-Ars，“GASAL2：面向高吞吐量下一代测序数据的 GPU 加速序列比对库”，BMC Bioinformatics，第 20 卷，20，2019 年 1，1号，第 – 页。[10] N. Ahmed, V.-M. Sima, E. Houtgast, K. Bertels 和 Z. Al-Ars，“BWA-MEM DNA比对算法的异构软硬件加速”，载于国际计算机辅助设计会议。IEEE，2015。[11] A. M. Aji, L. Zhang 和 W.-c. Feng，“GPU-RMAP：在图形处理器上加速短读段定位”，载于计算科学与工程会议。IEEE，2010。[12] M. Alser, Z. Bing¨ol, D. S. Cali, J. Kim, S. Ghose, C. Alkan 和 O. Mutlu，“加速基因组分析：持续探索之旅入门”，IEEE Micro，第 40 卷，第 5 期，第 65–75 页，2020。[13] M. G. Awan, J. Deslippe, A. Buluc, O. Selvitopi, S. Hofmeyr, L. Oliker 和 K. Yelick，“ADEPT：面向 GPU 架构的领域无关序列比对策略”，BMC Bioinformatics，第 21 卷，第 1 期，第 1–29 页，2020。[14] J. Blazewicz, W. Frohmberg, M. Kierzynka, E. Pesch 和 P. Woj- ciechowski，“采用高效回溯程序、运行于多个 GPU 上的蛋白质比对算法 ”，BMC Bioinformatics，第 12 卷，第 1 期，第 1–17 页，2011。[15] J. Blom, T. Jakobi, D. Doppmeier, S. Jaenicke, J. Kalinowski, J. Stoye 和 A. Goesmann，“使用图形处理器编程对微生物基因组进行精确且完整的短读段比对”，生物信息学，第 27 卷，第 10 期，第 1351–1358 页，2011。[16] P. Chen, C. Wang, X. Li 和 X. Zhou，“采用循环脉动阵列的带状Smith-Waterman算法硬件加速”，载于现场可编程技术会议。IEEE，2013。

[17] J. Choquette, E. Lee, R. Krashinsky, V. Balan 和 B. Khailany，“A100 数据中心 GPU 与 Ampere 架构”，载于 ISSCC，第 64 卷。IEEE，2021 年，第 48–50 页。[18] M. A. C. de Figueiredojr, J. P. Navarro, E. F. de Oliveira Sandes, G. Teodoro 和 A. C. M. Melo，“采用剪枝，在同构和异构 GPU 平台上对长 DNA 序列进行并行细粒度比较”，IEEE Transactions on Parallel and Distributed Systems，2021。[19] E. F. De O. Sandes, G. Miranda, X. Martorell, E. Ayguade, G. Teodoro 和 A. C. De Melo，“MASA：用于序列比对器的多平台架构与块剪枝”，ACM 并行计算汇刊，第 2 卷，第 4 期，第 1–31 页，2016。[20] E. F. de Oliveira Sandes, G. Miranda, X. Martorell, E. Ayguade, G. Teodoro 和 A. C. M. Melo，“CUDAlign 4.0：用于 GPU 集群全染色体精确比对的增量式推测回溯”，IEEE Transactions on Parallel and Distributed Systems，第 27 卷，第 10 期，第 2838–2850 页，2016。[21] F. d. O. Edans, G. Miranda, A. C. de Melo, X. Martorell 和 E. Ayguad´e，“CUDAlign 3.0：大型 GPU 集群上的并行生物序列比较”，载于 CCGrid。IEEE，2014。[22] D. Fujiki, A. Subramaniyan, T. Zhang, Y. Zeng, R. Das, D. Blaauw 和 S. Narayanasamy，“GenAx：基因组测序加速器”，载于 ISCA。IEEE，2018。[23] D. Fujiki, S. Wu, N. Ozog, K. Goliya, D. Blaauw, S. Narayanasamy 和 R. Das，“SeedEx：在次最小空间中实现最优比对的基因组测序加速器”，载于 MICRO。IEEE，2020。[24] A. Goyal, H. J. Kwon, K. Lee, R. Garg, S. Y. Yun, Y. H. Kim, S. Lee 和 M. S. Lee，“使用 DRAGEN™ bio-IT 处理器进行精准医疗的新一代人类基因组测序数据超快速处理”，《开放遗传学杂志》，第 7 卷，第 1 期，第 9–19 页，2017。[25] T. J. Ham, D. Bruns-Smith, B. Sweeney, Y. Lee, S. H. Seo, U. G. Song, Y. H. Oh, K. Asanovic, J. W. Lee 和 L. W. Wills，“Genesis：基因组数据分析硬件加速框架”，载于 ISCA。IEEE，2020。[26] B. Harris, A. C. Jacob, J. M. Lancaster, J. Buhler 和 R. D. Cham- berlain，“用于 Mercury BLASTP 的带状 Smith-Waterman FPGA 加速器”，载于 FPL。IEEE，2007。[27] S. Hong, S. K. Kim, T. Oguntebi 和 K. Olukotun，“以最大线程束规模加速 CUDA 图算法”，载于 PPoPP，2011。[28] E. J. Houtgast, V.-M. Sima, K. Bertels 和 Z. Al-Ars，“用于加速 BWA-MEM 基因组定位算法的基于 FPGA 的脉动阵列”，载于 SAMOS。IEEE，2015。[29] ——，“采用自适应负载均衡的 GPU 加速 BWA-MEM 基因组定位算法”，载于 ARCS。Springer，2016。[30] ——，“面向更长读长的 BWA-MEM 基因组短读长定位硬件加速”，《计算生物学与化学》，第 75 卷，第 54–64 页，2018。[31] E. J. Houtgast, V. Sima, K. Bertels 和 Z. Al-Ars，“使用 bwamem 高效实现 GPU 加速的基因组短读长定位”，ACM SIGARCH《计算机体系结构新闻》，第 44 卷，第 4 期，第 38–43 页，2017。[32] M. Khairy, J. Akshay, T. Aamodt 和 T. G. Rogers，“通过精确建模探索现代 GPU 内存系统设计挑战”，arXiv 预印本 arXiv:1810.07269，2018。[33] A. Khajeh-Saeed, S. Poole 和 J. B. Perot，“使用单个和多个图形处理器加速 Smith–Waterman算法”，《计算物理学杂志》，第 229 卷，4258，2010 年 11，4247号，第 – 页。[34] P. Klus, S. Lam, D. Lyberg, M. S. Cheung, G. Pullan, I. McFarlane, G. S. Yeo 和 B. Y. Lam，“BarraCUDA——一种使用图形处理器的快速短读长序列比对器”，BMC Research Notes，第 5 卷，第 1 期，第 1–7 页，2012。[35] M. Korpar 和 M. ˇSiki´c，“SW#——GPU 加速的基因组规模精确比对”，生物信息学，第 29 卷，第 19 期，第 2494–2495 页，2013。[36] R. Leinonen, H. Sugawara, M. Shumway 和 I. N. S. D. Collaboration，“序列读取档案库”，《核酸研究》，第 39 卷，增刊 1，第 D19–D21 页，2010。[37] H. Li，“使用 BWA-MEM 对测序读段、克隆序列和组装重叠群进行比对”，arXiv 预印本 arXiv:1303.3997，2013。[38] H. Li 和 R. Durbin，“通过 Burrows–Wheeler 变换实现快速、准确的短读段比对”，生物信息学，第 25 卷，第 14 期，第 1754–1760 页，2009。[39] C.-M. Liu, T. Wong, E. Wu, R. Luo, S.-M. Yiu, Y. Li, B. Wang, C. Yu, X. Chu, K. Zhao 等，“SOAP3：超快速、基于 GPU 的并行短读长比对工具”，《生物信息学》，第 28 卷，第 6 期，第 878–879 页，2012 年。[40] W. Liu、B. Schmidt、G. Voss、A. Schroder 和 W. Muller-Wittig，“GPU 上的生物序列数据库扫描”，载于 IPDPS。IEEE，2006 年。[41] Y. Liu、W. Huang、J. Johnson 和 S. Vaidya，“GPU 加速的史密斯-沃特曼算法”，载于 ICCS。Springer，2006 年。[42] Y. Liu、D. L. Maskell 和 B. Schmidt，“CUDASW++：针对支持 CUDA 的图形处理器优化史密斯-沃特曼序列数据库搜索”，《BMC Research Notes》，第 2 卷，第 1 期，第 1–10 页，2009 年。[43] Y. Liu、B. Popp 和 B. Schmidt，“使用 CUSHAW2 实现高速且准确的颜色空间短读长比对”，arXiv 预印本 arXiv: . . 4766, 2013 1304 ”[44] ——，“CUSHAW3：通过混合种子生成实现灵敏且准确的碱基空间和颜色空间短读长比对”，《PLoS ONE》，第 9 卷，第 1 期，第 e86869 页，2014 年。[45] Y. Liu 和 B. Schmidt，“CUSHAW2-GPU：利用 GPU 计算加速带缺口短读长比对”，《IEEE Design & Test》，第 31 卷，第 1 期，第 31–39 页，2013 年。[46] ——，“GSWABE：通过最优比对结果检索加速 GPU 序列比对，适用于短脱氧核糖核酸序列”，《Concurrency and Computation: Practice and Experience》，第 27 卷，第 4 期，第 958–972 页，2015 年。[47] Y. Liu、B. Schmidt 和 D. L. Maskell，“CUDASW++ 2.0：基于单指令多线程和虚拟化SIMD抽象，在支持 CUDA 的 GPU 上增强史密斯-沃特曼蛋白质数据库搜索”，《BMC Research Notes》，第 3 卷，第 1 期，第 1–12 页，2010 年。[48] ——，“CUSHAW：一种兼容 CUDA、基于 Burrows–Wheeler 变换、面向大型基因组的短读长比对器”，《生物信息学》，第 28 卷，第 14 期，第 1830–1837 页，2012 年。[49] Y. Liu、A. Wirawan 和 B. Schmidt，“CUDASW++ 3.0：通过结合中央处理器和 GPU 的单指令多数据指令，加速史密斯-沃特曼蛋白质数据库搜索”，《BMC Bioinformatics》，第 14 卷，第 1 期，第 1–10 页，2013 年。[50] R. Luo、T. Wong、J. Zhu、C.-M. Liu、X. Zhu、E. Wu、L.-K. Lee、H. Lin、W. Zhu、D. W. Cheung 等，“SOAP3-dp：快速、准确且灵敏的基于 GPU 的短读长比对器”，《PLoS ONE》，第 8 卷，第 5 期，第 e65632 页，2013 年。[51] S. B. Needleman 和 C. D. Wunsch，“一种适用于搜索两种蛋白质氨基酸序列相似性的方法”，《Journal of Molecular Biology》，第 48 卷，第 3 期，第 443–453 页，1970 年。[52] E. F. d. O. Sandes 和 A. C. M. de Melo，“利用 GPU 在线性空间中比对超长序列”，载于 IPDPS。IEEE，2011 年。[53] ——，“使用 GPU 和优化方法检索百万碱基级生物序列的史密斯-沃特曼比对结果”，《IEEE Transactions on Parallel and Distributed Systems》，第 24 卷，第 5 期，第 1009–1021 页，2012 年。[54] E. F. O. Sandes 和 A. C. M. de Melo，“CUDAlign：使用 GPU 加速百万碱基级基因组序列的比较”，载于 PPoPP，2010 年。[55] E. E. Schadt、S. Turner 和 A. Kasarskis，“第三代测序的一扇窗口”，《Human Molecular Genetics》，第 19 卷，第 R2 期，第 R227–R240 页，2010 年。[56] K. Scheffler、S. Kim、V. Jain、J. Yuan、W. Sherman、T. O’Connell、E. Ojard、L. Murray、R. Mehio 和 S. Catreux，“利用 DRAGEN 平台提高体细胞全基因组小变异检测的准确性”，2020 年。[57] T. F. Smith、M. S. Waterman 等，“常见分子子序列的识别”，《Journal of Molecular Biology》，第 147 卷，第 1 期，第 195–197 页，1981 年。[58] Y. Turakhia、G. Bejerano 和 W. J. Dally，“Darwin：一种基因组学协处理器，可将长读长组装速度最高提升 15,000 倍”，载于 ASPLOS。ACM，2018 年。[59] R. Wilton 和 A. S. Szalay，“Arioc：在多个 GPU 上实现高并发短读长比对”，《PLoS Computational Biology》，第 16 卷，第 11 期，第 e1008383 页，2020 年。[60] A. Zeni、G. Guidi、M. Ellis、N. Ding、M. D. Santambrogio、S. Hofmeyr、A. Buluc、L. Oliker 和 K. Yelick，“LOGAN：高性能、基于 GPU 的 x-drop 长读长比对”，载于 IPDPS。IEEE，2020 年。[61] J. Zhang、H. Wang 和 W.-c. Feng，“cuBLASTP：在中央处理器+GPU 上对蛋白质序列搜索进行细粒度并行化”，《IEEE/ACM Transactions on Computational Biology and Bioinformatics》，第 14 卷，第 4 期，第 830–843 页，2015 年。[62] P. Zhang、G. Tan 和 G. R. Gao，“在可重构超级计算平台上实现史密斯-沃特曼算法”，载于《高性能可重构计算技术与应用国际研讨会》，2007 年。