arXiv:2301.00861v1 [cs.AR] 2 Jan 2023

用于支持粗粒度可重构阵列多任务执行的硬件抽象与硬件机制

Taeyoung Kong、Kalhan Koul、Priyanka Raina、Mark Horowitz 和 Christopher Torng

斯坦福大学

{kongty,kkoul,praina,horowitz,ctorng}@stanford.edu

摘要

领域专用加速器应用于各种计算系统，涵盖从边缘设备到数据中心的广泛场景。粗粒度可重构阵列（*CGRA*）处于现场可编程门阵列的灵活性与专用集成电路的高效性之间，代表了一种架构上的折中；它有望服务于同一应用领域内的多任务工作负载。然而，将多个任务调度到粗粒度可重构阵列上颇具挑战。粗粒度可重构阵列缺乏能够描述硬件资源的抽象，使工作负载调度器无法评估不同调度方案的性能、能耗和利用率。本文首先提出一种粗粒度可重构阵列架构，可灵活划分关键资源，包括全局缓冲存储器容量、全局缓冲存储器带宽和计算资源。划分后的资源作为硬件抽象，将编译与资源分配解耦。编译器利用这些抽象进行粗粒度资源映射，而调度器则在运行时借助它们灵活分配资源。随后，我们提出两种支持多任务执行的硬件机制。灵活形状执行区域可映射资源需求各不相同的多个任务，从而提高整体资源利用率。动态部分重配置（*DPR*）使粗粒度可重构阵列能够根据调度器快速做出的决策更新硬件配置。我们表明，这种抽象有助于将多任务工作负载自动、高效地调度到目标粗粒度可重构阵列上，并实现较高的利用率。与一次仅运行单个任务的基线粗粒度可重构阵列相比，在多任务云工作负载中，吞吐量提高了 *1.05* 倍至 *1.24* 倍，延迟降低了 *23%* 至 *28%*；在自主系统工作负载中，延迟降低了 *60.8%*。

1. 引言

近年来，领域专用加速器日益受到关注，因为与通用处理器相比，它们能够提供更高的性能和能效。专用集成电路 [8, 18,21]展现出最高的性能和能效，因为它们针对图像处理或机器学习（ML）等目标应用进行了专门设计。然而，专用集成电路设计流程可能持续数年，而且随着应用不断演进，固定功能加速器很快就会过时。一些研究将应用部署在现场可编程门阵列上 [12, 16, 17]。现场可编程门阵列能够重新配置底层硬件，并可加速多种工作负载，但其位级灵活性会带来较高的面积和能耗开销。粗粒度可重构阵列是介于专用集成电路和现场可编程门阵列之间的有前景的架构。粗粒度可重构阵列中的算术单元和路由系统可按字级粒度进行配置，从而以低于现场可编程门阵列的开销提供灵活性。凭借其独特优势，粗粒度可重构阵列可广泛应用于对性能、功耗和灵活性有较高需求的领域。

随着硬件加速器被部署到各种场景中，对硬件多任务执行支持的需求日益增长。例如，许多供应商 [21, 13]提供推理即服务，多个租户共享同一硬件来运行推理任务。此外，自主系统需要处理并发任务，以处理来自众多传感器的各种类型数据。一些研究已经探索了专用集成电路和现场可编程门阵列中的多任务执行支持。PREMA [11] 和 Planaria [14] 分别提出了一种通过时间复用和空间复用支持多租户的脉动阵列。 [35, 29, 34]提出了一种支持多租户的现场可编程门阵列虚拟化框架。然而，迄今为止，对粗粒度可重构阵列上的多任务执行支持的研究仍然较少。一个值得注意的例外是 ChordMap [27]，它将同步数据流图中描述的多个任务调度到粗粒度可重构阵列上执行。然而，该方法假设所有任务都预先已知；而在多租户云或多任务边缘工作负载场景中，任务可能会动态到达，需要调度器做出响应以最大限度地提高利用率。

遗憾的是，由于缺乏能够描述硬件资源的抽象，将多个任务调度到粗粒度可重构阵列上颇具挑战性。本文通过划分关键硬件资源，提出了粗粒度可重构阵列的硬件抽象。编译器和调度器都可以利用这些抽象来推理性能、能耗和利用率。

此外，我们还开发了硬件机制，使粗粒度可重构阵列能够快速、灵活地执行多任务；调度器可以利用这些机制提高硬件利用率。我们通过两种不同的多任务工作负载场景评估了粗粒度可重构阵列，以展示其潜力。我们的主要贡献如下：

- ① 我们提出了一种粗粒度可重构阵列架构，可灵活地重新划分关键资源，包括全局缓冲区（GLB）的内存容量、GLB 的内存带宽以及计算资源。具体而言，我们将 GLB 划分为全局缓冲区切片，并将瓦片阵列划分为阵列切片，作为硬件抽象。编译器利用这些抽象进行粗粒度资源映射，而调度器则利用它们灵活地分配资源。

- ② 我们提出了两种硬件机制，以支持粗粒度可重构阵列上的多任务执行。首先，粗粒度可重构阵列能够在运行时形成灵活形状的执行区域。通过使调度器能够灵活分配全局缓冲区切片和阵列切片，这种机制提高了资源利用率。其次，我们提出了一种快速DPR方法，可根据调度器的决策快速重配置底层硬件。该方法还支持在运行时将任务迁移到任意可用的阵列切片，无需软件干预。

- ③ 我们通过两个不同的示例量化了所提出机制的优势。与基线 CGRA 相比，在云系统场景中，我们的粗粒度可重构阵列采用灵活的执行区域和快速DPR，吞吐量提高了 1.05 至 1.24 倍，延迟降低了 23%–28%；在自主系统场景中，延迟降低了 60.8%。

2. 粗粒度可重构阵列的多任务执行架构支持

本节将探讨粗粒度可重构阵列实现多任务执行所需的架构支持。第2.1节首先介绍一种基线粗粒度可重构阵列架构，该架构具有许多可重构加速器普遍采用的特性 [7, 32, 15, 6, 1, 28]。第2.2 节随后介绍如何通过分别将全局缓冲区（GLB）和瓦片阵列划分为全局缓冲区切片和阵列切片，为调度器抽象粗粒度可重构阵列中的硬件资源。我们还在这些抽象的基础上，进一步设计了支持多任务执行的硬件机制（第2.3节），包括灵活形状执行区域和动态部分重配置（DPR）。

2.1. 基线粗粒度可重构阵列架构

我们的基线 CGRA 由包含处理单元（PE）瓦片和存储器（MEM）瓦片的瓦片阵列以及全局缓冲区（GLB）组成（图1）。我们采用了与 Amber SoC 中几乎相同的硬件配置 [7]。该 CGRA 包含 32x16 个瓦片，其中有 384 个处理单元瓦片和 128 个存储器瓦片，瓦片之间通过静态配置的网格互连

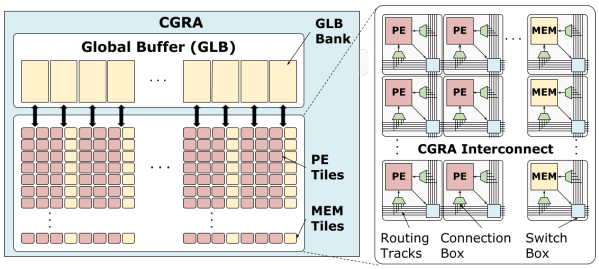


图 1： 与 [23] 对应的基线粗粒度可重构阵列框图。

App.	Task	Ver.	Tpt.	阵列切片	GLB 切片
ResNet-18	卷积层 2 x_	a	64	2	7
		b	256	6	7
	卷积层3 x_	a	64	2	4
		b	256	6	4
	卷积层4 x_	a	64	2	6
		b	256	6	6
	卷积层5 _x	a	64	2	20
		b	128	6	20
MobileNet	深度卷积_1 _逐点卷积_2_x	a	52	2	4
		b	208	5	4
	深度卷积_ _逐点卷积_3_x	a	52	2	4
		b	104	3	4
	深度卷积_ _逐点卷积_4_x	a	52	2	4
		b	104	3	4
摄像头流水线	摄像头流水线	a	3	4	4
		b	12	6	14
Harris	Harris	a	1	2	4
		b	2	4	7
		c	4	7	14

表 1： 不同资源使用量和吞吐量的任务变体。ResNet-18 和 MobileNet 由多个层组成，一个或多个层构成一个任务。ResNet-18 和 MobileNet 的吞吐量（Tpt.）单位为每周期乘加运算次数，摄像头流水线和 Harris 的吞吐量单位为每周期像素数。

进行通信。处理单元瓦片基于 Amber 版本扩展而来，以支持乘加运算。互连中的每个节点在每个方向上都有五条输入轨道和五条输出轨道，交换盒负责将数据从输入轨道路由至输出轨道。连接盒从输入轨道中选择数据，并将其路由至处理单元或存储器瓦片核心。全局缓冲区由 32 个存储体组成，每个存储体包含 128 KB 的静态随机存取存储器。每个 GLB 存储体通过位于阵列顶部的输入输出（IO）瓦片直接与瓦片阵列通信。

2.2. 调度器可见的硬件资源抽象

我们重点关注粗粒度可重构阵列（图1）中的三种关键硬件资源：全局缓冲区存储容量、全局缓冲区

[1]一个卷积_*dw*_逐点卷积 指深度卷积层和逐点卷积层合并而成的任务。

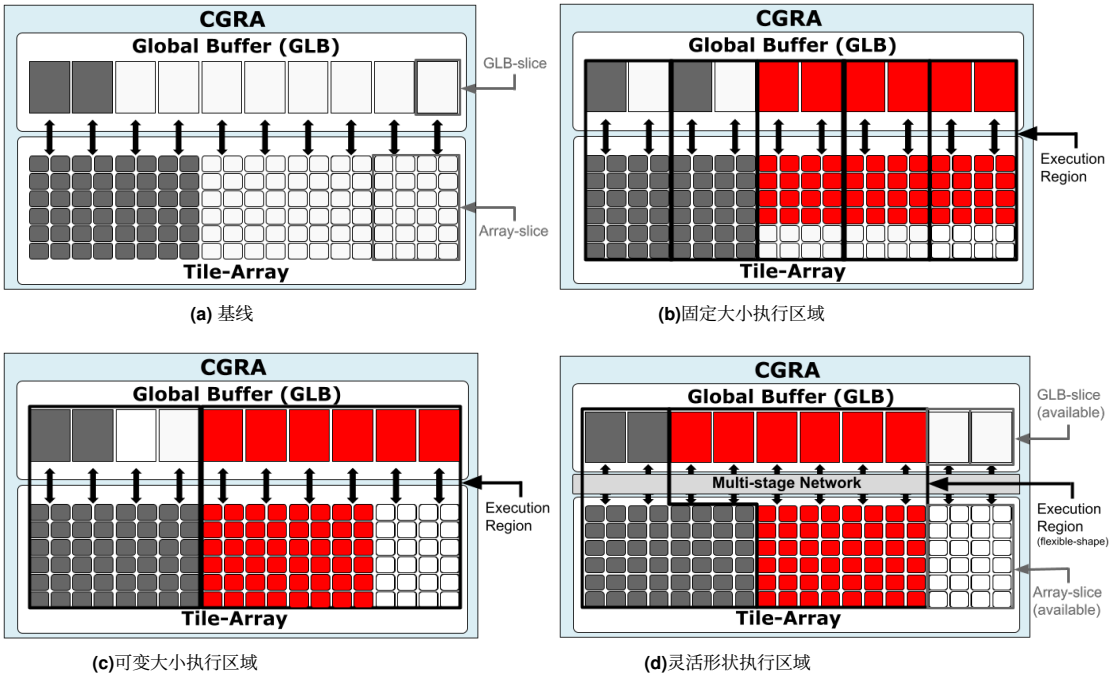


图 2： 基线 CGRA 和具有三种不同执行区域的 CGRA 中的资源分配。灰色方块表示当前正在运行的任务所占用的资源，红色方块表示下一个要运行的任务所占用的资源。

内存带宽，以及瓦片阵列中的计算资源。在 Amber 工具链中编译任务时 [23], , 编译器会将其转换为数据流图，其中每个节点和边分别表示硬件资源和通信。具体而言，全局缓冲区存储体用于中等规模的存储，以及与主机和瓦片阵列之间的通信；PE 和存储器（MEM）瓦片则用于计算，并作为小型暂存器。数据流图可用于推导内存容量、内存带宽、计算单元和吞吐量的使用情况。

我们通过分别将 GLB 和瓦片阵列划分为同构的全局缓冲区切片和阵列切片，对硬件资源进行抽象。例如，我们可以将 CGRA 中的每个 GLB 存储体抽象为一个全局缓冲区切片，并将瓦片阵列中每四列（48 个处理单元（PE）瓦片和 16 个存储器瓦片）抽象为一个阵列切片。这种抽象充当中间层，将编译器离线生成位流与调度器进行运行时资源分配解耦。在编译过程中，我们使用这些抽象后的全局缓冲区切片和阵列切片来表示每个任务的资源使用情况。例如，卷积2_*x* 层在 [19] 中使用 750KB 的全局缓冲区存储容量、17.3MB/s 的内存带宽、80 个处理单元（PE）瓦片和 17 个存储器瓦片，并在 500MHz 的时钟频率下实现每周期 64 次运算的吞吐量。按粗粒度资源切片用量计算，该任务抽象为七个全局缓冲区切片和两个阵列切片。通过调整编译器，可以生成同一任务的不同版本，使其具有不同的资源使用情况和吞吐量。例如，将同一任务的展开因子增加四倍，可实现 4 倍吞吐量（每周期 256 次运算），使用 288 个处理单元（PE）瓦片和 33 个存储器瓦片，同时保持相同的全局缓冲区存储存储容量和带宽，可抽象为七个全局缓冲区切片和六个阵列切片。我们的方法允许预先计算支持不同资源使用量和吞吐量的位流，并将其缓存在片上存储中，以支持快速动态部分重配置，稍后将对此进行讨论。表 1 总结了若干不同任务变体的资源使用量和吞吐量。在运行时，调度器利用硬件切片抽象以决定选择哪种任务变体、分配哪些资源，以及何时执行。

2.3. 硬件机制

灵活形状执行区域。为了管理多个并发运行的任务，我们需要一种方法来监控硬件资源和任务状态，这些任务基于上述抽象构建。我们引入一个执行区域，即 CGRA 上映射并执行单个任务的子区域。一个执行区域由一个或多个全局缓冲区切片和阵列切片组成。灵活构建不同大小和形状的执行区域，使调度器能够以简化且量化的方式查看硬件资源，同时提供足够的信息，为每个任务分配资源，从而最大限度地提高多任务工作负载中的资源利用率。

图 2 比较了构建执行区域的不同机制，以及它们对资源分配的影响。

灰色方块表示当前运行任务占用的资源，红色方块表示分配给下一个运行任务的资源。基线 CGRA（图 2a）不了解我们的硬件切片抽象，因此整个粗粒度可重构阵列充当一个大型执行区域。由于已有任务已映射到粗粒度可重构阵列上，后续任务只能等待前一个任务完成并释放该执行区域。

构成执行区域最简单的机制是仅支持固定大小的区域。例如，图2b 中的所有执行区域均由两个全局缓冲区切片和一个阵列切片组成。固定大小的区域并非最优。由于每个任务都必须适配固定大小的执行区域，因此资源使用量最高的最大任务决定了区域大小。另一方面，如果有多个可用的执行区域，可以展开任务并将其并行映射，以实现更高的吞吐量（例如，图2b中下一个运行的任务展开了三份）。这种方法无需对架构进行太大改动；假设所有目标任务都能适配一个执行区域，调度算法也很容易实现。然而，尽管展开能提高吞吐量，但要支持沿展开维度进行优化可能颇具挑战。

另一种方法是通过合并多个固定大小的区域来支持可变大小的执行区域。我们像固定大小区域那样定义区域的单位大小，但可以将多个单位区域合并，构成更大的执行区域。例如，图2c中，三个单位大小的区域合并起来，以执行下一个运行的任务（以红色标示）。可变大小执行区域的优势在于允许沿展开维度进行编译优化。例如，一个吞吐量为每周期 3 个像素的摄像头流水线任务使用四个阵列切片（表1）。直接将其展开四份，可使用 16 个阵列切片实现每周期 12 个像素的吞吐量。然而，编译器可以优化处理单元（PE）瓦片的分时复用，仅用六个阵列切片便实现每周期 12 个像素的吞吐量。支持可变大小的区域仍可预先计算具有不同资源使用量和吞吐量的任务变体的位流。不过，由于执行区域内全局缓冲区切片与阵列切片的比例始终不变，这种方法仍可能导致资源利用率偏低。

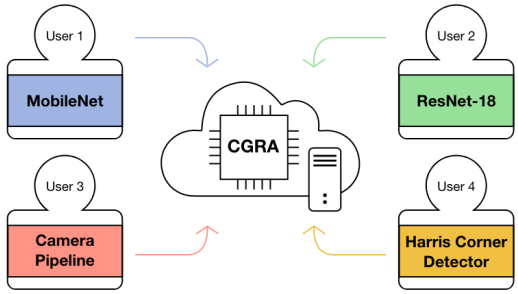
因此，我们提出灵活形状执行区域，其中全局缓冲区切片和阵列切片不再耦合。全局缓冲区切片与阵列切片的解耦能够实现更细粒度的资源分配。例如，图2d展示了如何为一个执行区域分配任意数量的全局缓冲区切片和阵列切片，从而形成非矩形形状，同时剩余的阵列切片和全局缓冲区切片可供其他任务使用。对灵活形状执行区域的支持提高了资源利用率，尤其适用于混合了内存密集型任务和计算密集型任务的多任务工作负载。不过，这可能需要在全局缓冲区切片和阵列切片之间进行额外通信。在本工作中，为简化研究，我们将执行区域内全局缓冲区切片和阵列切片的放置限制为连续布局。对灵活放置支持及其所需网络的设计空间探索将作为未来工作。第3.1节将通过一个云系统示例，更详细地介绍这些机制的优势。

动态部分重配置。动态部分重配置（DPR）是一种用于更新可重构架构中硬件配置的机制。我们提出了快速DPR，它沿用了 Amber SoC [7]，中提出的 DPR 机制，并增加了利用硬件抽象的功能。在 Amber 中，每隔一个 GLB 存储体就会存储配置比特流，并分别向瓦片阵列的两列传输配置。此外，时钟和配置信号会沿每列一同分配，使瓦片阵列能够在无需流水线级的情况下以较高时钟频率重新配置。在我们的粗粒度可重构阵列中，我们同样复用 GLB 模块来存储并向瓦片阵列传输位流，也采用相同的时钟分配网络。不过，与 Amber 不同的是，一个 GLB 存储体会向一个阵列切片（即瓦片阵列的四列）传输配置，因为阵列切片是执行区域的最小单位。

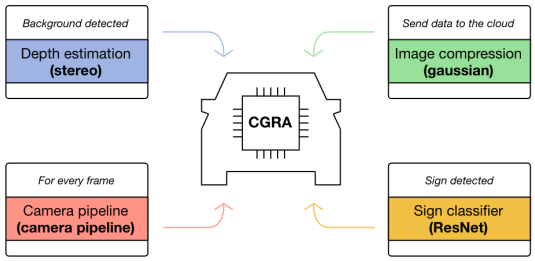
我们增加了在运行时重定位位流的功能，以进一步利用硬件抽象。在 Amber 中，编译器会生成区域感知型位流；即使两个区域具有相同的结构，某一区域的位流也无法在其他区域复用。这一限制源于不同列中每个配置寄存器的地址都带有不同的列号。另一方面，我们的编译器假设任务始终映射到最左侧的区域，因此会生成区域无关型位流。我们还在 GLB 存储体中添加了一个寄存器，用于指示 DPR 的目标区域。主机处理器触发 DPR 时，GLB 存储体会读取该寄存器，并通过 GLB 与瓦片阵列之间的网络，将位流传输到目标区域。借助这一位流重定位功能，用户可以预先将下一个任务的位流加载到 GLB 中，并只需写入一个寄存器，就能快速将其映射到下一个可用区域。

3. 评估

我们在两种不同的工作负载场景下评估多任务执行支持带来的优势。在云计算系统示例场景（第 3.1 节）中，与基线 CGRA 相比，我们采用灵活形状执行区域的 CGRA 吞吐量提高了 1.05 至 1.24 倍，归一化周转时间（NTAT）降低了 23% 至 28%。在自主系统示例场景（第 3.2 节）中，我们的 CGRA 将总延迟降低了 60.8%。

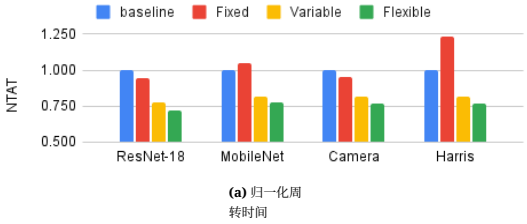


(a)云系统示例

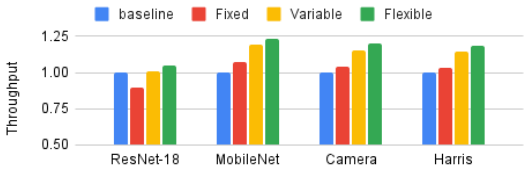


(b)自主系统示例

图 3： (a) 云系统示例场景：四个租户向粗粒度可重构阵列提交请求。每个租户分别分配到一个任务，任务来自 *MobileNet*、*ResNet-18*、摄像头流水线和 *Harris*。(b) 自主系统示例：任务可能在特定条件下触发。



(a) 归一化周转时间



(b) 吞吐量

图 4： 云系统示例中的评估。(a) 每个任务的归一化周转时间和 (b) 吞吐量；采用固定大小、可变大小和灵活形状资源划分，并以基线 CGRA 归一化。灵活形状分区使归一化周转时间降低 23–28%，吞吐量提高 1.05–1.24 倍。

3.1. 示例 1：云系统

概述。在此示例中，我们构建了一个合成云计算场景，用于模拟 CGRA 为多位用户处理应用请求的真实案例（图 3a）。我们使用来自机器学习（ML）和图像处理领域的内核构建多任务工作负载，包括机器学习领域的 ResNet-18 [19]和 MobileNet [20]，以及图像处理领域的摄像头流水线和 Harris角点检测器。表 1 总结了基准任务及其资源需求。

为生成多任务工作负载，我们假设四个租户共享粗粒度可重构阵列，并且每个租户被分配到四个目标应用之一。每个租户按照泊松分布向粗粒度可重构阵列发送请求。每当有新任务到达或现有任务完成时，调度器就会被触发，并运行贪心算法来调度下一个可用任务。调度器会在调度任务前检查其依赖关系是否满足（例如，在 ResNet-18 中，卷积*2_x* 依赖于 卷积*1_x*）。如果有多个版本的任务可以映射到可用资源上，贪心调度器总是选择吞吐量最高的版本。

指标。我们测量归一化周转时间和吞吐量，以比较基线 CGRA 与第2.3节所述的三种分区机制。周转时间（TAT）是从请求提交任务到任务完成之间的时间间隔。归一化周转时间（NTAT）是 TAT 与执行时间之比，表示任务的相对延迟（公式（1）-（2））。我们计算每个请求的 NTAT，并计算每个应用的算术平均值。我们还测量每个应用的平均吞吐量，以展示性能收益。

$$TAT = wait_time + execution_time \quad (1)$$
$$NTAT = TAT \,/\, execution_time \quad (2)$$

结果。图 4展示了与固定大小和可变大小执行区域相比，灵活形状执行区域在归一化周转时间和吞吐量方面的相对提升。即使采用简单的贪心调度算法，我们也实现了归一化周转时间降低 23–28%、吞吐量提高 1.05–1.24 倍。请注意，在本案例研究中，我们只为每个任务预编译了两个不同变体（表 1），调度器会在资源可用时贪心地选择吞吐量较高的变体。联合优化编译和调度策略可能会进一步改善归一化周转时间和吞吐量，这将留待未来工作研究。

3.2. 示例 2：自主系统

概述。在本案例研究中，我们构建了一个合成边缘系统场景，以模拟现实世界：图像处理和机器学习领域的多个任务并行运行，并可动态触发。具体而言，我们按照 [30] 中使用的方法，开发了一个自主系统场景，如图 3b 所示。[2] 该系统以 30 fps 的速度从传感器获取拜耳编码格式（RGGB）的 RAW图像，并首先运行一个摄像头

[2]我们还对任务进行了调整，以简化示例。

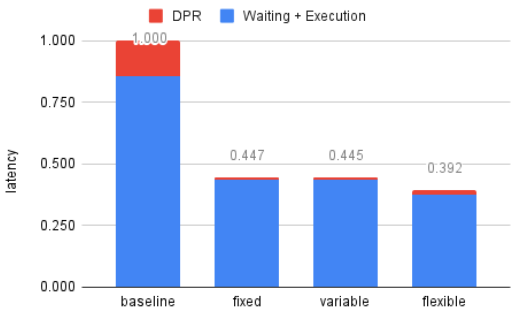


图5： 具有不同执行区域的自主系统示例的平均延迟。数值以基线结果归一化。红色柱表示重新配置所用的时间，蓝色柱表示等待时间与执行时间之和。为展示快速DPR（第 2.3 节）的优势，我们假设基线 CGRA 使用 AXI4-Lite 接口进行动态部分重配置，而其他方案使用快速DPR。

流水线任务，在粗粒度可重构阵列上将图像转换为 RGB图像。粗粒度可重构阵列生成 RGB图像后，系统会运行目标检测，并动态决定接下来执行的任务。[3] 当事件发生时（例如检测到特定背景），系统会处理该事件并执行相应的任务（例如深度估计）。除每帧都会运行的摄像头流水线外，我们将同一事件两次发生之间的间隔设为 3–7 帧之间的均匀随机分布。

结果。我们通过将所提出的粗粒度可重构阵列与采用基于AXI4-Lite的动态部分重配置的基线 CGRA 进行比较，评估硬件资源划分和快速DPR的优势。具体而言，基线 CGRA 每次仅映射一个任务。当发生多个事件时，基线会逐一处理每个任务，并通过顺序执行 AXI4-Lite 配置事务进行重新配置。在所提出的支持多任务执行的粗粒度可重构阵列中，我们利用灵活形状资源划分，在条件允许时让多个任务同时在粗粒度可重构阵列上运行。此外，我们采用第 2.3 节介绍的并行高频动态部分重配置机制来配置位流。我们计算所有帧的延迟算术平均值。如图5所示，与基线相比，我们的方法将延迟降低了60.8%。采用快速动态部分重配置后，重新配置所占时间不到总延迟的5%，相比之下，基线中这一比例为14.4%，降幅显著。

4. 相关工作

随着深度神经网络（DNN）在各个领域得到广泛应用，DNN加速器 [18, 17, 8, 9, 10, 25] 应运而生，并已部署于云系统 [21, 13]中。为此，许多先前的研究探索了云系统中DNN加速器对多租户的支持。

[3]本工作假设目标检测在系统中的其他硬件上执行（例如 GPU 或专用集成电路）。

面向云端和边缘计算的现场可编程门阵列中也研究了多任务执行支持。然而，通常有相当一部分FPGA资源被预留用于控制多任务执行，最终减少了可用的计算资源。ChordMap [27]探索了将多任务应用自动映射到粗粒度可重构阵列（CGRA）上，但其仅限于在流式应用中映射多个任务，且所有任务都需预先已知。我们的工作提出了硬件抽象和机制，编译器和调度器均可利用这些抽象和机制进行协同优化，从而提高云端和边缘系统中的资源利用率。

DNN加速器上的多任务执行。一些 DNN 加速器在软件层面为多个 DNN 任务提供服务。AI-MT [2] 和 Layerweaver [31] 提出了一种调度策略，将计算密集型任务与内存密集型任务混合执行，以提高硬件利用率。PREMA [11]通过实现可抢占式 NPU，利用时间复用支持多租户。许多研究通过提升加速器的灵活性来适配多个 DNN 任务。Planaria [14]提出了一种灵活的脉动阵列，并采用动态架构拆分来映射多个 DNN 任务。 [26] 提出了一种多向网络，以支持最多四个具有不同数据流的 DNN 任务。其他研究 [24, 3] 探索了配备多个具有不同硬件特性的 DNN 加速器的计算系统。虽然这些工作仅支持 DNN 工作负载，但我们的工作可以支持任何能够映射到粗粒度可重构阵列上的应用。

现场可编程门阵列上的多任务执行。在现场可编程门阵列中，虚拟化领域已对多任务执行支持进行了探索。一些研究将现场可编程门阵列划分为静态区域（即外壳），作为主机与现场可编程门阵列之间的粘合逻辑；以及动态区域（即角色），用于处理任务的计算。 [4, 5, 33] 将物理 FPGA 划分为多个固定大小的虚拟块，并由多个任务共享。AmorphOS [22]提出了一种现场可编程门阵列的硬件抽象——*Morphlet*，其大小会根据资源需求动态调整。ViTAL [35]提供了一个全栈框架，可在同构区域上运行多个不同大小的任务。 [34] 通过将硬件资源划分为多个处理单元核心并进行空间复用，支持在现场可编程门阵列上运行多项深度神经网络任务；而 [30] 则评估了利用动态部分重配置对现场可编程门阵列进行时间复用，以在嵌入式设备上运行视觉应用的优势。由于现场可编程门阵列的重配置时间较长，这些研究仅面向底层应用变化不频繁的场景；而我们的工作借助快速动态部分重配置，既支持云系统，也支持实时边缘系统。

5. 结论

在云端和边缘系统中，对加速器的多任务执行支持正变得越来越重要，并且

有望通过提高硬件利用率来提升性能。本研究提出将粗粒度可重构阵列中的硬件资源抽象为粒度更粗的单元，使工作负载调度器能够据此快速做出决策。基于这一抽象，我们开发了通过灵活形状的硬件分区和高吞吐量动态部分重配置来支持多任务执行的硬件机制。对云端和边缘系统场景进行建模的评估结果表明，这种抽象和硬件机制能够使自动调度器在未来的粗粒度可重构阵列上针对多任务工作负载实现高性能。

参考文献

[1] G. Ansaloni、P. Bonzini 和 L. Pozzi，《Egra：一种粗粒度可重构架构模板》，*IEEE 超大规模集成电路（VLSI）系统汇刊*，第19卷，第6期，1062–1074页，2010年。[2] E. Baek、D. Kwon 和 J. Kim，《一种多神经网络加速架构》，载于*2020 ACM/IEEE第47届计算机体系结构国际研讨会（ISCA）*。IEEE，2020年，940–953页。[3] A. Boroumand、S. Ghose、B. Akin、R. Narayanaswami、G. F. Oliveira、X. Ma、E. Shiu 和 O. Mutlu，《面向边缘设备的谷歌神经网络模型：分析并缓解机器学习推理瓶颈》，载于*2021*年第*30*届并行架构与编译技术国际会议（*PACT*）。IEEE，2021年，159–172页。[4] S. Byma、J. G. Steffan、H. Bannazadeh、A. Leon-Garcia 和 P. Chow，《云端FPGA：使用OpenStack启动虚拟化硬件加速器》，载于*2014 IEEE*第*22*届现场可编程定制计算机年度国际研讨会，2014年，109–116页。[5] ——，《云端FPGA：使用OpenStack启动虚拟化硬件加速器》，载于*2014 IEEE*第*22*届现场可编程定制计算机年度国际研讨会，2014年，109–116页。[6] F. Campi、A. Deledda、M. Pizzotti、L. Ciccarelli、P. Rolandi、C. Mucci、A. Lodi、A. Vitkovski 和 L. Vanzolini，《一种动态自适应数字信号处理器，适用于异构可重构平台》，载于*2007*年欧洲设计、自动化与测试会议暨展览会。IEEE，2007年，1–6页。[7] A. Carsello、K. Feng、T. Kong、K. Koul、Q. Liu、J. Melchert、G. Nyengele、M. Strange、K. Zhang、A. Nayak 等，《Amber：一款367 GOPS、538 GOPS/W的16纳米片上系统，配备粗粒度可重构阵列，可灵活加速稠密线性代数》，载于*2022 IEEE*超大规模集成电路技术与电路研讨会（*VLSI Technology and Circuits*）。IEEE，2022年，70–71页。[8] Y.-H. Chen、J. Emer 和 V. Sze，《Eyeriss：一种面向卷积神经网络能效数据流的空间架构》，*ACM SIGARCH 计算机体系结构新闻*，第44卷，第3期，367–379页，2016年。[9] Y.-H. Chen、T.-J. Yang、J. Emer 和 V. Sze，《Eyeriss v2：一种面向移动设备上新兴深度神经网络的灵活加速器》，*IEEE新兴与精选电路和系统主题期刊*，第9卷，第2期，292–308页，2019年。[10] Y. Chen、T. Luo、S. Liu、S. Zhang、L. He、J. Wang、L. Li、T. Chen、Z. Xu、N. Sun 等，《Dadiannao：一台机器学习超级计算机》，载于*2014*年第*47*届*IEEE/ACM*微体系结构国际研讨会。IEEE，2014年，609–622页。[11] Y. Choi 和 M. Rhu，《PREMA：一种面向可抢占神经处理单元的预测式多任务调度算法》，载于*2020 IEEE*高性能计算机体系结构国际研讨会（*HPCA*）。IEEE，2020年，220–233页。[12] J. Fowers、K. Ovtcharov、M. Papamichael、T. Massengill、M. Liu、D. Lo、S. Alkalay、M. Haselman、L. Adams、M. Ghandi 等，《一种用于实时人工智能的可配置云规模DNN处理器》，载于*2018 ACM/IEEE*第*45*届计算机体系结构国际研讨会（*ISCA*）。IEEE，2018年，1–14页。[13] ——，《一种用于实时人工智能的可配置云规模DNN处理器》，载于*2018 ACM/IEEE*第*45*届计算机体系结构国际研讨会（*ISCA*）。IEEE，2018年，1–14页。[14] S. Ghodrati、B. H. Ahn、J. Kyung Kim、S. Kinzer、B. R. Yatham、N. Alla、H. Sharma、M. Alian、E. Ebrahimi、N. S. Kim、C. Young 和 H. Esmaeilzadeh，《Planaria：通过动态架构拆分实现深度神经网络的空间多租户加速》，载于*2020*年第*53*届*IEEE/ACM*微体系结构国际研讨会（*MICRO*），2020年，681–697页。[15] G. Gobieski、A. O. Atli、K. Mai、B. Lucia 和 N. Beckmann，《Snafu：一种超低功耗、最低能耗的CGRA生成框架与架构》，载于*2021 ACM/IEEE*第*48*届计算机体系结构国际研讨会（*ISCA*）。IEEE，2021年，1027–1040页。[16] K. Guo、L. Sui、J. Qiu、J. Yu、J. Wang、S. Yao、S. Han、Y. Wang 和 H. Yang，《Angel-eye：将卷积神经网络映射到嵌入式FPGA的完整设计流程》，*IEEE*集成电路与系统计算机辅助设计汇刊，第37卷，第1期，35–47页，2017年。[17] S. Han、J. Kang、H. Mao、Y. Hu、X. Li、Y. Li、D. Xie、H. Luo、S. Yao、Y. Wang 等，《ESE：一种基于FPGA的稀疏长短期记忆网络高效语音识别引擎》，载于*2017*年*ACM/SIGDA*现场可编程门阵列国际研讨会论文集，2017年，75–84页。[18] S. Han、X. Liu、H. Mao、J. Pu、A. Pedram、M. A. Horowitz 和 W. J. Dally，《EIE：面向压缩深度神经网络的高效推理引擎》，*ACM SIGARCH 计算机体系结构新闻*，第44卷，第3期，243–254页，2016年。[19] K. He、X. Zhang、S. Ren 和 J. Sun，《用于图像识别的深度残差学习》，载于*IEEE*计算机视觉与模式识别会议论文集，2016年，770–778页。[20] A. G. Howard、M. Zhu、B. Chen、D. Kalenichenko、W. Wang、T. Weyand、M. Andreetto 和 H. Adam，《MobileNets：面向移动视觉应用的高效卷积神经网络》，*arXiv*预印本 *arXiv:1704.04861*，2017年。[21] N. P. Jouppi、C. Young、N. Patil、D. Patterson、G. Agrawal、R. Bajwa、S. Bates、S. Bhatia、N. Boden、A. Borchers 等，《张量处理单元的数据中心内性能分析》，载于第*44*届计算机体系结构国际研讨会论文集，2017年，1–12页。[22] A. Khawaja、J. Landgraf、R. Prakash、M. Wei、E. Schkufza 和 C. J. Rossbach，《AmorphOS：可重构逻辑阵列的共享、保护与兼容性》，载于第*13*届*USENIX*操作系统设计与实现研讨会（*OSDI 18*），2018年，107–127页。[23] K. Koul、J. Melchert、K. Sreedhar、L. Truong、G. Nyengele、K. Zhang、Q. Liu、J. Setter、P.-H. Chen、Y. Mei、M. Strange、R. Daly、C. Donovick、A. Carsello、T. Kong、K. Feng、D. Huff、A. Nayak、R. Setaluri、J. Thomas、N. Bhagdikar、D. Durst、Z. Myers、N. Tsiskaridze、S. Richardson、R. Bahr、K. Fatahalian、P. Hanrahan、C. Barrett、M. Horowitz、C. Torng、F. Kjolstad 和 P. Raina，《Aha：一种用于粗粒度可重构加速器及编译器设计的敏捷方法》，*ACM*嵌入式计算系统汇刊，2022年4月，已接收。 [在线]。可访问：https://doi.org/ 10 1145 3534933 。/ [24] H. Kwon、L. Lai、M. Pellauer、T. Krishna、Y.-H. Chen 和 V. Chandra，《面向多DNN工作负载的异构数据流加速器》，载于*2021 IEEE*高性能计算机体系结构国际研讨会（*HPCA*）。IEEE，2021年，71–83页。[25] H. Kwon、A. Samajdar 和 T. Krishna，《MAERI：通过可重构互连实现DNN加速器上的灵活数据流映射》，*ACM SIGPLAN*通报，第53卷，475，2018年2月，461号，第——页。[26] J. Lee、J. Choi、J. Kim、J. Lee 和 Y. Kim，《数据流镜像：为脉动阵列NPU上的高效细粒度空间多任务处理提供架构支持》，载于*2021*年第*58*届*ACM/IEEE*设计自动化会议（*DAC*）。IEEE，2021年，247–252页。[27] Z. Li、D. Wijerathne、X. Chen、A. Pathania 和 T. Mitra，《ChordMap：将流式应用自动映射到粗粒度可重构阵列》，*IEEE*集成电路与系统计算机辅助设计汇刊，第41卷，第2期，306–319页，2022年。[28] L. Liu、D. Wang、M. Zhu、Y. Wang、S. Yin、P. Cao、J. Yang 和 S. Wei，《一种用于多标准视频解码的高能效粗粒度可重构处理单元》，*IEEE*多媒体汇刊，第17卷，1720，2015年10月，1706号，第——页。[29] J. Mbongue、F. Hategekimana、D. T. Kwadjo、D. Andrews 和 C. Bobda，《FPGAVirt：一种面向云端FPGA的新型虚拟化框架》，载于*2018 IEEE*第*11*届云计算国际会议（*CLOUD*）。IEEE , 2018, 第862–865页。

[30] M. Nguyen、R. Tamburo、S. Narasimhan 和 J. C. Hoe，“量化动态部分重配置对嵌入式视觉应用的益处”，载于*2019*年第*29*届国际现场可编程逻辑与应用会议（*FPL*）。IEEE，2019年，第129–135页。[31] Y. H. Oh、S. Kim、Y. Jin、S.Son、J. Bae、J. Lee、Y. Park、D. U.Kim、T. J. Ham 和 J. W. Lee，“Layerweaver：通过逐层调度最大化神经处理单元的资源利用率”，载于*2021*年*IEEE*高性能计算机体系结构国际研讨会（*HPCA*）。IEEE，2021年，第584–597页。[32] A. Vasilyev、N. Bhagdikar、A. Pedram、S. Richardson、S. Kvatin-sky 和 M. Horowitz，“评估成像与视觉应用的可编程架构”，载于*2016*年第*49*届*IEEE/ACM*国际微体系结构研讨会（*MI-CRO*）。IEEE，2016年，第1–13页。[33] J. Weerasinghe、F. Abel、C. Hagleitner 和 A. Herkersdorf，“使现场可编程门阵列能够应用于超大规模数据中心”，载于*2015*年*IEEE*第*12*届普适智能与计算国际会议、*2015*年*IEEE*第*12*届自主与可信计算国际会议、*2015*年*IEEE*第*15*届可扩展计算与通信国际会议及其相关研讨会（*UIC-ATC-ScalCom*），2015年，第1078–1086页。[34] S. Zeng、G. Dai、H. Sun、K. Zhong、G. Ge、K. Guo、Y. Wang 和 H. Yang，“在云端实现高效、灵活的现场可编程门阵列虚拟化，以支持深度学习”，载于*2020*年*IEEE*第*28*届现场可编程定制计算机年度国际研讨会（*FCCM*）。IEEE，2020年，第102–110页。[35] Y. Zha 和 J. Li，“在云端虚拟化现场可编程门阵列”，载于第二十五届编程语言与操作系统架构支持国际会议论文集，2020年，第845–858页。