VEGA：使用 RVV 扩展进行基因组比对的可扩展软硬件协同设计

Weihong Xu*‡、Shijie Xu†、Qunyou Liu†、Giovanni Ansaloni†、Cheng Zhuo* 和 David Atienza†
*浙江大学集成电路学院，中国杭州
†嵌入式系统实验室（ESL），洛桑联邦理工学院（EPFL），洛桑，瑞士；电子邮箱：{wh.xu、czhuo}@zju.edu.cn，{shijie.xu、qunyou.liu、giovanni.ansaloni、david.atienza}@epfl.ch

arXiv:2611.28741v1 [eess.SP] 21 Nov 2026

摘要—基于动态规划（DP）的基因组序列分析是计算基因组学的基础，但随着基因组数据量持续增长并不断超越摩尔定律，其计算开销仍然十分巨大。现有硬件加速器通常需要牺牲可编程性，以及在不同比对算法之间的可扩展性。本文提出 VEGA，一种使用 RISC-V 向量扩展加速基因组比对的可扩展软硬件协同设计。VEGA 引入了专用向量指令和三项关键创新：能够最大限度减少片外存储器访问的前沿驻留数据流、能够最大限度提升 SIMD 通道利用率的序列间并行，以及能够加速核心比对操作的定制 ISA 扩展。评估结果表明，与 Ara2-RVV 向量基线相比，VEGA 在编辑距离和 Smith–Waterman 算法上分别实现最高达 15.1× 和 9.7× 的加速比，并在 10 千碱基对序列上持续获得 10.2× 和 6.7×的性能提升。综合后结果表明，相较于标准 RISC-V 核心，其面积开销仅为1.0% ，同时单位面积性能效率可媲美专用集成电路。因此，VEGA 在基因组序列比对的效率、可编程性和可扩展性方面实现了均衡提升。

索引词—RISC-V、向量扩展、基因组比对、向量处理、硬件加速器。

I. 引言

基因组测序是现代生物学和医学的基础，推动了个性化医疗 [1] 和病原体检测 [2]的发展。下一代测序（NGS）技术 [3] 的兴起产生了拍字节级数据集，且读段长度不断增加，对高效的计算分析提出了要求。动态规划（DP）算法，如 Smith–Waterman–Gotoh 算法 [4], 、Needleman–Wunsch 算法 [5],和编辑距离 [6], ，在比对准确性方面仍是金标准，但受限于二次复杂度。带状比对 [7–9]和波前比对 [10]等优化方法通过将计算限制在对角线上来提高效率；然而，在传统中央处理器上，它们仍面临严峻的运行时间挑战。

随着基因组数据的增长速度超过摩尔定律，研究人员正致力于开发更快速、更高效的硬件加速器，以应对日益复杂的比对工作负载。近期利用 CPU [8–14], GPU [15–17], 现场可编程门阵列 [18, 19],专用集成电路 [20–24],和近存处理（NMP）[25–27]的进展已取得显著加速比。然而，大多数固定功能加速器都针对特定算法，而基于动态规划的比对方法[8–10, 13, 14] 种类繁多，每种实现仍需进行大量硬件重新设计。

近期的加速器 [11, 12, 20, 22] 旨在实现基因组比对中性能与可编程性的平衡。GenDP [22] 采用基于ASIC的可编程单元来支持多种DP 算法，但需要面向特定算法的硬件。GMX [12] 通过基因组专用指令扩展开放的RISC-V指令集架构 [28] ，提供了灵活性，但加速比有限。WFAsic [20] 在RISC-V片上系统上实现了波前比对的高性能和高能效，但缺乏通用性。QUETZAL [11] 在A64FX片上系统 [30] 上使用 ARM SVE2 [29]来提高可编程性，但仍受限于封闭的ARM指令集架构。总体而言，基于ASIC的设计能够提供更高的速度，但灵活性较差；而指令集架构扩展可以提高可编程性，但性能增益较为适中。

RISC-V向量扩展（RVV）1.0 扩展 [31] 解决了现有可编程加速器的关键局限。其开放的指令集架构支持领域专用向量指令，同时保持软件兼容性，这比封闭指令集架构更具优势。RVV 的动态SIMD长度也更适合比对算法不规则的计算模式。尽管 RVV 已广泛应用于机器学习 [32],高性能计算 [33, 34], 和嵌入式系统 [35],，但在基因组学领域仍鲜有探索 [36]。它为结合硬件效率与现代基因组应用所需的算法灵活性提供了独特机遇。

借助 RVV指令集架构，我们提出了 VEGA，这是一款利用专用扩展进行基因组比对的加速器。VEGA 基于开源RVV处理器 [37], 构建，增强了面向比对工作负载的向量处理。性能分析（第 II-D 节）表明，现有 RVV 实现存在内存访问效率低下和指令开销过大的问题。为克服这些瓶颈，VEGA 引入了有针对性的优化方案，在保持完全可编程性的同时消除关键的向量处理低效问题，从而在各种比对算法中实现显著的性能提升。本文的主要贡献总结如下：

•VEGA 通过集成三种数据流优化来解决 RVV 的低效问题：前沿驻留（FFS）、序列间并行（ISP）以及融合加法-最小值操作。FFS 将重复的得分加载转换为片上数据传输，从而最大限度地减少数据移动；ISP 则将 SIMD通道 划分为多个前沿区域，以便

‡本工作由Weihong Xu在瑞士洛桑EPFL嵌入式系统实验室（ESL）担任博士后研究员期间完成。

同时处理多个序列对，最大限度地提高通道利用率。融合加法-最小值指令进一步通过合并连续操作来降低算术开销。总体而言，这些优化提升了吞吐量、效率和可编程性。

- VEGA 通过扩展 RVV 指令集架构，引入了专为基因组比对设计的领域专用指令。这些扩展仅带来 1.0% 的面积开销，同时显著提升了基因组工作负载的计算效率。

- 评估结果表明，对于短读段，与 Ara2-RVV 基线相比，VEGA 在编辑距离和 Smith–Waterman 算法上分别实现了 15.1× 和 9.7× 的加速比；对于 10 kbp 序列，VEGA 仍保持显著优势（加速比分别为 10.2× 和 6.7× ）。这种可扩展性使 VEGA 能够适用于现代基因组应用中的各种序列长度。

II. 背景与动机

A. 基于动态规划的基因组比对

基因组序列比对是生物信息学中的核心问题，用于比较 DNA、RNA 和蛋白质序列。动态规划（DP）算法构成了其基础：算法通过计算一个矩阵，为序列前缀中的每个位置存储最优比对得分。常用的变体包括用于带仿射间隙的局部比对的 Smith–Waterman–Gotoh（SWG）[4]、用于全局比对的 Needleman–Wunsch（NW）[5]，以及用于求得最少编辑转换次数的编辑距离算法 [6]。

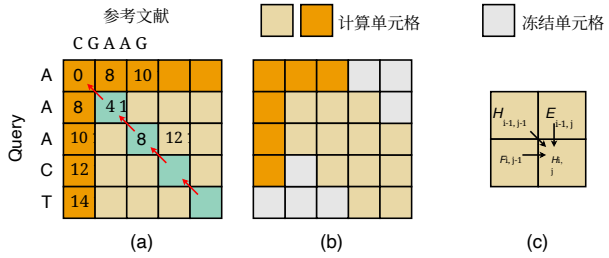


图 1：(a) 得分计算与回溯，(b) 带状比对，以及 (c) DP比对的数据依赖图。

图 1-(a) 展示了参考序列“CGAA”和查询序列“AACA”之间的 SWG DP 得分矩阵示例。给定两个序列 $Q = q_1 q_2 \dots q_m$ 和 $R = r_1 r_2 \dots r_n$、匹配 / 错配评分函数 $s(\cdot,\ \cdot)$、缺口开启惩罚 o 以及缺口延伸惩罚 e，仿射间隙SWG算法的递推关系如下：

$$E_{i,j} = \min \begin{cases} H_{i-1,j} + o + e \\ E_{i-1,j} + e \end{cases} \qquad F_{i,j} = \min \begin{cases} H_{i,j-1} + o + e \\ F_{i,j-1} + e \end{cases}$$

$$H_{i,j} = \min \begin{cases} E_{i,j} \\ F_{i,j} \\ H_{i-1,j-1} + s(q_i, r_j) \end{cases} \tag{1}$$

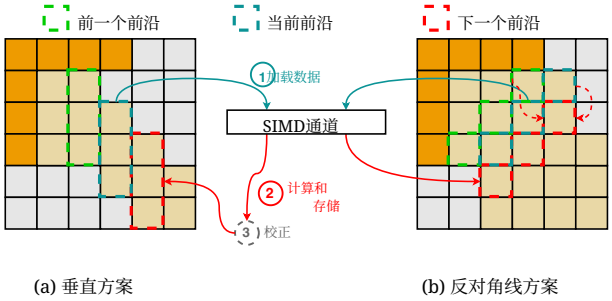


图 2：带状 DP比对的两种向量化数据流对比：(a) 垂直方案和 (b)反对角线方案。

其中，$H(i,\ j)$ 表示以位置 $i,\ j$ 结束的最优比对得分。$E(i,\ j)$ 和 $F(i,\ j)$ 分别表示以查询序列 (Q) 或参考序列 (R) 中的缺口结束的比对得分。$s(q_i,\ r_j)$ 是替换评分函数，通常对 DNA/RNA 序列采用匹配/错配方案。基于 DP 的算法首先根据这些递归方程填充矩阵。计算完所有单元格的值后，通过从得分最高的单元格（对于局部比对）或最后一个单元格（对于全局比对）执行回溯，重构最优得分路径，从而确定比对路径。

尽管基于 DP 的算法能够保证获得最优解，但其二次 $\mathcal{O}(mn)$计算复杂度和内存占用是现代基因组学工作负载的主要瓶颈。为提高计算效率，人们开发了各种优化技术。图 1-(b) 展示了带状比对 [7], ，它将计算限制在 DP 矩阵中的一个反对角线区域内；对于相似序列，计算复杂度因此从 $\mathcal{O}(mn)$降至 $\mathcal{O}(\mathrm{BW} \cdot n)$，其中 BW 表示带宽。

波前比对算法(WFA) [10] 通过扩展匹配碱基的对角线（波前）来构建比对路径，从而优化 DP，达到 $\mathcal{O}(s^2)$复杂度。该算法能加速处理得分较低的短而准确的读段 s，但先前研究[9, 13] 指出，在高错误率序列上的速度会显著下降，因此难以针对易出错数据实现高效处理。

B. 向量化基因组比对

在这项工作中，我们聚焦于带状 DP比对[7–9, 13, 14]，因为它在长读长测序中具有可扩展性和并行性。图 2 对比了现代比对器 [8, 9, 13, 14] 和加速器 [11, 15, 19, 21, 22]中常用的两种向量化带状比对（垂直方案和反对角线方案）。两者都会将数据加载到 SIMD 通道并更新得分，但前沿组织方式不同。**垂直方案**易于映射到内存中，但会受到垂直依赖的影响（$E_{i,\ j}$ 取决于$E_{i-1,\ j}$）。先前的前缀扫描 [13] 和 F 循环 [9, 38]优化只能部分缓解这一问题。**反对角线方案**消除了这类依赖，从而实现完全并行 [11, 15, 19]，但需要进行字符串反转 [8, 13] 并采用经过修改的内存布局。VEGA 采用反对角线方案，以充分利用 RVV 的高并行性，

引入经过优化的数据流和内存策略，以克服这些挑战。

C. RISC-V 向量扩展

RISC-V 向量（RVV）扩展 [31] 为向量处理提供了灵活且可扩展的框架。与 x86 AVX 或 ARM NEON 等固定宽度 SIMD 架构不同， [39],RVV 1.0 引入了一种与向量长度无关的模型，可确保软件在向量长度不同的硬件之间具有可移植性。可配置的 VLEN 和通过 vsetvli 在运行时调节的参数，使其能够高效适配各种工作负载。尽管 RVV 已广泛应用于机器学习和高性能计算， [32–35]，但其在基因组序列比对中的潜力仍未得到探索。GMX [12] 和 WFAsic 等现有加速器 [20] 依赖 RISC-V 标量指令，忽略了 RVV 并行性。本文通过提出一种基于 RVV 的加速框架，结合专用向量扩展，实现高效的基因组计算，从而弥补了这一空白。

清单1：SIMD 向量化带状比对的伪代码。

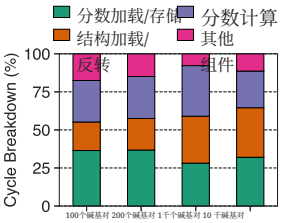
```
n_前沿  =m+n+1;
对于(h=2;h<n_前沿;++h) {
  对于(v=1;v<带宽+1;v+=16) {
    /* 计算新的 E 前沿 */
    v_prev_E  = load_得分(E);
    v_prev_H  = load_得分(H);
    v_prev_E  = saddu_u16(v_prev_E, e);
    v_prev_H {v  = saddu_u16(v_prev_H, o+e);
    v_E  = min_u16(v_prev_E, v_prev_H);
    存储_得分(E, v_E);
    /* 计算新的 F 前沿 */
    . . .
    /* 加载参考序列和 查询字符串 */
    load_str_rev(str_ref, v_ref); //加载和反转
    load_str(str_query, v_query);
    v_得分  = 向量混合(不匹配, 匹配, 相等比较(vquery,
        vref))
    /* 计算新的 H 前沿 */
    v_prev_H  = load_得分(H);
    v_H  = saddu_u16(v_prev_H, v_得分);
    v_H {v{2}min{v{3}u16(v_E, v_H);
    v_H {v{2}分钟_u16(v_F, v_H);
    存储_得分(H, v_H);
  }
}
```

D. 动机分析

为了评估采用反对角线数据流的带状动态规划比对，我们分别使用 AVX2 和 RVV 实现并分析了两个向量化 C 内核（清单1）。实验在一台 16 线程 AMD Ryzen 7 9700X [40] 和开源 Ara2 RVV1.0 处理器 [37]上进行。更多设置详情见第 IV 节。我们将性能分析结果总结为以下几点：

挑战 1：指令与 I/O 开销。如图3-(a)所示，超过 50% 的 CPU 周期用于得分计算和字符串加载/存储的内存操作，这些操作经常导致计算停滞并降低算术单元利用率，使得实际得分计算仅占约25% 的周期。

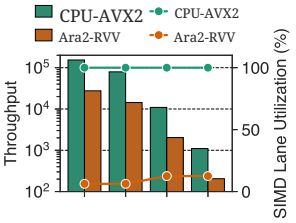
挑战 2：有限的 SIMD 宽度。 如图 3-(b) 所示，AVX2 和 ARM SVE2 的 SIMD 宽度较窄，与现代比对器中使用的 16–1024 带宽存在巨大差距



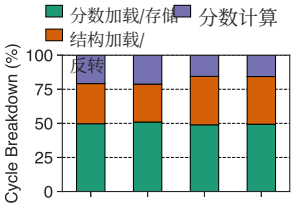
（a）CPU 上 AVX2 内核的周期分解

ISA	SIMD 宽度
AVX2 [41]	256b
ARM SVE2 [29]	128-2048b
RISC-V V 1.0 [31]	灵活
算法	**带宽**
ABEA [19]	100
BSAlign [9]	32-1024
块比对器 [13]	32-128
libgaba [8]	16-256
minimap2 [14]	500-1000

(b) 现有指令集架构和带状算法



（c）AVX2 和 RVV 内核的单核性能



（d）RVV 内核在 Ara2 [37] 上的周期分解

图3：向量化带状 SWG 的性能分析结果。

。对于 AVX2 这类 256位宽的 SIMD格式，清单1中的内层循环必须运行 $16 \cdot$带宽 256 次才能更新一个 16b 的前沿向量，从而造成过高的指令开销并限制吞吐量。因此，需要一种可扩展性更强、宽度更大的 SIMD指令集架构。

挑战3：RVV 效率低下。为克服指令开销和有限的 SIMD 宽度，我们利用 RVV 可扩展的 SIMD 设计，其中 4096b 的 VLEN 可对 256 个元素执行一次加法或最小值操作。我们在 Ara2 处理器 [37]，上实现了 RVV 1.0 内核 [31] ，但图 3-(c)显示，尽管通道利用率接近满载，Ara2-RVV 的性能仍不及 x86 AVX2。瓶颈源于较低的 SIMD通道使用率（BW =16–32 仅使用了 4096b 的 12.5–25%）以及较高的内存开销（如图3-(d)所示，评分输入输出占 50–55% 的周期，字符串处理占 25–30%，而计算仅占 15–20%）。

III. 提出的VEGA加速器

A. 概述

图4展示了所提出的用于基因组比对加速的VEGA软硬件协同设计。VEGA将RISC-V标量和向量扩展与定制化软件相结合，高效实现所提出的基于RVV的扩展（第III-C节），同时保持与RISC-V的完全兼容性。它为多种比对算法提供可扩展的多核解决方案，并通过以下两项关键设计策略兼顾可编程性与高性能：

VEGA软件 包括基因组优化数据流和内建函数，可在几乎无需修改代码的情况下充分利用 RVV 并行性。第 III-B 节中的 DP 比对内核采用三种技术进行优化：前沿驻留（FFS）

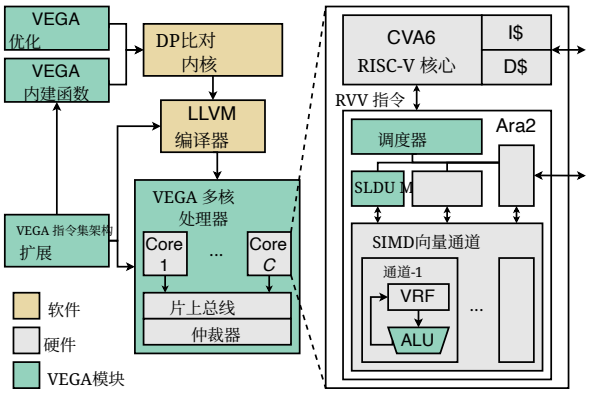


图4：VEGA 框架的总体架构。

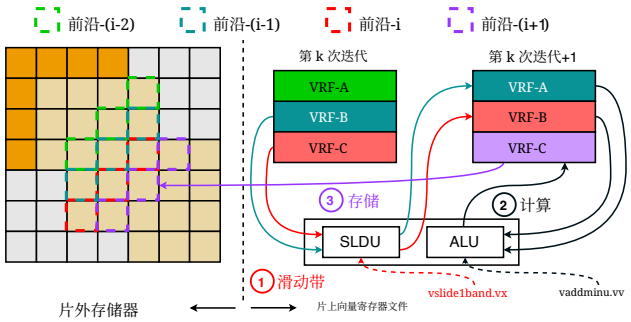


图5：VEGA 中的前沿驻留（FFS）数据流。

数据流、序列间并行以及融合加法–最小值运算。随后，这些内核通过支持 VEGA 指令集架构扩展的 LLVM 工具链进行编译，并在 VEGA 多核处理器上执行。

VEGA 硬件基于 Ara2 [37], ——一种先进的 RVV 1.0 处理器 [31]——为 CVA6 RISC-V 核心 [42] 添加了向量功能。其关键组件包括负责内存传输的向量加载存储单元 VLSU、负责向量重排的 SLDU、负责掩码处理的掩码单元 MASKU，以及 L SIMD通道（向量长度 VLEN 最高可配置为 4096b），每个通道都配有本地向量寄存器文件 VRF 和专用算术逻辑单元 ALU。为克服 RVV 在基因组比对方面的局限性，我们增强了 Ara2 的调度器、SLDU 和 ALU。调度器经过扩展以支持 VEGA 指令集架构控制，SLDU 经过优化以实现快速字符串加载和反转，ALU 则升级以支持融合加法-最小值运算。这些针对性改动减少了指令和数据冗余，最大限度地发挥 RVV 并行性，以实现 DP比对。微架构细节见 III-C节。

B. 数据流优化策略

前沿驻留数据流。第 II-D 节表明，在 RVV 硬件上，DP比对的主要瓶颈是反复加载得分向量所带来的高数据移动开销。然而，带状 DP 比对在反对角线方向上存在数据复用机会 [8, 11], ，因为图 5 显示，每个新前沿仅取决于前两个前沿。以往的 CPU 实现 [8, 11]未能利用这一局部性，因为有限的 SIMD 宽度要求将数据传输到缓存或片外存储器，以处理较宽的带。

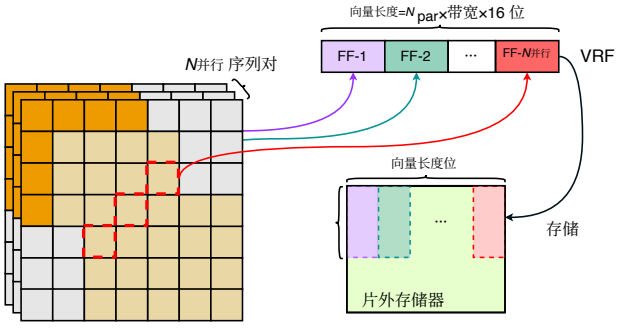


图 6：序列间并行（ISP）及其内存布局。

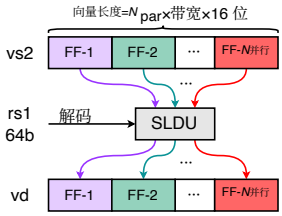
借助 RVV 宽大的向量寄存器文件（VRF），VEGA 在图 5 中提出了前沿驻留（ FFS ）数据流，通过将中间结果保留在向量寄存器文件中，最大限度地降低数据移动开销。FFS 数据流包括三个步骤： 先根据下一个前沿 $i-1$ 的移动方向，对存储在 VRF-B 中的前一个前沿和存储在 VRF-C 中的当前前沿 i进行移位 $i+1$。这通过 SLDU 执行的滑动操作实现。前一个和当前前沿的向量数据不会传输到片外，而是存入新的向量寄存器文件，即， VRF-A 和 VRF-B，供后续计算使用。 随后，将移位后的前沿取出并加载到 SIMD 算术逻辑单元，以更新下一个前沿 $i+1$，并将其写回 VRF-C。更新后的前沿向量 $i+1$ 则写回片外存储器，因为回溯（TB）过程需要用到它。每次迭代都会重复执行 至 。FFS 大幅降低了外部带宽使用量，从而实现高效的流水线处理和较高的 SIMD 利用率。与朴素内核相比，得分加载次数从 $N_{\text{forefront}}$ 降至每次比对仅一次。

序列间并行。 此前的 FFS 数据流实现了前沿级并行，但无法充分利用 RVV 宽大的 SIMD 通道。为此，VEGA 在图 6 中引入了序列间并行（ISP），通过并发处理 N_{par} 序列对来补充 FFS，其中每个序列对占用 VLEN 位 SIMD 通道的一部分。向量长度与带宽之间的关系为 VLEN $= N_{\text{par}} \times \text{BW} \times 16$, 因此最多可同时处理 $N_{\text{par}} = {}_{\text{VLEN BW}\times 16}$ 个具有 16 位精度的序列。不同于将独立比对分配给~~不同~~ SIMD 单元的 Parasail [43]和 swipe [44], , VEGA 的 ISP 保留了 FFS 的数据局部性，并最大限度地减少了片外带宽需求。使用多个前沿填充 SIMD 通道可降低指令开销——每次更新 N_{par} 个前沿只需一条指令——并提高吞吐量，即使对于窄带比对也是如此。

ISP 数据流还要求采用图 6 所示的合并内存布局。动态规划矩阵存储在连续的 VLEN $\times N_{\text{forefront}}$ 数组中，每行包含 N_{par} 个前沿。比对期间，（1）初始化前 BW 行；（2）将前两个前沿加载到向量寄存器文件中进行计算；（3）仅写回更新后的前沿。在片上处理带移位和填充，可确保按 VLEN 位对齐传输，最大限度地提高内存带宽并避免非对齐访问惩罚。

表 I：VEGA 指令集架构扩展

指令	功能码3	描述
vslide1带.vx vd, vs2, rs1	OPMVX	带滑动和字符串加载
vrevband.vx vd, vs2, rs1	OPMVX	字符串带反转
vaddminu.vx vd, vs2, rs1 vaddminu.vv vd, vs2, vs1	OPIVX OPIVV	融合加-最小值操作



(a) 带滑动操作

代码 - 4b	SLDU行为
0x0	保持带静止
0x1-0x7	带向上滑动 1 位 + 填充
0x8-0xf	带向下滑动 1 位 + 填充

(b) rs1 操作数的编码表

图 7：(a) 使用 vslide1band.vx 的带滑动操作。(b) SLDU 行为编码表。

融合加法-最小值运算。DP比对高度依赖加法和最小值运算，这两类运算占据了指令数量的大部分。为提高效率，我们引入融合加法-最小值运算，将连续的加法和最小值指令（vaddminu.vv/vaddminu.vx）合并为一条指令，从而降低指令开销。指令集架构和实现细节见 III-C节。

C. 指令集架构扩展与实现

III-B节中的数据流优化策略需要指令集架构和微架构的支持。表 I 总结了 VEGA 的专用指令集架构扩展，用于解决性能分析中发现的性能瓶颈。这些自定义 RVV 指令针对基因组比对中特定的内存访问模式和计算特性，解决了当前 RVV 实现中 SIMD 通道利用率低的问题，同时保留了 RISC-V架构的可编程性优势。

所提出的 VEGA 扩展属于 RVV 的 OPMVX、OPIVX 和 OPIVV类别，可对向量—向量或向量—标量输入进行操作。我们通过添加 VEGA 专用功能单元（包括优化后的 SLDU 和 SIMD 算术逻辑单元）来增强 Ara2，以支持这些用于基因组计算的新指令集架构。

带滑动操作。在 FFS 数据流中，SIMD通道上的每个前沿都必须根据其方向移动一个元素或保持不动。这给实现序列间并行带来了挑战：SIMD通道上的每个带都需要独立控制（移动或保持不动）。然而，RVV 1.0 [31]缺乏这种细粒度滑动能力，因为其原生 vslide1up/down指令作用于整个 VLEN位通道。为了解决这一限制，我们引入了 vslide1band.vx 指令，以实现逐带滑移和高效的字符串加载。vslide1band.vx 指令采用 OPMVX 格式，并提供两项关键功能：细粒度带滑移和快速字符串加载操作。我们修改了 Ara2 的 SLDU，以实现带滑移操作。图 7-(a) 展示了如何以最少的硬件开销实现带滑动操作。SLDU 会对 64 位 rs1 操作数进行解码，其中每个 4 位片段使用图 7-(b) 中的编码表控制对应的带行为。例如，0x0 使带保持不动，而 0x1 到 0x7 则通过填充使其向高位移动 1 个元素。这样，一条指令即可独立移动或保持最多 16 个带。

快速字符串加载与反转。为实现快速字符串加载，我们重复使用同一编码表来传输输入序列中的新字符。由于 DNA 序列（AGCT）可用 2 位编码，vslide1band.vx 每个带仅传输一个字符，而不必加载整个 BW 长度的字符串，从而降低延迟和内存带宽消耗。采用 OPMVX 格式的 vrevband.vx 可实现低延迟带反转功能，解决参考字符串反转操作开销过大的问题。我们添加了一个 256 位的混洗网络来实现字符串反转操作，并将控制信号编码到 rs1 操作数中，以确定反转宽度。VEGA 支持从 16 到 256 的 BW。与 RVV 的 vrgather.vv 指令相比，vrevband.vx 的延迟显著更低。

融合加法-最小值实现。为实现 III-B 节中的融合加最小值算术优化，我们开发了 vaddminu.vx 和 vaddminu.vv 指令，分别用于实现向量-标量和向量-向量融合加最小值操作。为提高 VEGA 的 SIMD 通道利用率，我们在每个 Ara2 通道的算术逻辑单元中增加了四个加最小值算术单元，从而减少所需发射的指令数量。

IV. 评估

A. 方法

软件实现。 带状 DP 对齐的 AVX2 和 RVV 内核使用 C 实现，并使用 LLVM（v16.0.6）在 -O3 优化级别下编译，采用 16 位整数算术。RVV 实现通过 SPIKE RISC-V 模拟器 [45]进行功能验证。

硬件仿真。VEGA 构建于 Ara2 RVV 处理器 [37], 之上，采用单发射、顺序执行的六级 CVA6 核心 [42] 以及一个支持 Zve64x 和 VEGA 扩展的向量单元（向量长度为 =4096b，包含 8 个通道）。16 核系统通过 Constellation [46]连接各核心，并配备一个 8 通道 DDR4-3200 内存。设计使用 GF 22 纳米 FD-SOI 工艺，并借助 Synopsys Design Compiler 在 1.25 GHz 下完成综合。SRAM 的时序和能耗参数使用 CACTI-3DD [47]建模。

基线。 我们通过测量裸机模式下执行的动态规划内核的性能，评估四种硬件基线（CPU、GPU、RVV 和专用集成电路）。我们实现了 CPU-AVX2 内核 [10]，使其吞吐量可与 BSAlign [9]相当。该内核运行于一台 8 核、采用 7 nm 工艺的 AMD Ryzen 7 9700X 上（TDP 为 65 W，芯片面积为 71 mm^2），处理器频率为 3.8–5.5 GHz，配备 64 GB DDR5-4800 内存。我们采用 AGAThA [48]作为 GPU 基线，其运行于 NVIDIA RTX 4090 显卡（采用 5 nm 工艺，TDP 为 450 W，芯片面积为 609 mm^2），频率为 2.5 GHz，配备 24 GB GDDR6X 内存。Ara2-RVV 基线在 Ara2 处理器 [37]上执行向量化内核。最后，专用集成电路基线包括可编程加速器和基于 RISC-V 的加速器，例如 Darwin [23],、GenDP[22],、GMX [12], 和 QUETZAL [11] （Fujitsu A64FX [30],、ARM SVE2 [29]）。

表 II：VEGA 面积和功耗结果

设计/组件		面积（mm^2）	功耗（mW）
CVA6 内核		0.125	17.3
原始 Ara2		0.670	48.0
VEGA	Ara2 通道 ×8	0.518	58.6
	Ara2 SLDU	0.022	1.6
	Ara2 VLSU	0.053	3.0
	Ara2 MASKU	0.052	2.4
	其他组件	0.006	0.7
原始 CVA6+Ara2		0.796	65.3
VEGA 内核（CVA6+Ara2）		0.804	67.3
VEGA 片上总线		0.218	14.1
VEGA 16 核处理器		12.898	1,090.4

基线。 我们通过测量裸机模式下执行的动态规划内核的性能，评估四种硬件基线（CPU、GPU、RVV 和专用集成电路）。我们实现了 CPU-AVX2 内核 [10]，使其吞吐量可与 BSAlign [9]相当。该内核运行于一台 8 核、采用 7 nm 工艺的 AMD Ryzen 7 9700X 上（TDP 为 65 W，芯片面积为 71 mm^2），处理器频率为 3.8–5.5 GHz，配备 64 GB DDR5-4800 内存。我们采用 AGAThA [48] 作为 GPU 基线，其运行于 NVIDIA RTX 4090 显卡（采用 5 nm 工艺，TDP 为 450 W，芯片面积为 609 mm^2），频率为 2.5 GHz，配备 24 GB GDDR6X 内存。Ara2-RVV 基线在 Ara2 处理器 [37]上执行向量化内核。最后，专用集成电路基线包括可编程加速器和基于 RISC-V 的加速器，例如 Darwin [23]、GenDP[22]、GMX [12]，和 QUETZAL [11]（Fujitsu A64FX [30]、ARM SVE2 [29]）。

工作负载。 我们评估了基因组分析中两种广泛使用的工作负载：*编辑距离* 和 *Smith-Waterman-Gotoh(SWG)* 算法。SWG 使用仿射罚分：$match = 0, mismatch = 4, o = 6\ e = 2$。我们评估了四种长度，包括两个短读长数据集（100个碱基对和200个碱基对）以及两个长读长数据集（1千个碱基对和10 千碱基对）。采用 [9, 10, 13] 中使用的真实世界数据集作为评估数据集。

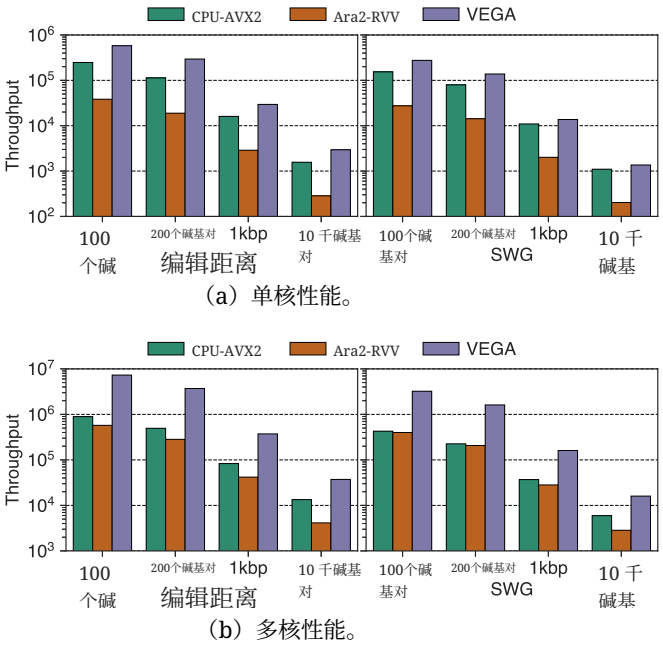
B. 面积与功耗分析

表 II 总结了 VEGA 综合后的面积和功耗结果。采用定制化指令集架构扩展的 VEGA 内核仅使基线 CVA6 [42]+Ara 2 [37] 设计的面积和功耗分别增加 1.0% 和 3.1%，增幅很小，主要来自新增的洗牌网络和融合加法-最小值算术逻辑单元。16 核 VEGA 处理器的面积为 12.898 mm^2，功耗为 1090.4 mW，在面积和功耗开销极小的情况下显著加速了基因组比对。

C. 性能比较

单核与多核性能。图 8-(a) 比较了 VEGA 的单核吞吐量与 CPU-AVX2 和 Ara2-RVV 基线。对于短读段，VEGA 在编辑距离和 SWG 方面相较于 Ara2-RVV，最高分别提升 15.1× 和 9.7×；对于 10 kbp 长序列，仍可实现 10.2× 和 6.7× 的加速比。其相较于 CPU-AVX2 优势较小，原因在于后者频率更高（5.5 GHz 对比 1.25 GHz），且缓存层次结构更大。尽管存在这些因素，VEGA 仍能保持

实际吞吐量为每秒 2,946 次比对（编辑距离）和每秒 1,361 次比对（SWG）。采用 16 核 VEGA



（a）单核性能。

（b）多核性能。

图 8：与 AVX2 和 RVV 基线的吞吐量比较。

表 III：长读长带状 SWG 设计的面积和功耗效率比较。

设计	CPU-AVX2 QUETZAL [11]	GMX-SWG [12]	AGAThA [48]	VEGA（我们的方案）
平台	CPU	CPU	CPU+专用集成电路	GPU	VPU
ISA	x86 AVX2	ARM SVE2	定制化 RV64G	PTX	RISC-V RV64GCV
VLEN	256b	512b	/	/	4096b
面积（mm^2）	71	2.9	1.2	609	12.9
Tp./mm^2 ↑*	4.9 (1.3×)	3.7† (1.0×)	1,625.0 (439.2×)	40.6 (11.0×)	1,241.5 (335.5×)
比对数/毫焦耳 ↑*	0.5 (1.0×)	/	5.1 (10.2×)	1.8 (3.6×)	14.7 (29.4×)

*按 22 纳米和 10 千碱基对缩放。 † 将 10 千碱基对和 30 千碱基对缩放至 10 千碱基对。

配备 8 通道 DDR4-3200 内存（图 8-(b)），相较于 Ara2-RVV，在短读段上实现了12.7× 和 8.0× 更高的吞吐量，并在 10 千碱基对序列上保持 5.6×–8.9× 的增益，展现出良好的可扩展性和并行效率。

与现有加速器的比较。 表 III 将 VEGA 与最先进的 CPU 和 GPU 基线进行比较 [11, 48]。为确保公平比较，所有设计均采用 [49] 中的方法缩放至 22 纳米。VEGA 的面积效率达到 1,241.5 吞吐量/mm^2，在可编程设计中最高，仅次于硬连线的 GMX-SWG ASIC；与 CPU-AVX2 和 GPU 基线相比，其性能分别提升了 253.4×和 30.6× 。在所有参与比较的设计中，VEGA 还实现了最高的能效，达到 14.7 比对数/毫焦耳。VEGA 的卓越性能提升源自其专用向量扩展和 4096 位向量长度。值得注意的是，QUETZAL 是另一款基于 ARM SVE 指令集架构的基因组加速器。它主要针对降低延迟的内存访问进行优化，从而提供加速效果。然而，与 VEGA 相比，QUETZAL 的长读段性能较差，因为它采用了

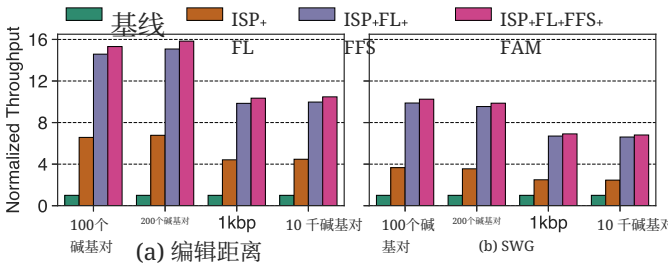


图 9：VEGA 各项优化带来的增量改进（ISP=序列间并行，FL=快速字符串加载，FFS=前沿驻留数据流，以及 FAM=融合加最小值）。

固定的 512位向量长度，无法充分利用序列间并行。相比之下，采用灵活向量长度的 VEGA 能够有效利用不同层级的数据并行，从而实现更高的吞吐量和效率。

优化的影响。图 9 展示了所提出的 VEGA 优化方案相较于 Ara2-RVV 基线，在不同序列长度和 DP 算法下带来的增量贡献。前沿驻留数据流（FFS）和快速字符串加载（FL）带来了最显著的两项性能提升。融合加最小值（FAM）操作由于降低了指令开销，额外提升了 10% 的吞吐量。完整的优化栈（ISP+FL+FFS+FAM）相较于 Ara2-RVV 基线，吞吐量最高提升 $15\times$ 。在不同序列长度和算法类型下，性能优势始终如一。

V. 结论

本文介绍了 VEGA，这是一种新型且可扩展的软件与硬件协同设计方案，通过专用 RVV 扩展加速基因组比对。我们的方法带来了三项关键创新：前沿驻留数据流，可最大限度地减少内存带宽占用；序列间并行，可最大限度地提高 SIMD 通道利用率；以及面积开销极小的专用向量指令。VEGA 在效率、可编程性和可扩展性之间实现了有效平衡，在保持算法灵活性的同时显著提升了性能。

参考文献

[1] E. A. Ashley，“迈向精准医学”，Nature Reviews Genetics，第 17 卷，第 9 期，第 507–522 页，2016 年。[2] A. L. Byrd 等人，“Clinical pathoscope：利用未组装测序数据快速比对和过滤临床样本，以准确鉴定病原体”，BMC Bioinformatics，第 15 卷，第 1–14 页，2014 年。[3] B. E. Slatko 等人，“下一代测序技术概述”，Current Protocols in Molecular Biology，第 122 卷，59，2018 年 1 期，第 e 页。[4] T. F. Smith 等人，“常见分子子序列的鉴定”，Journal of Molecular Biology，第 147 卷，第 1 期，第 195–197 页，1981 年。[5] S. B. Needleman 和 C. D. Wunsch，“一种适用于搜索两个蛋白质氨基酸序列相似性的一般方法”，Journal of Molecular Biology，第 48 卷，第 3 期，第 443–453 页，1970 年。[6] V. I. Levenshtein 等人，“能够纠正删除、插入和反转的二进制编码”，载于《苏联物理学文献》，第 10 卷，第 8 期，1966 年，第 707–710 页。[7] K.-M. Chao 等人，“在指定对角带内比对两个序列”，Bioinformatics，第 8 卷，第 5 期，第 481–487 页，1992 年。[8] H. Suzuki 和 M. Kasahara，“引入差分递推关系以加速长序列的半全局比对”，BMC Bioinformatics，第 19 卷，第 33–47 页，2018 年。[9] H. Shao 和 J. Ruan，“BSAlign：核苷酸序列比对库”，Genomics, Proteomics & Bioinformatics，第 22 卷，第 2 期，第 qzae025 页，2024 年。[10] S. Marco-Sola 等人，“使用波前算法进行快速间隙仿射成对比对”，Bioinformatics，第 37 卷，第 4 期，第 456–463 页，2021 年。[11] J. Pavon 等人，“QUETZAL：用于现代基因组序列分析算法的向量加速框架”，载于 ACM/IEEE 计算机体系结构年度国际研讨会（ISCA），2024 年，第 597–612 页。[12] M. Doblas 等人，“GMX：用于快速、可扩展且高效的基因组序列比对的指令集扩展”，载于 IEEE/ACM 微体系结构国际研讨会（MICRO），2023 年，第 1466–1480 页。[13] D. Liu 和 M. Steinegger，“块比对器：一种针对序列和位置特异性评分矩阵的自适应 SIMD 加速比对器”，Bioinformatics，第 39 卷，第 8 期，第 btad487 页，2023 年。[14] H. Li，“Minimap2：核苷酸序列的成对比对”，Bioinformatics，第 34 卷，第 18 期，第 3094–3100 页，2018 年。[15] A. Zeni 等人，“利用差分递推关系实现高性能 GPU 基因组比对”，载于并行体系结构与编译技术国际会议，2024 年，第 133–143 页。[16] N. Ahmed 等人，“Gasal2：用于高通量 NGS 数据的 GPU 加速序列比对库”，BMC Bioinformatics，第 20 卷，20，2019 年 1pp. –。[17] Q. Aguado-Puig 等人，“WFA-GPU：利用 GPU 进行间隙仿射成对读段比对”，Bioinformatics，第 39 卷，第 12 期，第 btad701 页，2023 年。[18] Y.-L. Chen 等人，“用于全人类基因组短读段比对的高通量 FPGA 加速器”，IEEE Transactions on Parallel and Distributed Systems，第 32 卷，第 6 期，第 1465–1478 页，2021 年。[19] Y. Feng 等人，“用于纳米孔测序数据分析中自适应带状事件比对的 FPGA 加速器”，BMC Bioinformatics，第 26 卷，第 1 期，第 83 页，2025 年。[20] A. Haghi 等人，“WFASic：用于 RISC-V 片上系统上 DNA 序列比对的高性能 ASIC 加速器”，载于并行处理国际会议，2023 年，第 392–401 页。[21] Y.-L. Liao 等人，“适用于长读段的自适应带状 Smith-Waterman 算法及其硬件加速器”，载于 IEEE 应用专用系统、体系结构与处理器国际会议，2018 年，第 1–9 页。[22] Y. Gu 等人，“GenDP：基因组测序分析的动态规划加速框架”，载于 ACM/IEEE 计算机体系结构年度国际研讨会（ISCA），2023 年，第 1–15 页。[23] Y. Turakhia 等人，“Darwin：一种基因组学协处理器，可将长读段组装速度提升至 15,000 倍”，ACM SIGPLAN Notices，第 53 卷，第 2 期，第 199–213 页，2018 年。[24] D. S. Cali 等人，“GenASM：用于基因组序列分析的高性能、低功耗近似字符串匹配加速框架”，载于 IEEE/ACM 微体系结构国际研讨会（MICRO），2020 年，第 951–966 页。[25] A. Alonso-Mar´ın 等人，“BIMSA：利用存内处理加速长序列比对”，Bioinformatics，第 40 卷，第 11 期，2024 年。[26] W. Xu 等人，“RapidX：用于序列比对的高性能 ReRAM 存内处理加速器”，IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems，第42卷，第10期，第3275–3288页，2023年。[27] N. Mansouri Ghiasi 等人，“Genstore：一种用于基因组序列分析的高性能存储内处理系统”，载于 ACM International Conference on Architectural Support for Programming Languages and Operating Systems，2022年，第635–654页。[28] A. Waterman 等人，“RISC-V指令集手册，第一卷：用户级指令集架构，2.0版”，加州大学伯克利分校电气工程与计算机科学系，技术报告 UCB/EECS-2014-54，第4页，2014年。[29] Arm有限公司，“SVE2”，2020年。 [在线]。网址：https://developer.arm. com/documentation/ 102340 0100 2 / /Introducing-SVE [30] 富士通，“富士通处理器 A64FX”，https://www.fujitsu.com/global/products/computing/servers/supercomputer/a64fx/。[31] RISC-V向量扩展任务组，“RISC-V向量扩展，1.0版”，GitHub代码库，2021年。 [在线]。网址：https: //github.com/riscvarchive/riscv-v-spec/releases/tag/v 1 0 。[32] C. Wang 等人，“Speed：一种可扩展的RISC-V向量处理器，实现高效的多精度深度神经网络推理”，IEEE Transactions on Very Large Scale Integration (VLSI) Systems，2024年。[33] M. Perotti 等人，“Spatz：通过集群化紧凑型基于RISC-V的向量单元，最大限度提升计算效率”，IEEE集成电路与系统计算机辅助设计汇刊，2025年。[34] F. Minervini 等人，“Vitruvius+：面向高性能计算应用的高面积效率RISC-V解耦式向量协处理器”，ACM架构与代码优化汇刊，第20卷，第2期，第1–25页，2023年。[35] T. Szymkowiak 等人，“Marian：一种支持Zvk向量密码学扩展的开源RISC-V处理器”，载于2024年ACM SIGSAC计算机与通信安全会议论文集，2024年，第4931–4933页。[36] L. L´opez-Villellas 等人，“用于基因组数据分析的RISC-V：机遇与挑战”，载于电路与集成系统设计会议（DCIS），2023年，第1–6页。[37] M. Perotti 等人，“Ara2：利用高效且符合RVV 1.0标准的开源处理器，探索单核与多核向量处理”，IEEE计算机汇刊，2024年。[38] M. Farrar，“条带式Smith–Waterman算法使数据库搜索速度比其他SIMD实现快六倍”，BMC Bioinformatics，第23卷，第2期，第156–161页，2007年。[39] Arm有限公司，“Arm Neon SIMD”，https://developer.arm.com/Architectures/Neon。[40] AMD，“AMD Ryzen 7 9700X处理器”，https://www.amd.com/en/products/processors/desktops/ryzen/9000-series/amd-ryzen-7-9700x.html。[41] 英特尔公司，“Intel®高级向量扩展2”，2022年。[42] F. Zaruba 和 L. Benini，“应用级处理的代价：22纳米FDSOI技术中适用于Linux的1.7 GHz 64位RISC-V核心的能耗与性能分析”，IEEE Transactions on Very Large Scale Integration (VLSI) Systems，第27卷，第11期，第2629–2640页，2019年。[43] J. Daily，“Parasail：用于全局、半全局和局部两两序列比对的SIMD C语言库”，BMC Bioinformatics，第17卷，第1期，第81页，2016年2月。[44] T. Rognes，“通过序列间SIMD并行化加速Smith-Waterman数据库搜索”，BMC Bioinformatics，第12卷，第1期，第221页，2011年6月。[45] “Spike：RISC-V指令集架构模拟器”，https://github.com/riscv-software-src/ riscv-isa-sim。[46] J. Zhao 等人，“Constellation：一种开源、支持片上系统的片上网络生成器”，载于IEEE/ACM片上网络架构国际研讨会（NoCArc），2022年，第1–7页。[47] K. Chen 等人，“CACTI-3DD：用于3D堆叠DRAM主存的架构级建模”，载于欧洲设计、自动化与测试会议暨展览会（DATE），2012年，第33–38页。[48] S. Park 等人，“AGAThA：面向长读长比对的引导式序列比对GPU快速高效加速”，载于ACM SIGPLAN并行编程原理与实践年度研讨会，2024年，第431–444页。[49] S. Sarangi 和 B. Baas，“DeepScaleTool：用于准确估算深亚微米时代技术缩放的工具”，载于IEEE国际电路与系统研讨会（ISCAS），2021年，第1–5页。