面向高层次物理综合的自动化设计空间探索

Linfeng Du[1], Jiawei Liang[2], Jason Lau[1], Yuze Chi[1], Yutong Xie[1], Chunyou Su[2], Afzal Ahmad[2], Zifan He[3], Jake Ke[3], Jinming Ge[2], Jason Cong[3], Wei Zhang[2§], Licheng Guo[1§]

[1]RapidStream Design Automation 公司 [2] 香港科技大学 [3] 加利福尼亚大学洛杉矶分校

linfeng.du@connect.ust.hk,wei.zhang@ust.hk,lcguo@rapidstream-da.com

摘 要

在大规模多芯粒 FPGA 上实现 HLS 加速器极具挑战。为此，研究者提出了高层次物理综合（HLPS），通过协同优化高层次综合与物理设计来提升可达频率。然而，现有 HLPS 技术的结果质量（QoR）往往不稳定且缺乏一致性，主要原因在于仍有大量参数需要用户凭经验临时选定。因此，要获得令人满意的方案，仍然需要大量人工投入以及底层电路设计方面的专业知识。

本文提出一套稳健且实用的设计空间探索（DSE）框架，通过自动化的迭代参数调优过程来提升 HLPS 的可靠性和结果质量。该框架依据从物理实现结果中提取的指标，采用有针对性的启发式策略细化 HLPS 参数，从而实现稳定且自动化的时序收敛。在针对大规模真实设计、并在代表性多芯粒器件上开展的评估中，该框架取得了 311.06 MHz 的平均频率，分别达到 AMD Vitis/Vivado 工具链（128.48 MHz）和领先学术方案（186.21 MHz）的 2.42 倍和 1.67 倍。

CCS 概念

• **硬件 → 高层次综合与寄存器传输级综合**; **分区与布局规划**; **EDA 软件工具**.

关键词

高层次物理综合、设计空间探索、多芯粒 FPGA、时序收敛、布局规划、流水化、物理指标

IEEE 引用格式:

Linfeng Du, Jiawei Liang, Jason Lau, Yuze Chi, Yutong Xie, Chunyou Su, Afzal Ahmad, Zifan He, Jake Ke, Jinming Ge, Jason Cong, Wei Zhang, Licheng Guo. 2025. Automated Design Space Exploration in High-Level Physical Synthesis. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '25)*. https://doi.org/10.1109/ICCAD66269.2025.11240757

本文为非正式中文译本，仅供个人学习与参考。如中英文内容存在差异，以英文原版为准。

1 引言

现代 FPGA 已演进为更大规模的多芯粒架构，但在这些器件上编译 HLS 加速器设计仍是一个尚未解决的难题，其根源在于器件和编译器两方面的局限。在器件层面，跨芯粒边界的物理连线密度有限（仅约为芯粒内连接的 ~1/4 [7]），且在 AMD/Xilinx 堆叠硅互连（SSI）技术 [1] 中，信号延迟会增加 4–8 倍。在编译器层面，HLS 工具以无时序约束的高层语言为输入，由于难以预测布局布线（PnR）后的物理布局，往往难以获得理想的结果质量（QoR），容易产生流水级不足、跨距过长的连接，从而降低工作频率。

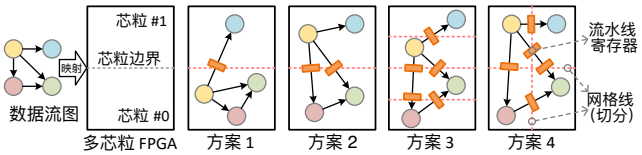

图 1: 一个简单示例，展示了 HLPS 中不同的器件抽象、布局规划和流水策略：方案 1 沿芯粒边界划分器件；方案 2 通过添加流水寄存器平衡芯粒间的资源利用率；方案 3 通过划分为四个纵向堆叠的槽位来解决器件纵向偏长的外形比例所导致的长垂直连线；方案 4 添加纵向切割以实现更合理的水平逻辑分布。

近年来，研究者引入了**高层次物理综合（HLPS）**[7, 12, 19] 方法来提升可达频率，并在众多真实设计 [3, 4, 6, 14, 15, 29–34] 中展示了显著效果。HLPS 的核心思想是在 C/C++ 层面识别潜在的长线连接，并在从 C 到寄存器传输级（RTL）的编译过程中预先插入流水寄存器，利用 HLS 设计中普遍存在的延迟容忍性。HLPS 在延迟容忍的接口处对设计进行分区，在 FPGA 上进行布局规划，并沿长距离连接插入流水寄存器以改善时序收敛。该过程不会牺牲吞吐量，且对初始延迟的影响极小 [12]。

尽管已有工作展示了 HLPS 的潜力，但其方案的时序质量仍然高度不稳定，原因在于分区、布局规划和流水化中存在庞大的参数空间。图 1 展示了同一 HLS 设计如何因所选参数的不同而产生截然不同的布局。部分方案将模块紧密排列以减少全局线长，但代价是增加局部拥塞；另一些方案则倾向于更分散的布局，优先保证局部可布线性而非全局线长最小化。

然而，由于 PnR 工具复杂的黑盒特性，在未完成完整实现的情况下预测最终 QoR 极具挑战性，甚至不可能。例如，在 GPT-2 Medium 加速器 [13] 的实验中，两个相近布局规划方案之间的细微参数调整就会导致 5 倍的频率差异（229.2 vs. 43.1 MHz）。此外，不同设计往往需要不同的优化策略，一个 HLPS 设计可能达到 300+ MHz，而使用相同参数的另一个设计却可能无法完成布线。因此，以往的 HLPS 应用始终依赖大量人工干预和手动参数调优。

本文引入一套实用且高效的探索框架，通过迭代式、物理引导的优化自主搜索最优参数配置。该框架从早期迭代的实现结果中提取指标，以指导后续迭代的参数选择，从而降低加速器设计者对人工干预和领域专门知识的依赖。本工作的主要贡献如下：

§ 通讯作者：Wei Zhang 和 Licheng Guo.

ICCAD '25, 2025 年, 美国新泽西

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1109/ICCAD66269.2025.11240757

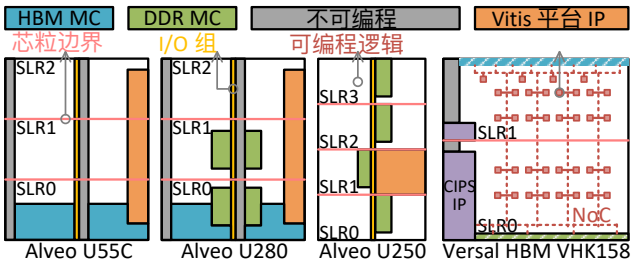

图 2: 所评估的代表性多芯粒 FPGA 架构概览。

- 据我们所知，本工作提出了首个面向 HLPS 的迭代式设计空间探索（DSE）框架，显著提升了方案的稳健性，并实现了高于现有商业和学术方案的工作频率。
- 我们引入了一组代表性的物理指标来评估 HLPS 方案质量，并开发了提取这些指标的工具。基于这些指标，我们提出了简洁而有效的启发式策略来指导 HLPS 参数的探索。
- 我们通过真实 HLPS 应用中可达频率的显著提升展示了该框架的有效性。在六个大规模加速器设计和四种不同多芯粒架构上的评估中，框架取得了 311.06 MHz 的平均频率，分别达到商业工具 AMD Vitis（128.48 MHz）和此前最先进的 HLPS 工具 AutoBridge [12]（186.21 MHz）的 2.42 倍和 1.67 倍。

2 背景

2.1 多芯粒 FPGA

现代 FPGA 集成了具有多样化架构的多个芯粒。我们考察四种代表性多芯粒 FPGA（图 2）以说明其面临的挑战。Alveo U55C 拥有三个芯粒（即 AMD 术语中的超级逻辑区域 SLR），在 SLR0 上配备 32 个 HBM 端口，右侧一列承载用于主机–设备通信的 Vitis 平台 IP。Alveo U280 在 SLR0/SLR1 上额外配备两个 DDR 内存控制器（MC）。Alveo U250 拥有四个芯粒，最新的 Vitis 平台 IP 占据了 SLR1 右半部分的全部空间。Versal FPGA 通过分布于器件各处的硬件片上网络（NoC）访问存储器，具有更大的逻辑芯粒和更稀疏的超长线（SLL）分布。这些异构资源与复杂互连给物理实现带来了巨大挑战。

2.2 高层次物理综合

HLPS 将输入的 HLS 设计抽象为图 $G(V, E)$，其中顶点 V 表示设计模块，边 E 表示延迟容忍的连接。HLPS 流程包含三个主要阶段：

(1) **器件抽象**：将目标 FPGA 器件表示为 $\pi_X \times \pi_Y$ 的网格，每个槽位代表一个独立的物理区域。
(2) **布局规划**：在满足资源约束、均衡利用率和避免拥塞的条件下，将模块分配到槽位。
(3) **流水化与布线**：为跨槽位连接插入流水寄存器，并在线容量限制下求解布线路径。

完成这三个步骤后，HLPS 会生成物理感知的 RTL 和布局规划约束，并交由综合与实现工具，以落实预期的物理布局。

传统 HLPS 流程涉及大量需凭经验确定的参数，如表 1 所总结。这些参数的细微变化可能导致截然不同的方案，如图 1 所示，最终工作频率也会出现显著波动。下面举几个代表性的例子，帮助读者直观理解这些参数。例如，在步骤 (1) 中，用户

表 1: HLPS 设计空间中的可调参数

符号	定义	定义域
器件抽象相关（§6.1）		
π_X, π_Y	π_X（列）× π_Y（行）的器件网格	$\pi_X, \pi_Y \in \{1, 2, 3, 4\}$
$M_{i,q}$	类型 q 的第 i 个接口绑定到 IP $M_{i,q}$	$q \in$ {DDR, HBM, NoC, PCIe, ...}
Φ	抽象调度方式	$\Phi \in$ {扁平, 层次化}
布局规划相关（§6.2）		
R_t^+, R_t^-	类型 t 的全局最大/最小资源限制	$R_t^{+/-} \in [0, 1]$; $t \in$ {BRAM, DSP, FF, LUT, URAM}
$R_{s,t}^+, R_{s,t}^-$	槽位 s 中类型 t 的最大/最小限制	$R_{s,t}^{+/-} \in [0, 1]$
流水化与布线相关（§6.3）		
C_b	边界 b 的跨槽位线（穿越）容量	$C_b \geq 0$
d	流水密度	$d \in$ {槽内, 双级, 混合, 单级}
l_i	第 i 个跨芯粒线束的可用 SLL 比例	$l_i \in [0, 1]$

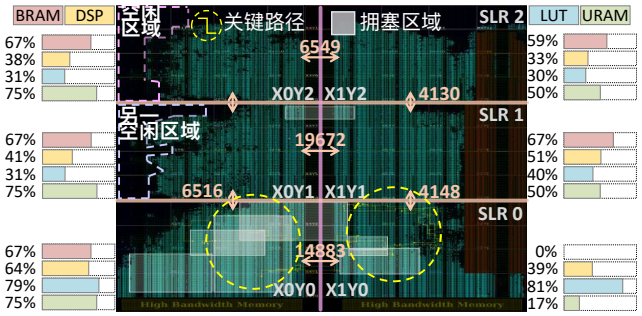

图 3: Sextans SpMM 在 Alveo U55C 上实现后的物理指标，其中三芯粒器件被抽象为 2（列）× 3（行）的网格。

必须选择网格尺寸 $\pi_X \times \pi_Y$；过大的槽位可能导致槽位内连接流水级不足，而过小的槽位则可能影响细粒度布局质量。在步骤 (2) 中，用户需要为所有槽位定义合适的资源上限，以防止局部拥塞和资源利用不足。在步骤 (3) 中，用户需要指定插入的流水寄存器数量，以及槽位对之间允许通过的连线数量。合适的配置有助于减少热点区域的连线，从而缓解布线拥塞。

3 示例分析

为说明 HLPS 参数搜索中的挑战与机遇，我们展示调优两个关键参数——资源限制和线容量——如何显著影响实现质量。我们以 Sextans 为例，这是一个真实的稀疏矩阵–稠密矩阵乘法（SpMM）设计 [30]，部署在 Alveo U55C 上。默认的 AMD Vitis 流程对此设计达到 101.52 MHz，而 AutoBridge [12]（一种传统 HLPS 方法）经过布局布线后将频率提升至 182.05 MHz。图 3 展示了实现后的器件视图及相关物理指标。

该设计的物理布局显示底部芯粒存在过度拥塞，尤其是在槽位 X0Y0，这是由于 HBM 内存控制器集中实例化所致，而槽位 X0Y1/X0Y2 则大部分空闲。我们通过在布局规划阶段降低 X0Y0 的资源限制 $R^+_{X0Y0,LUT}$ 来解决这一利用率不均衡问题，使模块从 X0Y0 迁移到上方槽位，从而将最大频率（Fmax）提升到 248.92 MHz，比 AutoBridge 提升 36.73%。

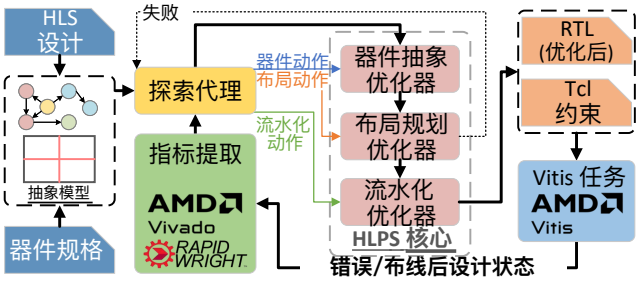

图 4: HLPS 中指标引导的 DSE 框架。

进一步分析槽位边界的穿越情况揭示了另一个优化机会。大量跨越纵向边界 X0Y0–X1Y0 和 X0Y1–X1Y1 的连线形成了密集拥塞。通过降低这些边界处的线容量，即 $C_{X0Y0-X1Y0}$、$C_{X0Y1-X1Y1}$，连线在跨槽位布线阶段会被重路由到其他路径，使各边界的利用率更趋均衡。该优化达到 285.36 MHz，最大频率提升 56.75%。

这两个直观的修正揭示了显著的优化机会，但需要系统性的分析才能充分释放参数搜索的潜力。这一观察激发了我们的 HLPS DSE 框架，在物理指标与有效的 HLPS 参数调优动作之间建立直接联系。

4 框架概览

图 4 展示了我们的自动化 HLPS DSE 框架。HLPS 核心根据当前探索得到的参数执行物理优化。每次迭代后，指标提取器分析实现结果，并指导探索代理调整参数，我们将这些调整称为 HLPS 动作。值得注意的是，我们增强了 HLPS 核心的全部三个阶段：

- **器件抽象优化器**：一个新组件，负责：(1) 尽早将类型 q 的第 i 个接口绑定到 IP $M_{i,q}$（如 DDR/HBM/NoC），(2) 配置器件网格维度 (π_X, π_Y)，以及 (3) 在扁平与层次化方法之间自适应切换布局规划整数线性规划（ILP）的求解调度，以权衡方案质量与执行时间。
- **布局规划优化器**：增强了传统 HLPS 布局规划的 ILP 公式（表 2），提供更精细的控制和更好的可扩展性。它在全局和槽位级资源约束 $(R_t^{+/-}, R_{s,t}^{+/-})$ 下寻找最优槽位分配，同时最小化跨槽位连接总数。
- **流水化优化器**：用 ILP 替代传统的固定 HLPS 流水化和布线方案（表 3）。它为每个跨槽位连接 n 选择最优路径 $p_{n,i}$，在线容量限制 C_b 下最小化最大穿越利用率 μ，同时确定每个连接的合适流水密度 d。对于跨芯粒线束，还额外优化 SLL 可用比例 l_i 以实现更精细的控制。

这三个优化器构成了框架的核心。为有效指导其参数选择，我们首先在第 5 节介绍用于刻画 HLPS 方案关键特征的物理指标。

5 物理指标

我们从实现后结果中识别出五个关键指标，用于评估 HLPS 方案并指导 DSE。这些指标为参数优化提供全面反馈：

(A) 实际资源利用率：每个槽位的资源利用率是物理优化的基础指导。高利用率的槽位表明需要向其分配更少的逻辑以减少拥塞。对于每个槽位，我们提取实现后关键资源的利用率：BRAM、DSP、FF、URAM 和 LUT。

表 2: 面向布局规划优化的最小总穿越 ILP

目标：最小化槽位穿越总数	$\min \sum_{e\in E}\sum_{s\in S}\sum_{d\in S} c_{e,s,d}\cdot x_{e,s,d}$ 其中 $c_{e,s,d} = \sum_{c\in \text{path}(s,d)} 1$
每条边恰好被路由一次	$\sum_s \sum_d x_{e,s,d} = 1,\ x_{e,s,d} \in \{0,1\}$
每个顶点 v 恰好分配到一个槽位	$\sum_s y_{v,s} = 1,\ y_{v,s} \in \{0,1\}$
边的源槽位与顶点分配匹配	$\sum_d x_{e,s,d} = y_{v,s}, v = \text{src}(e)$
边的目标槽位与顶点分配匹配	$\sum_s x_{e,s,d} = y_{v,d},\ v = \text{dst}(e)$
全局资源限制	$R_t^- \cdot A_{s,t} \le \sum_v r_{v,t}\cdot y_{v,s} \le R_t^+ \cdot A_{s,t}$
槽位级资源限制	$R_{s,t}^- \cdot A_{s,t} \le \sum_v r_{v,t}\cdot y_{v,s} \le R_{s,t}^+ \cdot A_{s,t}$
边穿越切割的定义	$z_{e,c} = \sum_{s\in S_1}\sum_{d\in S_2} x_{e,s,d} + \sum_{s\in S_2}\sum_{d\in S_1} x_{e,s,d}$
穿越切割的边宽度不超过容量 C_c	$\sum_e w_e \cdot z_{e,c} \le C_c$

表 3: 面向跨槽位布线优化的最小穿越比率 ILP

目标：最小化最大穿越利用率 μ	$\min \mu$
为连接 n 选择第 i 条路径	$\sum_{i=1}^{P_n} p_{n,i} = 1, \forall n \in N$
每个边界 b 的最大穿越利用率	$\mu \ge \frac{\sum_{(n,i)\in \text{crossings}_b} w_n \cdot p_{n,i}}{\max(C_b,1)}, \forall b \in B$
路径选择的二值约束	$p_{n,i} \in \{0,1\}$
非负上界	$\mu \ge 0$

(B) 布线拥塞分布：我们将拥塞窗口（互连使用高度密集的矩形区域）作为辅助指标，用于微调布局规划相关参数。其分布通过以下方式分析：(1) 以 LUT 数量衡量的每个槽位覆盖面积，(2) 相邻槽位之间共享的拥塞窗口，从而指导模块重分配和流水化优化。

(C) 关键路径分布：与最差负裕量（WNS）相关的时序路径不一定分布在高利用率或高拥塞区域。这些路径跨槽位的分布模式对应不同的优化策略：跨槽位关键路径指导流水密度决策，槽位内关键路径则指导资源限制调整。

(D) 槽位穿越分布：对于给定的槽位边界，穿越点的分布可能高度不均衡，尤其是在芯粒边界处。例如，大部分连线可能通过边界左半部分布线，而右半部分利用不足。这种空间上的不均衡为进一步优化提供了有价值的线索，包括缩小槽位尺寸、改变布线路径以及均衡各边界上的穿越点分布。

(E) 空闲区域分布：我们专门识别空闲区域，定义为没有放置逻辑的大片连续区域。槽位内存在空闲区域表明可用物理资源超出需求，这是一个强信号，表明可以安全地将更多逻辑移入该槽位而不违反资源约束。

6 指标引导的优化动作

基于在 UltraScale+ 和 Versal FPGA 上对大规模设计开展的广泛实验，我们识别出六个有效的优化动作，对应 HLPS 的三个阶段：器件抽象、布局规划和流水化优化。每个动作都在物理指

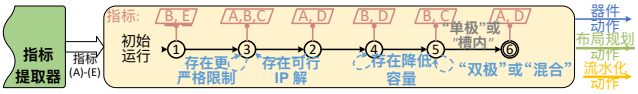

图 5: 指标引导的优化动作，组织为有限状态机。

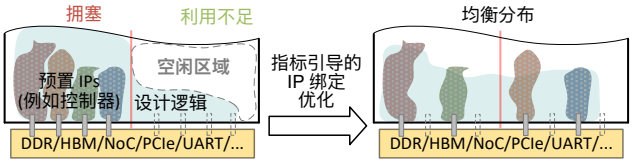

图 6: 基于物理指标引导的接口绑定，将接口重新绑定到利用不足槽位中的备选端口/IP，以缓解拥塞。

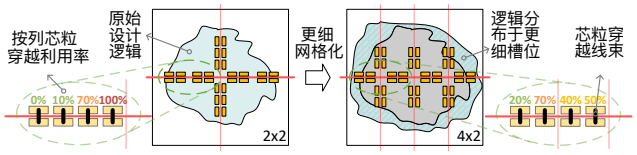

图 7: 更细的器件网格化有助于获得更好的初始物理布局、更广泛的逻辑铺展和更均衡的芯粒穿越。

标的引导下调整表 1 中定义的特定参数。这些动作遵循图 5 所示的自然拓扑依赖关系，形成系统化的优化流程。

6.1 器件抽象动作

动作 1 – 优化 IP 绑定：先前的 HLPS 工作 [7, 12, 19] 已证明快速布局规划与流水化优化是有效的，但忽略了接口到预置 IP 的绑定。这些方法依赖粗略的面积估算和经验式的槽位分配，并通过简单扣减估计的 IP 面积来设定资源上限。这种不精确的资源预算与固定的预分配方式往往会引发时序问题。以 HBM 内存控制器为例，它们通常会占用不可忽视的逻辑资源[1]并造成拥塞或迫使布线绕行 [8]。图 6 展示了一个左侧槽位因内存控制器全部投入使用而拥塞的案例。尽管拥塞（指标 (B)）可以指导一般性的重绑定决策，但对于预置 IP，我们会优先依据空闲区域进行分析。具体而言，我们的策略会识别相邻空闲空间充足（指标 (E)）的备选内存端口，因为这类布局紧凑的 IP 需要特定的资源形态和布线空间。该动作因其独立性而优先执行，并可持续优化直至最大频率或资源利用率趋于稳定；随后其结果可在后续迭代中复用，而不受其他动作影响。

动作 2 – 细化网格粒度：器件抽象需要将器件划分为 π_X 列、π_Y 行的网格。更细的网格使分区和流水化更精细，有助于切分时序路径，但也会增加逻辑开销。反之，较粗的网格虽然能够最小化开销，却可能不足以切分关键路径。

我们提出系统地探索网格维度 (π_X, π_Y)，因为先前的 HLPS 工作依赖经验性的切割插入，却未研究其对布局质量的影响。最优粒度因设计而异：过大的槽位会形成致密放置区域，需要物理层工具额外适配；过细的网格则会导致逻辑碎片化和利用率低下，可能引发时序退化或不可行方案。这两种情形都不符合 HLPS 的目标，即在不依赖设计专有经验或用户物理直觉的前提下，实现自动化物理优化。

[1]例如，Alveo U280 上的 32 个 HBM 内存控制器占据 SLR0 约 15% 的 LUT。

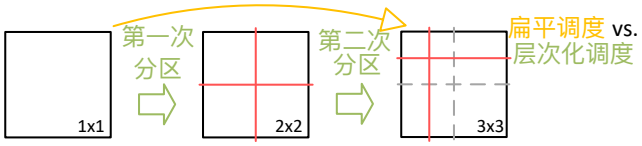

图 8: 将切割过多的扁平抽象调度切换为层次化调度，以保证 ILP 求解在计算上可行。

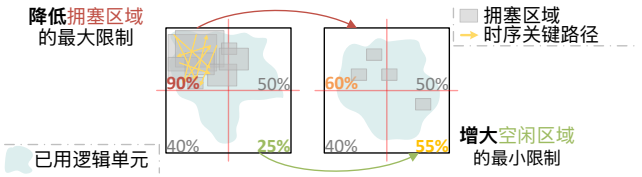

图 9: 基于拥塞分布指导的槽位级资源限制调整。

有别于仅将芯粒边界或 I/O Bank 作为器件切割的已有 HLPS 工作，我们引入与这些架构边界平行或垂直的切割，以释放更细网格化的潜力。我们的细化策略考虑槽位资源利用率（指标 (A)）和穿越分布（指标 (D)）。如图 7 所示，2×2 网格会导致逻辑集中在中心，边缘区域利用不足，芯粒穿越线束的利用率也不均衡。虽然这两类切割都有助于细分中等利用率的槽位，并促进子槽位间更均衡的逻辑分布，但垂直切割具有关键优势：当芯粒边界被切分为更短的边界段时，它能缓解芯粒穿越线束周围因线容量有限和布线不均衡而形成的拥塞瓶颈。这一细化产生了 4×2 网格，使流水化优化器能够利用先前位于角落槽位之间、尚未被利用的切割边界。跨槽位路径上的额外寄存器在现代 FPGA 丰富的 FF 资源下只带来可接受的开销 [8]，并可进一步配合第 6.3 节详述的线容量控制。

网格化过程每次迭代添加或调整一条切割。我们优先切割较长的边界，尤其是在水平芯粒边界上施加纵向切割以优化时序。所有划分区域都必须保持矩形，以避免出现 U 形区域。该过程会持续细化粒度，直到达到终止条件：频率进入平台期、模块与槽位不兼容，或每个槽位达到 n 个时钟区域；对于时钟区域较大的器件，较小的 n 值更有利于维持足够的粒度。随着槽位进一步细化，ILP 求解的计算开销也会随之增长，带来额外挑战。为提高 ILP 效率，我们根据模块分配数量对切割方案分组，因为这一指标与问题复杂度相关。我们不使用纯增量布局规划 [8] 或以运行时为导向的迭代二分 [12]，而是在超出时间限制时，自适应地将抽象调度从扁平模式切换为层次化模式（图 8）。

鉴于网格化粒度从根本上影响后续的布局规划/流水化质量和 ILP 可行性，每个新的网格和调度 $(\pi_X, \pi_Y; \Phi)$ 都会启动一轮全新的布局规划试验，并先于 DSE 过程中的所有流水化动作执行。

6.2 布局规划动作

动作 3 – 控制资源限制：在布局规划阶段，必须指定每个槽位的资源限制以指导模块分配，同时最小化全局线长。较高的限制允许更密集的模块打包，减少全局连线但可能导致局部拥塞；较低的限制带来更分散的放置，缓解局部拥塞但增加全局连接长度。

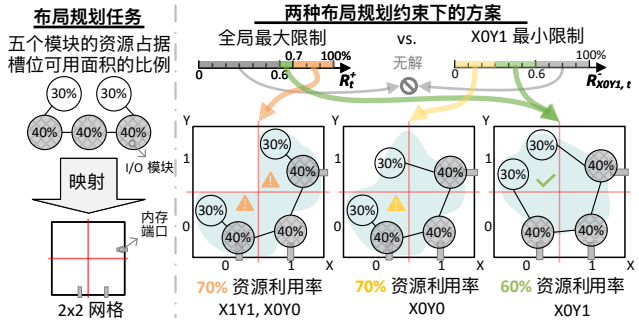

图 10: 最小利用率约束有效补充最大利用率约束，共同加速向均衡布局规划的收敛。

我们的框架接受用户指定的全局资源限制范围 $R_t^{+/-}$。在此全局控制之外，我们提出由指标 (A)–(C) 引导的槽位级资源限制调节，对拥塞严重的槽位施加更紧的约束，详见算法 1。例如，设全局限制为 90%（图 9），左上槽位可能因密集连接的模块聚集而过度拥塞，导致连线绕行和关键路径。框架随后迭代地降低那些拥塞严重或关键路径密度较高槽位的限制，促使其利用率向全局平均值回归，同时确保其他槽位有足够的预算用于模块迁移。

在实践中，当无用户输入时，全局限制以 [2] 中的建议值为起点。我们据此初始化槽位级限制：最大限制 $R_{s,t}^{+}$ 从 R_t^{+} 开始，而最小限制 $R_{s,t}^{-}$ 则可从较高的 $\gamma \cdot \bar{u}$（其中 $\bar{u}$ 为平均利用率）起步，以加速收敛。搜索以步长 δ 推进，直到最大/最小限制分别收敛到 $\bar{u}+\theta$、$\bar{u}-\theta$，其中 θ 为围绕均值的阈值。若 ILP 求解器超出时间限制或出现实现失败，则提前终止。

由于流水化取决于布局规划确定的模块位置，资源限制探索先于流水化动作执行，后者复用最细可行的器件抽象方案及其优化后的资源限制。

引入槽位级最大限制 $R_{s,t}^{+}$ 扩展了解空间，但也让布局规划的收敛更具挑战性。图 10 展示了五个模块映射到 2×2 网格的场景，其中 I/O 模块（灰色）预分配到带有内存端口的槽位。仅使用全局最大限制 R_t^{+} 时，要达到均衡状态，R_t^{+} 必须落在 $[0.6, 0.7)$ 内，这是一个在较大 DSE 步长下很容易错过的狭窄区间。另一种做法是为 X0Y0 和 X1Y1 设置槽位级最大限制 $[0.4, 0.7)$，也能达到相同的均衡效果。若没有这些约束，任何 $R_t^{+} \in [0.7, 1.0]$ 都会因最小化穿越比率这一目标而导致 X0Y1 闲置。然而，当我们检测到 X0Y1 利用不足并引入最小限制后，设置 $R_{X0Y1,t}^{-} \in (0, 0.3]$ 可强制将一个模块移入 X0Y1，而 $R_{X0Y1,t}^{-} \in (0.3, 0.6]$ 则可实现理想均衡。这表明，同时搜索全局/槽位级与最大/最小限制，有助于实现更快、更稳健的布局规划收敛。

6.3 流水化与布线动作

动作 4 – 调节边界容量：在解决槽位利用率不均衡的问题时，另一个有效策略是管理每个边界 b 上的线容量 C_b。较高的边界容量允许更多经过流水化处理的连接穿过相应边界。如果检测到跨边界拥塞，降低其容量可以鼓励布线绕行至其他较不拥塞的边界，从而缓解局部压力并改善整体可布线性。图 11 展示了一个初始方案，共有 25 次穿越，且 FPGA 下半部分拥塞严重。为均衡槽位利用率，可以对槽位 X0Y1 和 X1Y1 施加 $(0.3, 0.5]$ 的最小限制。另一种方法是降低高拥塞边界处的线容量（指标 (B)）。将槽位 X0Y0 和 X1Y0 之间的容量降低到 $[0, 10)$，虽然总穿越数有所增加，但通过将穿越重路由到上方边界，能够实现更好的跨槽位均衡。算法 2 详述了基于拥塞的容量缩减，以及当利用率接近最大穿越比率时对芯粒穿越容量进行的迭代调整。该线容量优化先于其他流水化动作执行，以建立更好的初始布线条件；若延后执行，则可能打乱此前已经搜索出的结果。

Algorithm 1 槽位级资源限制调整

Require: 指标 M，先前限制 R_{old}（表 1 中的 $R_{s,t}^{+/-}$）
Ensure: 更新后的限制 R_{new}

初始化空的槽位级限制 R^{+}, R^{-}
从 M 中加载关键路径 P、槽位拥塞面积 C_s、LUT 利用率 u
$\bar{u} \leftarrow$ 各槽位 LUT 利用率均值；$n \leftarrow$ 每轮调整的槽位数
for all 关键路径 $p \in P$，直到 $|R^{+}| \geq n$ **do**
  for all 路径 p 经过的槽位 s，且 $u(s) > \bar{u}$ **do**
    $R^{+}(s) \leftarrow \max((1+\theta)\bar{u},\ u(s) - \delta)$
  end for
end for
for all 按 C_s 排序的前 n 个拥塞槽位 s **do**
  if $R^{+}(s)$ 已存在 **then**
    $R^{+}(s) \leftarrow R^{+}(s) - \delta$
  else
    $R^{+}(s) \leftarrow \max((1+\theta)\bar{u},\ u(s) - \delta)$
  end if
end for
for all 利用率最低的 n 个槽位 s **do**
  $R^{-}(s) \leftarrow \min((1-\theta)\bar{u},\ \max(u(s), \gamma\bar{u}) + \delta)$
end for
return 将 R^{+}, R^{-} 与 R_{old} 合并得到 R_{new}

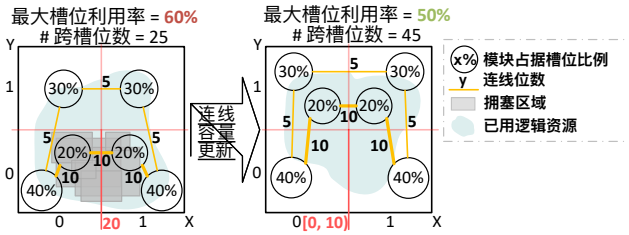

图 11: 槽位边界处的线容量管理，以改善跨槽位的资源利用率均衡。

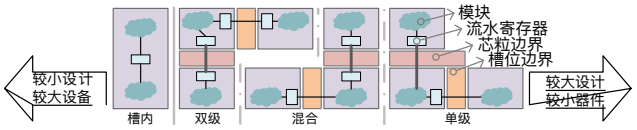

图 12: 从高（槽内）到低（跨槽位单级）密度的各种流水化方案。

动作 5 – 调整流水密度：在已有优化布局规划和线容量的基础上，流水寄存器插入策略仍会深刻影响时序质量。关键问题随之产生：同一芯粒内跨槽位的连线应使用一级流水，还是两级更为合适？跨芯粒连线又该如何处理？流水级不足可能无法

Algorithm 2 线容量调整

Require: 指标 M，先前容量配置 C_{old}
Ensure: 更新后的器件及调整后的线容量 C_{new}

```
初始化空的容量更新 C_new
从 M 中加载拥塞的相邻槽位对及穿越利用率 u
for all 拥塞槽位对 (s1, s2) do
    usage ← s1 与 s2 之间的穿越计数
    C_new(s1, s2) ← usage · α/max_crossings {α: 收缩比}
end for
for all 槽位对 (s1, s2) do
    if (s1, s2) ∉ C_new 且 C_old(s1, s2) · β < u(s1, s2)/u_max then
        C_new(s1, s2) ← C_old(s1, s2) · β {β: SLL 收缩比}
    end if
end for
return 具有更新容量 C_new 的器件
```

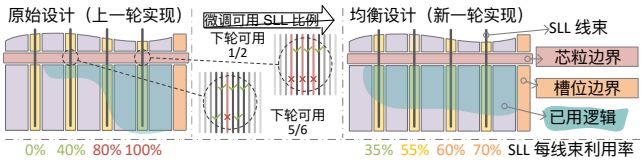

图 13: 调整芯粒穿越线束中的 SLL 可用比例，以实现水平均衡的布局布线结果。

消除关键路径，而过度流水化则可能因引入额外布线拥塞而适得其反。不同于先前采用固定流水密度，或针对特定设计依赖经验性深流水线的 HLPS 工作 [10, 17, 21]，我们自动选择四种不同密度的流水方案 d（图 12），由拥塞（指标 (B)）和关键路径（指标 (C)）引导。槽内方案通过在槽位内也添加寄存器来最大化流水化；双级方案仅对槽位穿越应用两级流水化；混合方案除芯粒边界外均采用一级；单级方案采用最少的流水化，即每次穿越插入一个寄存器。对于关键路径密集且拥塞严重的大型设计，我们采用混合或单级方案来降低流水密度。对于关键路径与拥塞关联较弱的小型设计，我们则探索双级和槽内方案，以更充分地利用丰富的布线资源。

该动作排在线容量调优之后，因为流水线插入影响跨槽位的逻辑（FF）密度。若提前执行，每次线容量调整都会导致槽位间更剧烈的资源波动，可能破坏搜索过程的稳定性。这一排序确保了渐进且稳健的 DSE 改进。

动作 6 – 均衡 SLL 分布：图 13 展示了 UltraScale+ FPGA 中芯粒穿越的架构示意。SLL 以专用的芯粒穿越线束形式水平分布，支持不同芯粒上逻辑之间的连接。这些跨芯粒连线密度远低于芯粒内连线[2]，且线束间利用率变化显著。图 13（左）显示，由于布局倾向，靠近纵向槽位边界的线束利用率更高，而边缘线束利用不足。这种失衡分布反映出布局器的局限，迫使布线器绕经更远的线束，常常会恶化时序质量。因此，这类问题需要在布局之前通过 SLL 可用性控制主动加以处理。

我们基于先前实现中的利用率（指标 (D)）来调整每个线束 i 的 SLL 可用比例 l_i。如图 13（中）所示，我们降低先前满负荷线束的 l_i，同时提高利用不足线束的 l_i。这一布线资源调整会引导布局器在各线束间更均匀地分配逻辑单元，从而改善整体时序。经过逐线束调优后，所有 l_i 值还会根据整体设计规模（指标 (A)）进一步缩放。较低的比例可在小型设计中强制执行更激进的重路由，但有布局失败的风险；较高的比例则为大型设计提供更保守但更稳定的改善。经过新一轮布局布线后，SLL 利用率在线束间更加均衡，如图 13（右）所示。该 SLL 均衡被放在 DSE FSM 的最终状态，因为其有效性取决于上游的布局规划和流水化决策。

[2]AMD/Xilinx UltraScale+ 器件中，相邻两个 SLR 之间通常有 16 个 SLL 列，每列包含 1440 根跨芯粒连线。

6.4 完整的探索代理：指标到动作的集成

基于上述拓扑依赖关系，我们的 DSE 框架通过图 5 所示的有限状态机（FSM）协调指标与动作。首先，动作 1 优化预置 IP 绑定。然后，在默认的 SLR 级抽象下，动作 3 优化全局和槽位级资源限制以均衡利用率，直到继续调整会导致频率下降或出现不可行方案。接着，动作 2 细化器件网格化，每个可行粒度都会启动新一轮资源限制搜索。基于达到最高频率的网格，动作 4 调节边界容量以均衡线分布。随后，基于当前频率最高的方案，动作 5 结合设计规模和拥塞模式调整流水密度。最后，动作 6 均衡 SLL 分布以优化芯粒穿越利用率。

尽管也可能采用其他动作排序，但我们面临来自 PnR 工具的严峻运行时挑战，大型设计的单次 PnR 迭代往往超过十小时。这限制了我们穷举探索设计空间的能力。虽然无法保证理论上的全局最优，但我们的 DSE 框架具有很强的实用性和工程价值，能为 HLPS 社区带来切实收益，并使设计者摆脱繁琐的调参过程。改进效果和执行时间的详细分析见第 7 节。

6.5 自动化 DSE

我们的 DSE 及其底层 HLPS 思想借助器件无关的模型和核心动作，可较好地推广到不同 FPGA 厂商。该框架提供自动化优化，在标准 Vitis 流程之外仅需极少的用户干预。我们提供覆盖数十款主流多芯粒 FPGA 的器件库，只需一行配置代码即可自动生成器件规格。表 1 中的参数会从综合后资源估计给出的初始值开始自动探索，同时也保留人工干预以加速收敛的可能性。框架的稳健性由 DSE FSM 的中间状态表示来保证，支持在用户暂停或系统中断后恢复。虽然各动作按顺序执行，但每个动作的搜索空间都允许通过并发实现作业进行部分并行探索，并可根据可用系统内存调整并行规模。

7 实验与结果

7.1 基准测试与实验设置

我们在 AutoBridge [12] 基础上实现了 HLPS 核心，并通过定制的 RapidWright 程序（指标 (A)、(D)、(E)）和 Vivado Tcl 脚本（指标 (B)、(C)）提取指标。我们的评估覆盖了来自多个领域的六个大规模 HLS 加速器，每个设计具有独特的架构挑战和约束。

- **Sextans [30]**：部署在 Alveo U55C 上的 SpMM 设计，利用 29 个 HBM 内存控制器，资源利用率**较高且分布较均衡**。
- **Callipepla [31]**：一个使用 32 个 HBM 通道中 26 个的共轭梯度求解器，用于考察本框架可达到的最大频率上限。
- **MM 10x13 [34]**：部署在 Alveo U55C 上的矩阵乘法脉动阵列，利用 **90% LUT**，作为高密度压力测试。
- **Serpens [29]**：部署在 Alveo U280 上的稀疏矩阵–向量乘法（SpMV）实现，通过 32 个 HBM 通道（256 位）和 2 个 DDR 接口（各 512 位）进行**密集内存访问**。

表 4: 评估基准测试的资源统计

基准	器件	# 芯粒	BRAM	DSP	FF	LUT	URAM
Sextans	U55C	3	58%	32%	21%	54%	54%
Callipepla	U55C	3	29%	18%	15%	37%	40%
MM 10x13	U55C	3	30%	38%	40%	90%	0%
MM 10x13	U250	4	22%	29%	32%	69%	0%
Serpens	U280	3	58%	17%	17%	38%	54%
NTT	U280	3	18%	16%	13%	38%	0%
GPT-2 M.	VHK158	2	25%	35%	22%	51%	32%

注：资源百分比相对于用户可用的总逻辑计算，不含基础设施 IP 和内存控制器。

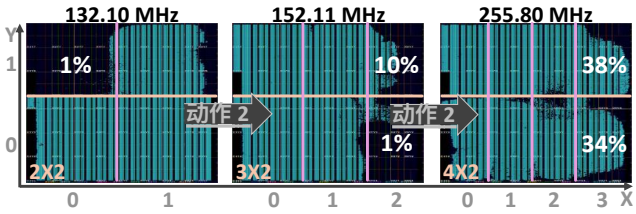

图 14: GPT-2 Medium 在 Versal VHK158 上通过网格细化（2×2、3×2 到 4×2）的渐进式资源均衡。

- **NTT [5]**：一个数论变换设计，代表一种天然较粗粒度的架构，其**主导模块**为多层蝶形网络，占据 Alveo U280 上约 1/3 个芯粒的面积。
- **GPT-2 Medium [13]**：一个具有**复杂控制流**的 Transformer 推理引擎，部署在 Versal VHK158 上；其两个芯粒在本文考察的器件中尺寸最大。

7.2 逐动作案例研究

为展示第 6 节中各动作的有效性，我们分析了七个基准（表 4）上系统化 HLPS DSE 的代表性迭代。每个案例研究直观验证特定动作的优化效果。

动作 1 – 优化 IP 绑定：该动作的可视化较为直观，为简洁起见省略。

动作 2 – 细化网格粒度：更细网格化的动机源于各槽位间不均衡的逻辑利用率。GPT-2 Medium 在 VHK158 上是一个理想的案例研究，其较大的芯粒尺寸提供了充足的空间来展示子芯粒级网格化如何有效将逻辑推向器件边缘，实现整个 FPGA 上更好的资源利用率均衡。图 14（左）显示 VHK158 上带一条纵向切割的初始 2×2 网格在 80% LUT 利用率限制下导致分布不均，槽位 X0Y1 几乎空闲。鉴于该器件的时钟区域排列（10×7），纵向切割对逻辑分布的影响更强。省略中间水平切割结果，我们考察带有纵向切割的 3×2 和 4×2 配置。4×2 网格通过匹配模块粒度实现更好的槽位级均衡，而 3×2 变体在相同利用率约束下在槽位 X2Y0/X2Y1 中分布不足。

动作 3 – 控制资源限制：高度模块化且粒度较细的加速器设计通常可以在各槽位间灵活进行布局规划，但具有硬约束的模块（如 HBM 内存控制器）除外。图 15 展示了 Sextans 在 Alveo U55C 上的挑战，29 个内存控制器沿 SLR0 底部边缘产生显著资源压力。在布局规划阶段简单地扣除内存控制器资源并不能解决时序问题。图 15 左侧揭示了后果：拥塞和关键路径集中在 SLR0。在动作 3 的逐步搜索中，持续降低 SLR0 的最大资源限制会不断改善时序质量，如图 15 右侧所示。SLR0 中大部分拥塞窗口被消除，尽管 SLR1 出现了轻微拥塞（因为先

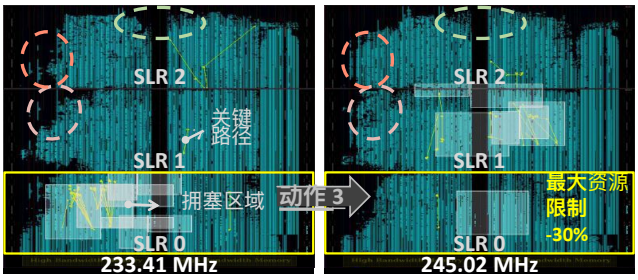

图 15: 通过将 SLR0 的最大资源限制降低 30% 实现的 Sextans（Alveo U55C, 2×3）模块重分配。白色方框标示最拥塞区域；黄色箭头指示关键路径。

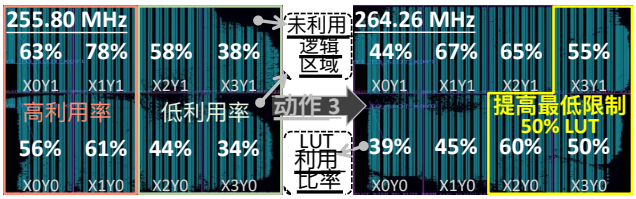

图 16: 通过对槽位 X2Y0、X3Y0、X3Y1 施加 50% 最小 LUT 利用率限制改善的 GPT-2 Medium（Versal VHK158, 4×2）逻辑分布。

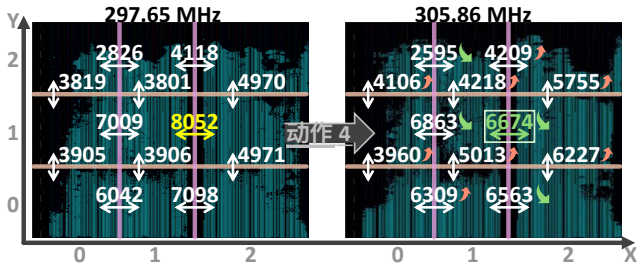

图 17: 通过降低边界 X1Y1–X2Y1 处线容量驱动的 Callipepla（Alveo U55C, 3×3）均衡线分布。

前无拥塞的区域没有施加限制控制）。椭圆框标示了成功迁移到 SLR1 和 SLR2 边缘区域的模块，而流水化保证了这些区域的时序收敛。这些优化后的 SLR0 限制可作为后续迭代中防止内存控制器引起拥塞的可复用参考。

作为最大限制控制的补充，图 16 展示了最小资源约束的有效性，对比了 GPT-2 Medium 在 VHK158 上 4×2 网格中添加最小约束前后的槽位级资源分布。优化前利用率不均，左半部分 LUT 使用拥挤，而右侧槽位利用不足。通过对槽位 X2Y0、X3Y0 和 X3Y1 施加最小 LUT 利用率约束，框架实现了更均衡的模块分布，最大频率从 255.80 MHz 提升至 264.26 MHz。

动作 4 – 调节边界容量：图 17 揭示了 Callipepla 中不均衡的槽位穿越分布：槽位 X1Y1 的东边界承载 8,052 根线——是其南北穿越的 2 倍，尽管整体拥塞较低。为解决这一不均衡，算法 2 中的第二个 **for** 循环会降低高利用率边界的容量。在探索过程中，将该边界容量下调 15% 后，可有效迫使部分连线经由利用不足的边界绕行。容量降低不仅能通过在槽位 X1Y1 的各

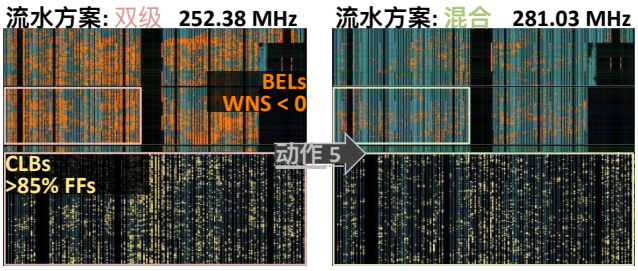

图 18: 通过流水方案优化（双级到混合）支持的 MM 10x13 (Alveo U55C, 3 × 3）逻辑密度缓解。

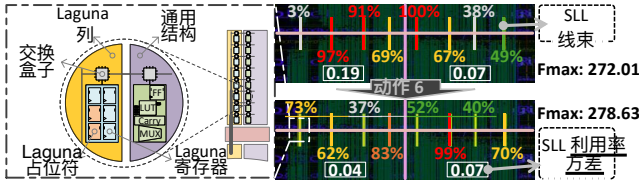

图 19: 在 Sextans 的 Alveo U55C（3 × 3）上应用 Laguna 占位符进行 SLL 均衡，聚焦于 SLR0–SLR1 边界的左半部分。

边界上更均匀地分散连线来均衡穿越分布，还能在器件上形成更合理的分布式布线路径，从而改善时序质量。

动作 5 – 调整流水密度：图 18 展示了流水密度对 MM 10x13（Alveo U55C, 3 × 3）的影响，这是一个具有极端 LUT 利用率的细粒度设计。黄色点表示 FF 利用率超过 85% 的可配置逻辑块（CLB），橙色点标示建立裕量为负的基本逻辑元素（BEL）。从双级切换到混合方案后，负裕量 BEL 的分布更稀疏，流水寄存器也更少。尽管 HLPS 流水化通常仅带来极小的资源开销 [12]，但在这种高利用率场景下，插入 FF 仍会显著影响时序。然而，流水密度与时序之间的关系在高利用率设计中是非单调的：从双级降到混合后，频率从 252.38 MHz 提升至 281.03 MHz；但进一步降到单级则退化至 246.71 MHz。这种非线性行为证明了渐进探索的必要性，而不能仅依据利用率水平套用固定方案。

动作 6 – SLL 均衡：我们以 Sextans 在 Alveo U55C 上为例展示 SLL 均衡的实现细节和有效性。U55C 在三个纵向堆叠芯粒的每个芯粒边界上具有 16 个 SLL 线束。图 19 展示了使用 3 × 3 抽象方案时，SLR0 和 SLR1 之间左半部分边界的情况（中间有一个纵向切割）。上方部分揭示了默认 Vitis 实现中显著的 SLL 利用率不均衡，部分 SLL 线束在布局后的估计利用率超过 100%，而其他线束利用不足。为控制 SLL 可用性，我们实现了基于占位符的机制，利用线束端点的 Laguna TX/RX 寄存器。基于先前实现中的利用率，我们用虚拟寄存器占据不同比例的 Laguna 对：利用率在 [90%, 100%] 的线束占据 2/3，[75%, 90%) 的占据 1/2，[60%, 75%) 的占据 1/3，[40%, 60%) 的占据 1/6。这些占位符在逻辑布局前通过虚拟连线相连。布局完成后，在布线前移除这些占位符单元和连线。如图 19 所示，这些寄存器为穿越线束提供了专用且经过优化的连接。最终实现在线束利用率上更为均衡，方差值也有所降低（见白色方框）。通过主动向布局器显式传达各 SLL 线束可用容量下降这一信息，该方法促进了水平方向的模块分布并缓解局部拥塞，从而改善时序质量。

7.3 端到端时序收敛

表 5 总结了所有测试案例的端到端评估，与最先进的基线进行比较。实验运行在配备 256GB DDR4-3200 内存的 AMD EPYC 7742 服务器上。

基线：我们评估了两个基线：商业 AMD Vitis 和学术 AutoBridge [12]，两者均保留参考来源中的内存端口绑定。Vitis（无 HLPS）实现采用最高级别的 `-O3` 优化和 `Explore` 策略。目标频率从 300 MHz 开始，以 15 MHz 为步长递减，直到成功完成 PnR，以避免过于激进的约束导致失败。AutoBridge 应用 70%–90% 的最大全局资源限制，并按设计规模缩放以确保资源充足。

最大频率：表 5 中的时序结果遵循图 5 的动作顺序，其中每一列包含所有前序动作的累积效果。例如，动作 2 的结果反映的是完整流程中子 FSM "动作 1 → 动作 3 → 动作 2 "的累计执行效果。

Sextans 通过所有动作展示了渐进的时序改善。初始 29 个 HBM 内存控制器导致 SLR0 拥塞，这一问题部分由动作 1 中基于拥塞引导的内存端口重分配所缓解。动作 3 进一步将 SLR0 的最大限制降低至多 30%，达到 248.92 MHz，但仍受限于初始 2 × 3 抽象。动作 2 过渡到 3 × 3 网格，使逻辑更好地分布到器件边缘，并促成新一轮资源限制调优。DSE 识别出左上区域的利用不足槽位（X0Y1、X0Y2、X1Y2），通过施加约 38% 的最小利用率约束，最终收敛到 279.62 MHz。动作 4 通过降低 X1Y0–X2Y0 和 X1Y1–X2Y1 边界的容量来解决高流量槽位穿越问题，有效缓解了水平方向的拥塞。鉴于该设计的高利用率，动作 5 发现混合流水方案比双级和单级方案都更有效，既不会引入过多 FF，也不会造成跨槽位时序问题。最终，动作 6 施加高层级 SLL 均衡，最大频率提升至 313.48 MHz。

其他设计因其独特特征展现了不同的时序收敛模式。**Serpens** 在所有 HBM 和 DDR 端口上的内存控制器均处于满载状态，主要受益于动作 2 中对 SLR0/SLR1 最大限制的降低以缓解拥塞。鉴于其低利用率，细粒度的资源限制和线容量搜索效果有限，而槽内流水化（动作 5）带来更显著的改善。这一槽内方案同样使 **Callipepla** 受益。然而，Serpens 的 DDR 内存控制器位于 SLR0/SLR1，限制了 SLL 均衡的效果；而 Callipepla 因 HBM 内存控制器较少且无 DDR，通过动作 6 取得了最高频率。

MM 10x13 在 Alveo U55C 上的极端 LUT 利用率（90%）使混合流水方案特别有效。高利用率要求 3 × 3 或更细的抽象，最优方案达到近乎完美的均衡（每槽位 89%–90% LUT）——任何线容量调整都会使部分槽位过度拥塞。而在更大的 Alveo U250 上，动作 4 有效减少了 SLR2 右半部分周围的密集槽位穿越——该区域因 SLR2/SLR3 的 DDR 内存控制器和 SLR1 不可用的右半部分而容易拥塞。

NTT 展示了当软件抽象导致 HLS 工程师物理感知不足时 HLPS DSE 对欠优化设计的有效性。它具有一个本可在 HLS 层面更好分区的大型模块。该模块被约束在 Alveo U280 2×3 网格的 X0Y1/X0Y2 槽位中，在任何内存、布局规划或 SLL 分布调整下都会导致 ILP 失败。尽管存在这些限制，框架识别出不重叠的时序瓶颈：X0Y0 的 HBM 内存控制器拥塞窗口和 X0Y1–X1Y1 边界的跨芯粒关键路径。这种空间分离使动作 5 能有效应用双级和槽内方案，将最大频率提升至 310.95 MHz。

GPT-2 Medium 具有独特的 NoC 端口绑定，且由于芯粒较大，需要更细的器件网格。关键改善出现在水平切割将其高而窄的芯粒中纵向延伸的连线分段时。该设计在采用 4 × 4 网格，并对最右侧槽位进行渐进式最小限制搜索后，达到峰值性能。

平均而言，我们的 HLPS DSE 分别达到 Vitis 和 AutoBridge 基线频率的 2.42 倍和 1.67 倍。

7.4 执行时间

为应对运行时挑战，我们的框架通过并行 HLPS DSE 缓解 PnR 开销。受限于可用系统内存，在表 5 所示的实验中，每个动作支持每批最多 4 个并行 Vitis 作业，实际并行度由搜索依赖关系决定。收敛模式因设计而显著不同。例如，Sextans 执行一个 2×4 和五个 3×3 方案（均扩展自 2×3）分两个并行批次，而 4×3 和 3×4 配置因依赖 3×3 结果需要额外一个批次。反直觉的是，小规模的 Callipepla 需要更多动作 2 迭代，因为其从 SLR0/SLR1 到 SLR2 的渐进扩展由较低的最小约束驱动。相比之下，MM 10x13 在 U55C 上因高初始最小约束而快速收敛，仅需对槽位 X0Y2 施加 85% 的约束。Serpens 因完全的内存利用率而跳过动作 1，NTT 的大型模块则自然约束了物理设计空间。GPT-2 Medium 由于采用最细的网格粒度，需要最广泛的槽位级限制调优。

尽管 DSE 过程可能持续数天，且对设计空间的覆盖仍然有限，我们的方法相较于手动调优依然展现出显著的实用价值。硬件工程师的反馈一致表明，他们愿意接受更长的运行时间以换取更高的频率。未来一个可能的方向，是从设计子集的部分布局布线中提取物理反馈，以缩短整体运行时间，但这超出了本文的研究范围。

8 相关工作

8.1 加速器设计中的时序优化

加速器时序优化方面的研究已经通过多种技术取得了显著进展。早期工作聚焦于流水化策略 [10, 17, 21]，尤其关注跨芯粒场景带来的挑战。通过利用 FPGA 原语内置寄存器及其天然优化的互连 [17, 36]，这些方法的有效性得到进一步增强。随着设计日趋复杂，研究者开始探索布局规划技术 [14, 16, 27]，以降低 SLL 利用率并缓解拥塞。近来，研究进一步推进到前端设计与后端实现之间的迭代优化 [9, 22]，并通过集成多种方法取得了显著成果 [9]。然而，这些方法通常面向较小规模的设计，且需要大量手动定制工作，限制了其在大规模加速器设计中的适用性。

8.2 高层次物理综合

物理感知的 HLS 优化近年取得了显著进展。AutoBridge [12] 率先引入了基于 ILP 的布局规划和固定布线方案的 HLPS 技术，以最小化芯粒/槽位穿越。在此基础上，后续工作 [7, 8, 11, 18, 28] 将 HLPS 方法论扩展到 HLS 编译器优化，纳入了缓冲区管理和任务并行加速 API。HLPS 的研究范畴进一步拓展，多项工作 [2, 23, 25, 26, 28, 35] 探讨了多样的多芯粒和多 FPGA 分区挑战。尽管这些方法取得了显著成果，但它们主要关注单一优化方面，缺乏全面的物理反馈。早期的物理感知优化尝试要么面向较小规模的设计 [37]，要么仅关注线长优化 [24]。即使在具有硬件 NoC 的 Versal FPGA 等现代架构上，最近的研究 [20] 仍强调高效利用 SLL 的重要性。尽管取得了诸多进展，HLPS 技术尚未在物理指标的引导下被系统性地编排和探索。

9 结论

本工作提出了首个面向高层次物理综合（HLPS）的自动化设计空间探索框架。我们识别出一组具有代表性的物理指标，用于评估 HLPS 方案质量，并据此开发了简洁而有效的启发式策略，以搜索能够实现时序收敛的 HLPS 参数配置。通过在六个大规模设计和四种不同多芯粒 FPGA 上进行评估，我们的工具取得了显著的性能提升，平均频率达到 311.06 MHz，分别是 AMD Vitis/Vivado 工具链（128.48 MHz）和领先学术方案（186.21 MHz）的 2.42 倍和 1.67 倍。

参考文献

[1] Advanced Micro Devices, Inc. 2023. 使用 SSI 技术的 Versal 器件（ug1304）（Versal Devices Using SSI Technology (ug1304)）. https://docs.amd.com/r/en-US/ug1304-versal-acap-ssdg/Versal-Devices-Using-SSI-Technology.

[2] Tomas Alonso, Lucian Petrica, Mario Ruiz, Jakoba Petri-Koenig, Yaman Umuroglu, Ioannis Stamelos, Elias Koromilas, Michaela Blott, and Kees Vissers. 2021. Elastic-DF：通过自动分区扩展 FPGA 云中 DNN 推理性能（Elastic-DF: Scaling Performance of DNN Inference in FPGA Clouds through Automatic Partitioning）. *ACM Trans. on Reconfigurable Technol. and Syst. (TRETS)* 15, 2 (2021), 1–34.

[3] Suhail Basalama, Atefeh Sohrabizadeh, Jie Wang, Licheng Guo, and Jason Cong. 2023. FlexCNN：面向 FPGA 上 CNN 加速器组合的端到端框架（FlexCNN: An End-to-End Framework for Composing CNN Accelerators on FPGA）. *ACM Trans. on Reconfigurable Technol. and Syst.* 16, 2 (2023), 1–32.

[4] Yuze Chi, Licheng Guo, and Jason Cong. 2022. 面向幂律图的单源最短路径加速（Accelerating SSSP for Power-Law Graphs）. In *Proc. ACM/SIGDA Int. Symp. on Field-Program. Gate Arrays (FPGA)*. 190–200.

[5] Young-kyu Choi. 2024. AutoNTT：面向 FPGA NTT 加速器的代码生成器（AutoNTT: Code Generator for FPGA NTT Accelerator）. https://github.com/applesforme/AutoNTT.

[6] Young-kyu Choi, Yuze Chi, Jason Lau, and Jason Cong. 2022. TARO：面向 FPGA 高层次综合中无阻塞内核的自动优化（Taro: Automatic Optimization for Free-Running Kernels in FPGA High-Level Synthesis）. *IEEE Trans. on Comput.-Aided Des. of Integr. Circuits and Syst.* 42, 7 (2022), 2423–2427.

[7] Linfeng Du, Tianqi Liang, Sharad Sinha, Zhiyao Xie, and Wei Zhang. 2023. FADO：面向多芯粒 FPGA 高层次综合设计的布局感知指令优化（Fado: Floorplan-Aware Directive Optimization for High-Level Synthesis Designs on Multi-Die FPGAs）. In *Proc. ACM/SIGDA Int. Symp. on Field Program. Gate Arrays (FPGA)*. 15–25.

[8] Linfeng Du, Tianqi Liang, Xiaofan Zhou, Jinming Ge, Shulin Li, Sharad Sinha, Jieru Zhao, Zhiyao Xie, and Wei Zhang. 2024. FADO：基于综合与解析模型的面向多芯粒 FPGA 高层次综合设计的布局感知指令优化（Fado: Floorplan-Aware Directive Optimization Based on Synthesis and Analytical Models for High-Level Synthesis Designs on Multi-Die FPGAs）. *ACM Trans. on Reconfigurable Technol. and Syst.* (2024).

[9] Yixiao Du, Yuwei Hu, Zhongchun Zhou, and Zhiru Zhang. 2022. 使用 HLS 在配备 HBM 的 FPGA 上实现高性能稀疏线性代数：以 SpMV 为例（High-Performance Sparse Linear Algebra on HBM-Equipped FPGAs Using HLS: A Case Study on SpMV）. In *Proc. ACM/SIGDA Int. Symp. on Field-Program. Gate Arrays (FPGA)*. 54–64.

[10] Ermusic Fournier, Ciprian Teodorov, and Loïc Lagadec. 2021. Carnac：面向 FPGA 上快速群体验证的算法可变性（Carnac: Algorithm Variability for Fast Swarm Verification on FPGA）. In *Int. Conf. on Field-Program. Log. and Appl. (FPL)*. 185–189.

[11] Licheng Guo, Yuze Chi, Jason Lau, Linghao Song, Xinheng Tian, Moazin Khatti, Weikang Qiao, Jie Wang, Ecenur Ustun, Zhenman Fang, et al. 2023. TAPA：面向现代 FPGA 的可扩展任务并行数据流及高层次综合与物理结构协同优化的编程框架（TAPA: A Scalable Task-Parallel Dataflow Programming Framework for Modern FPGAs with Co-Optimization of HLS and Physical Design）. *ACM Trans. on Reconfigurable Technol. and Syst.* 16, 4 (2023), 1–31.

[12] Licheng Guo, Yuze Chi, Jie Wang, Jason Lau, Weikang Qiao, Ecenur Ustun, Zhiru Zhang, and Jason Cong. 2021. AutoBridge：面向多芯粒 FPGA 高频高层次综合设计的粗粒度布局规划与流水线协同优化的自动化框架（AutoBridge: Coupling Coarse-Grained Floorplanning and Pipelining for High-Frequency HLS Design on Multi-Die FPGAs）. In *ACM/SIGDA Int. Symp. on Field-Program. Gate Arrays (FPGA)*. 81–92.

[13] Zifan He, Hritik Gupta, Hanrui Ke, and Jason Cong. 2025. IntRRA：面向 FPGA 上高效 Transformer 推理的任务间资源复用加速器（IntRRA: Inter-Task Resource-Repurposing Accelerator for Efficient Transformer Inference on FPGAs）. In *Proc. ACM/SIGDA Int. Symp. on Field-Program. Gate Arrays (FPGA)*. 44.

[14] Zifan He, Linghao Song, Robert F. Lucas, and Jason Cong. 2024. LevelST：面向稀疏三角求解器的基于流的加速器（LevelST: Stream-Based Accelerator for Sparse Triangular Solver）. In *Proc. ACM/SIGDA Int. Symp. on Field Program. Gate Arrays (FPGA)*. 67–77.

[15] Yuwei Hu, Yixiao Du, Ecenur Ustun, and Zhiru Zhang. 2021. GraphLily：在配备 HBM 的 FPGA 上加速图线性代数运算（GraphLily: Accelerating Graph

表 5: 按图 5 拓扑顺序在特定动作完成后达到的最高频率（MHz）和累计耗时（小时）

设计	器件	Vitis		AutoBridge		动作 1		动作 3		动作 2		动作 4		动作 5		动作 6	
		频率	T	频率	T	频率	T(n,p)	频率	T(n,p)	频率	T(n,p)	频率	T(n,p)	频率	T(n,p)	频率	T(n,p)
Sextans	Alveo U55C	101.72	16	182.05	16	233.42	33(7,2)	248.92	67(6,2)	279.62	119(8,3)	291.72	152(6,2)	310.46	170(2,1)	**313.48**	187(2,1)
Callipepla	Alveo U55C	149.74	10	263.13	10	~~240.15~~	30(3,1)	271.64	52(8,2)	305.15	92(15,4)	310.49	113(5,2)	327.69	123(2,1)	**343.52**	132(3,1)
MM 10x13	Alveo U55C	89.42	14	159.10	14	184.29	30(5,2)	*N/A*	46(2,1)	252.38	61(4,1)	*N/A*	76(2,1)	**281.03**	92(1,1)	~~266.45~~	105(3,1)
MM 10x13	Alveo U250	118.28	18	*N/A*	19	142.68	20(4,1)	295.48	59(7,2)	~~181.29~~	82(2,1)	317.35	120(5,2)	**332.89**	138(2,1)	*N/A*	156(3,1)
Serpens	Alveo U280	174.88	10	244.80	11	*N/A*	10(1,1)	293.17	30(8,2)	308.06	71(11,4)	310.57	91(8,2)	**325.83**	101(2,1)	~~324.36~~	108(3,1)
NTT	Alveo U280	150.36	7	152.30	7	*N/A*	8(4,1)	195.62	15(4,1)	*N/A*	23(2,1)	251.87	39(5,2)	**310.95**	46(2,1)	*N/A*	51(3,1)
GPT-2 M.	Versal VHK158	114.93	19	115.90	19	132.10	61(12,3)	229.20	101(7,2)	255.80	233(19,6)	264.26	291(10,3)	**269.75**	310(2,1)	*N/A*	325(3,1)

频率（MHz）——~~数字~~：该步骤中达到的最优结果，但先前动作已产生更优结果；"*N/A*"：该步骤中未找到成功的 Vitis 实现。

T(n,p)（小时 (方案数, 批数)）——"T"：该动作完成时的累计耗时；"n"：该动作中搜索的方案数；"p"：为这 n 个方案执行的并行 Vitis 实现批数。

Linear Algebra on HBM-Equipped FPGAs）. In *IEEE/ACM Int. Conf. On Comput. Aided Des. (ICCAD)*. 1–9.

[16] Aman K. Jain, Chandrasekhara Ravishankar, Hamid Omidian, Sanjay Kumar, Mihir Kulkarni, Anshul Tripathi, and Dinesh Gaitonde. 2023. 面向 FPGA 上稀疏矩阵-向量乘法的具有弹性的模块化精简架构（Modular and Lean Architecture with Elasticity for Sparse Matrix Vector Multiplication on FPGAs）. In *IEEE Int. Symp. on Field-Program. Custom Comput. Machines (FCCM)*. 133–143.

[17] Sahand Kashani, Mahyar Emami, Keisuke Kamahori, Sahand Pourghannad, Ritik Raj, and James R. Larus. 2024. 面向 RTL 仿真的 475 MHz 众核 FPGA 加速器（A 475 MHz Manycore FPGA Accelerator for RTL Simulation）. In *Proc. ACM/SIGDA Int. Symp. on Field Program. Gate Arrays (FPGA)*. 78–84.

[18] Moazin Khatti, Xinheng Tian, Yuze Chi, Licheng Guo, Jason Cong, and Zhenman Fang. 2023. PASTA：面向现代多芯粒 FPGA 上可扩展任务并行 HLS 程序的编程与自动化支持（PASTA: Programming and Automation Support for Scalable Task-Parallel HLS Programs on Modern Multi-Die FPGAs）. In *IEEE Int. Symp. on Field-Program. Custom Comput. Machines (FCCM)*. 12–22.

[19] Jason Lau, Yuanlong Xiao, Yutong Xie, Yuze Chi, Linghao Song, Shaojie Xiang, Michael Lo, Zhiru Zhang, Jason Cong, and Licheng Guo. 2024. RapidStream IR：面向 FPGA 高层次物理综合的基础设施（RapidStream IR: Infrastructure for FPGA High-Level Physical Synthesis）. In *Proc. IEEE/ACM Int. Conf. on Comput. Aided Des. (ICCAD)*. 1–11.

[20] Siyuan Liu, Jake Ke, Tony Nowatzki, and Jason Cong. 2025. 揭秘 FPGA 硬核 NoC 性能（Demystifying FPGA Hard NoC Performance）. *arXiv preprint arXiv:2503.10861* (2025).

[21] Michael Lo, Zhenman Fang, Jie Wang, Peng Zhou, Mau-Chung Frank Chang, and Jason Cong. 2020. 面向基因组分析工具包中 BQSR 加速的算法-硬件协同设计（Algorithm-Hardware Co-Design for BQSR Acceleration in Genome Analysis Toolkit）. In *IEEE Int. Symp. on Field-Program. Custom Comput. Machines (FCCM)*. 157–166.

[22] Alec Lu, Zhenman Fang, Weihua Liu, and Lesley Shannon. 2021. 通过微基准测试为软件程序员揭秘现代数据中心 FPGA 的存储系统（Demystifying the Memory System of Modern Datacenter FPGAs for Software Programmers through Microbenchmarking）. In *ACM/SIGDA Int. Symp. on Field-Program. Gate Arrays (FPGA)*. 105–115.

[23] Jingwei Luo, Xingchen Liu, Fangxin Chen, and Yajun Ha. 2023. HRFF：面向基于 NoC 的可扩展多芯粒 FPGA 的层次化递归布局规划框架（HRFF: Hierarchical and Recursive Floorplanning Framework for NoC-Based Scalable Multi-Die FPGAs）. *IEEE Trans. on Circuits and Syst. I: Regular Papers* (2023).

[24] Feng Mao, Wei Zhang, Bingsheng Feng, Bingsheng He, and Yuchun Ma. 2016. 面向基于中介层的多 FPGA 系统的模块化布局（Modular Placement for Interposer Based Multi-FPGA Systems）. In *Int. Great Lakes Symp. on VLSI (GLSVLSI)*. 93–98.

[25] Moein Mazraeli, Yuan Gao, and Paul Chow. 2023. 面向数据中心的大规模多 FPGA 应用分区（Partitioning Large-Scale, Multi-FPGA Applications for the Data Center）. In *Int. Conf. on Field-Program. Log. and Appl. (FPL)*. 253–258.

[26] Thanh Nguyen, Zachary Blair, Stephen Neuendorffer, and John Wawrzynek. 2023. SPADES：面向 Versal 可编程逻辑的高效设计流程（SPADES: A Productive Design Flow for Versal Programmable Logic）. In *Int. Conf. on Field-Program. Log. and Appl. (FPL)*. 65–71.

[27] Surya Kumar Prakash, Harsh Patel, and Nachiket Kapre. 2022. 利用 FPGA 覆盖 NoC 管理多芯粒 FPGA 上的 HBM 带宽（Managing HBM Bandwidth on Multi-Die FPGAs with FPGA Overlay NoCs）. In *IEEE Int. Symp. on Field-Program. Custom Comput. Machines (FCCM)*. 1–9.

[28] Neha Prakriya, Yuze Chi, Suhail Basalama, Linghao Song, and Jason Cong. 2024. TAPA-CS：在分布式 HBM-FPGA 上实现可扩展加速器设计（TAPA-CS: Enabling Scalable Accelerator Design on Distributed HBM-FPGAs）. In *Proc. ACM Int. Conf. on Architectural Support for Program. Languages and Operating Syst. (ASPLOS)*. 966–980.

[29] Linghao Song, Yuze Chi, Licheng Guo, and Jason Cong. 2022. Serpens：基于高带宽存储器的通用稀疏矩阵-向量乘法加速器（Serpens: A High Bandwidth Memory Based Accelerator for General-Purpose Sparse Matrix-Vector Multiplication）. In *Proc. ACM/IEEE Des. Automat. Conf. (DAC)*. 211–216.

[30] Linghao Song, Yuze Chi, Atefeh Sohrabizadeh, Young-kyu Choi, Jason Lau, and Jason Cong. 2022. Sextans：面向通用稀疏矩阵与稠密矩阵乘法（Sextans: A Streaming Accelerator for General-Purpose Sparse-Matrix Dense-Matrix Multiplication）. In *Proc. ACM/SIGDA Int. Symp. on Field-Program. Gate Arrays (FPGA)*. 65–77.

[31] Linghao Song, Licheng Guo, Suhail Basalama, Yuze Chi, Robert F. Lucas, and Jason Cong. 2023. Callipepla：面向加速共轭梯度求解器的流中心指令集与混合精度方案（Callipepla: Stream Centric Instruction Set and Mixed Precision for Accelerating Conjugate Gradient Solver）. In *Proc. ACM/SIGDA Int. Symp. on Field Program. Gate Arrays (FPGA)*. 247–258.

[32] Xinheng Tian, Zhifan Ye, Alec Lu, Licheng Guo, Yuze Chi, and Zhenman Fang. 2023. SASA：面向基于 HBM 的 FPGA 上优化混合时空并行的可扩展自动模板计算加速框架（SASA: A Scalable and Automatic Stencil Acceleration Framework for Optimized Hybrid Spatial and Temporal Parallelism on HBM-Based FPGAs）. *ACM Trans. on Reconfigurable Technol. and Syst.* 16, 2 (2023), 1–33.

[33] Abdul Wadood, Alec Lu, Kaiwen Zhang, and Zhenman Fang. 2024. FORC：面向大数据引擎中优化行列式文件解码器的高吞吐率流式 FPGA 加速器（FORC: A High-Throughput Streaming FPGA Accelerator for Optimized Row Columnar File Decoders in Big Data Engines）. In *Int. Conf. on Field-Program. Log. and Appl. (FPL)*. 318–324.

[34] Jie Wang, Licheng Guo, and Jason Cong. 2021. AutoSA：面向 FPGA 上高性能脉动阵列的多面体编译器（AutoSA: A Polyhedral Compiler for High-Performance Systolic Arrays on FPGA）. In *ACM/SIGDA Int. Symp. on Field-Program. Gate Arrays (FPGA)*. 93–104.

[35] Yuanlong Xiao, Daniel Park, Zhijing J. Niu, Aman Hota, and Andre Dehon. 2024. ExHiPR：面向快速增量式 FPGA 编译的扩展高层次部分重构（ExHiPR: Extended High-Level Partial Reconfiguration for Fast Incremental FPGA Compilation）. *ACM Trans. on Reconfigurable Technol. and Syst.* 17, 2 (2024), 1–28.

[36] Jialiang Zhang, Wei Zhang, Guojie Luo, Xing Wei, Yun Liang, and Jason Cong. 2019. 基于脉动阵列的 CNN 在 FPGA 上的频率提升（Frequency Improvement of Systolic Array-Based CNNs on FPGAs）. In *IEEE Int. Symp. on Circuits and Syst. (ISCAS)*. 1–4.

[37] Hao Zheng, Swathi T. Gurumani, Kyle Rupnow, and Deming Chen. 2014. 面向 FPGA 的快速高效布局布线导向高层次综合（Fast and Effective Placement and Routing Directed High-Level Synthesis for FPGAs）. In *Proc. ACM/SIGDA Int. Symp. on Field-Program. Gate Arrays (FPGA)*. 1–10.