计算机研究与发展
Journal of Computer Research and Development
https://crad.ict.ac.cn
63(11): 2788−2806, 2026

本期推荐

TH-FCAD: 面向 FPGA 集群的大规模神经网络自动化部署框架

赵言亢　黎　渊　齐星云　陆亿行　刘谱光　王　强　谢　旻　彭　林　赖明澈
（国防科技大学计算机学院　长沙　410073）
（zhaoyankangnudt@nudt.edu.cn）

TH-FCAD: Automated Deployment Framework for Large-Scale Neural Networks on FPGA Clusters

Zhao Yankang, Li Yuan, Qi Xingyun, Lu Yihang, Liu Puguang, Wang Qiang, Xie Min, Peng Lin, and Lai Mingche
(*College of Computer Science and Technology*, *National University of Defense Technology*, *Changsha* 410073)

Abstract　With the rapid growth of neural network model sizes, single FPGA resources have become insufficient for accelerating large-scale models, while FPGA clusters face critical challenges, including complex architecture design, inefficient task partitioning, and the lack of automated toolchains. To address these issues, an end-to-end automated deployment framework for neural networks on FPGA clusters, named TH-FCAD (automated deployment framework for large-scale neural networks on FPGA clusters) is proposed. First, a two-level interconnect FPGA cluster architecture based on TH-Express is constructed. By integrating a high-speed global GTH (gigabit transceiver high-speed) network with a local LVDS (low voltage differential signaling) full-mesh interconnect, it achieves high-bandwidth and low-latency cross-node communication. Second, a latency-balancing optimization and model partitioning method is introduced to evenly partition large-scale neural networks into multiple acceleration kernels while maintaining inter-kernel delay equilibrium, which is essential for pipeline efficiency. Finally, a cluster-aware mapping algorithm is designed to automatically generate optimal placement schemes by jointly considering FPGA resource constraints and the cluster communication topology, thereby minimizing communication overhead. Furthermore, a complete automated toolchain is developed to streamline the entire deployment process, enabling seamless compilation from PyTorch/TensorFlow trained models to FPGA cluster configurations. Evaluations on a built 16 node FPGA cluster demonstrate that when deploying ResNet-8, SqueezeNet, and ResNet-50, TH-FCAD achieves inference latencies of 0.051 ms, 1.24 ms, and 1.36 ms, respectively, alongside a peak computational performance of 5.95 TOPS. Compared with state-of-the-art multi-FPGA approaches, the proposed framework delivers a 28.4% performance improvement and outperforms corresponding CPU and GPU implementations.

Key words　neural network; hardware acceleration; FPGA cluster; task partition; automated deployment framework

摘　要　针对单颗 FPGA 资源受限难以加速大规模神经网络，且 FPGA 集群面临架构设计复杂、任务划分低效及自动化工具链缺失等挑战，提出面向 FPGA 集群的端到端自动化部署框架 TH-FCAD（automated deployment framework for large-scale neural networks on FPGA clusters）。首先，构建基于 TH-Express 的 2 级互连集群架构，融合全局互连与局部互连，实现高带宽、低延迟的跨节点通信；其次，提出延迟平衡优化与模型切分方法，将大规模模型均匀切分为多加速核并保持核间延时均衡；最后，设计集群感知映射算法，联

收稿日期：2025-12-05；**修回日期**：2026-07-09
基金项目：国家重点研发计划项目（2023YFB4403400）
This work was supported by the National Key Research and Development Program of China (2023YFB4403400).
通信作者：赖明澈（mingchelai@nudt.edu.cn）

合 FPGA 资源约束与通信拓扑自动生成最优映射，以最小化通信开销。构建了完整的工具链，实现了从 PyTorch/TensorFlow 模型到集群的端到端自动化编译部署。基于 16 节点 FPGA 集群的评估表明，部署 ResNet-8, SqueezeNet, ResNet-50 时，推理延迟分别为 0.051 ms, 1.24 ms, 1.36 ms，峰值算力达 5.95 TOPS；较现有多 FPGA 方案性能提升 28.4%，且优于 CPU 与 GPU 平台。

关键词 神经网络；硬件加速；FPGA 集群；任务划分；自动化部署框架

中图法分类号 TP302.1；TP301

DOI: 10.7544/issn1000-1239.202550866 **CSTR:** 32373.14.issn1000-1239.202550866

神经网络的广泛应用[1-2]与模型规模的指数级增长[3]使传统计算架构面临严峻的功耗与存储墙限制[4]。相比高功耗的 GPGPU[5]与缺乏灵活性的 ASIC，FPGA 凭借可重构性、低功耗及细粒度流水线并行能力，成为神经网络高能效加速的理想选择[6-7]。然而，单颗 FPGA 有限的片上存储和计算资源难以适应神经网络模型规模的快速增长[8]。

为应对这一挑战，传统方案主要采取 2 种策略：一方面，利用稀疏化、量化等模型压缩技术减少参数量和计算量；另一方面，依赖片外存储器存储庞大的权重参数设计统一的硬件架构，并通过复用计算单元来串行执行网络层。这类架构通过编译指令控制计算和数据调度，虽然实现了一些大规模模型在单 FPGA 芯片上的部署[9-11]，但存在明显缺陷：模型压缩导致精度损失，层间串行执行和频繁的片外访存引入了显著的延迟开销[3]。

数据流架构为应对上述挑战提供了新的途径。该架构将神经网络层映射为独立的硬件加速核，实现层间流水并行与加速核间的片上直接数据传输，从而有效消除了中间结果的片外访存需求，显著降低了系统推理延时。同时，其固有的高度可扩展性，为突破单 FPGA 资源瓶颈提供了可能。该架构天然的分布式特征使其能够将计算任务有效地划分到多个 FPGA 节点上，通过跨节点的协同计算来突破单个芯片的资源限制。已有实证研究表明，这种架构能有效克服传统“内存墙”问题，在神经网络加速中实现更低的推理延迟[12-14]。正是基于数据流架构展现的高度可扩展性与性能潜力，研究者们开始尝试挖掘 FPGA 集群加速神经网络的潜力[15-17]。然而，现有方案大多仅针对特定应用实现定制化设计，在架构设计、编程抽象和通信优化等关键维度上缺乏系统性解决方案。

具体而言，在 FPGA 集群平台上实现大规模神经网络模型的高效加速，仍面临以下三大挑战：1）可扩展 FPGA 集群架构设计。如何设计高效可扩展的 FPGA 集群架构，确保多节点间实现高带宽、低延迟的数据传输，以支撑大规模并行协同计算。2）任务划分方法。如何设计大规模神经网络模型推理任务到多 FPGA 的划分策略，并实现与之匹配的并行策略和协同机制。3）自动化工具链缺失。如何建立端到端的编译优化框架，将模型描述自动映射为多 FPGA 配置方案，同时优化硬件资源利用率与通信开销。

此外，由于 FPGA 的开发流程较为复杂，研究人员不仅需掌握底层硬件开发技能，还需深入理解神经网络的执行机制[18-19]。在 FPGA 集群环境中，更需针对每个节点进行定制化设计，并处理跨设备的数据通信与任务同步，进一步加大了开发难度。这些因素共同导致当前面向 FPGA 集群的神经网络加速研究工作仍然相对匮乏。

针对上述挑战，本文提出一种面向 FPGA 集群的神经网络端到端自动化部署框架——TH-FCAD（automated deployment framework for large-scale neural networks on FPGA clusters），主要贡献包括：

1）提出 2 级互连的 FPGA 集群架构。通过全局 GTH（gigabit transceiver high-speed）高速网络与局部 LVDS（low voltage differential signaling）全互连的分层通信机制，实现了多 FPGA 节点间高带宽、低延迟的数据交换，局部互连特性可将通信延时降低 18.7%。

2）提出延迟平衡优化与模型切分方法。基于性能-资源模型的延迟平衡方法能够自动优化各层设计参数，实现流水线延迟均衡；同时提出支持多加速核协同推理的模型切分策略，使神经网络模型能够在多 FPGA 上分布式部署加速。

3）提出集群感知的加速核映射算法与自动化集群构建工具。针对多加速核的部署需求，提出集群感知的映射算法，在满足资源约束条件下最小化通信开销，并开发自动化集群构建流程，实现了加速核到多 FPGA 节点的高效映射与工程自动生成，大幅提升了部署效率与系统性能。

4）构建了包含 16 节点的 FPGA 集群进行实验验

证。结果表明, TH-FCAD 框架在 ResNet-8, SqueezeNet, ResNet-50 上分别实现 0.051 ms, 1.24 ms, 1.36 ms 推理延迟, 以及 5.95 TOPS 的峰值计算性能, 相比现有多 FPGA 方法提升了 28.4%。

需要说明的是, 本文中“大规模神经网络”的界定, 主要立足于 FPGA 集群自动化部署加速的特定视角。在单 FPGA 数据流架构下, 片上资源(尤其是 DSP 和 BRAM)严格制约着可直接部署的模型规模; 现有代表性的自动化部署工具(如 HLS4ML[20], FINN[21] 等)通常仅能高效支持较小或经高度压缩的网络。因此, 从硬件资源承载能力的角度来看, ResNet-50 这类包含 50 余层及残差连接的模型已对单芯片构成显著挑战, 足以成为验证多 FPGA 协同计算与自动化部署框架有效性的代表性案例。而参数量更大的 Transformer 系列模型(如 BERT, Llama 等), 则超出了当前框架验证范围, 将其高效部署于 FPGA 集群将是我们未来的重点研究方向。

1 相关工作

1.1 FPGA 加速器架构

神经网络的推理过程本质上是通过层次化结构的乘累加(multiply-accumulate, MAC)运算和非线性变换组合实现的。在满足数据依赖的前提下, 推理计算可以通过多个计算单元沿不同维度进行并行化处理, 包括层间并行、输入批次并行以及通道并行等。从架构上看, 当前主流的 FPGA 加速架构主要分为处理单元阵列(processing element array, PEA)和数据流(dataflow)两类, 具体如图 1 所示。

处理单元阵列架构采用固定并行度的 MAC 单元阵列, 通过灵活编译执行多种模型计算任务[22-24]。该架构采用统一的硬件架构适配所有网络层, 不同层需按照数据依赖关系串行执行, 通过编译指令控制计算流程和调度数据搬运, 并依赖片外存储器存储权重和中间结果。这种架构虽然具有设计复用性高、资源消耗少的优点, 但其存在 3 个根本性缺陷: 1)固定的计算结构难以适配不同层的计算特征, 导致资源利用率低下; 2)层间数据依赖导致流水线执行效率受限, 且中间数据的频繁搬移会产生额外延迟; 3)加速器需要存储整个模型的权重和大量中间结果, 而 FPGA 片上存储器容量有限, 使得片外存储器带宽成为性能瓶颈。

数据流架构则采用多个加速核协作完成模型推理任务, 各层被映射到不同的计算单元并行执行, 并采用片上缓存流式传输中间结果[20-21,25-27]。该架构从 3 个层面有效突破了传统冯·诺伊曼架构所面临的“内存墙”瓶颈问题[28]: 1)通过片上数据流传输消除中间结果的存储需求; 2)参数完全存储在片上, 消除了片外存储访问开销; 3)通过层间计算重叠技术打破顺序执行限制, 从而显著降低总体延时。

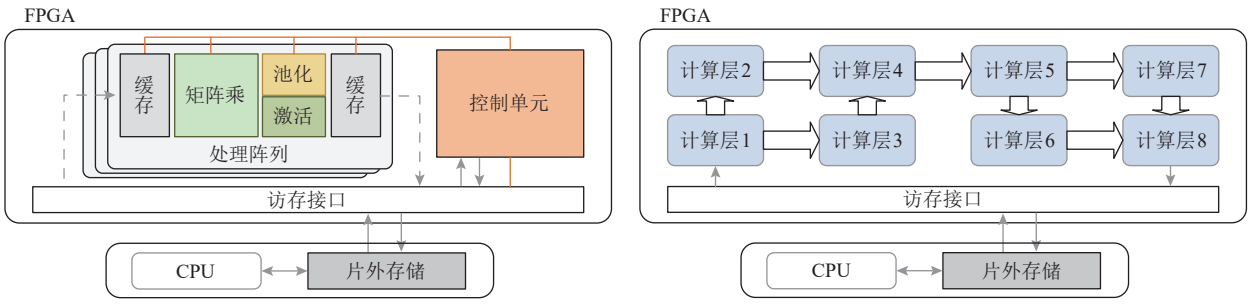


Fig. 1 Comparison of two accelerator architectures

图 1 2 种加速器架构对比

数据流架构通过将网络各层展开为独立的硬件加速核以实现并行计算, 其固有的分布式特性赋予了其优异的扩展性。然而, 这种资源完全展开的设计特性也导致其资源消耗与模型复杂度(如层数和参数量)成正比。现有研究如 HLS4ML[20], FINN[21] 等实现的数据流架构加速器, 在单块 FPGA 芯片的物理资源约束下, 其所能实现的神经网络规模仍然受限。正是为了充分利用数据流架构的扩展性优势以突破单 FPGA 的资源瓶颈, 研究者转向探索多 FPGA 协同计算方案。然而, 此类方案在实现过程中也面临跨设备协同计算、通信开销优化等一系列新的技术挑战。

1.2 神经网络自动化加速框架

随着高层次综合(high-level synthesis, HLS)[29] 技术的不断发展, FPGA 开发门槛显著降低, 并呈现出众多自动化工具。然而, 现有工具在功能和适用性方面仍存在诸多局限性。例如, fpgaConvNet[25] 基于

同步数据流模型实现了设计空间探索，但在跨层优化方面能力不足；Vitis AI[30] 虽提供端到端部署方案，但其闭源特性限制了定制化扩展的灵活性；FINN[21,31] 专注于低位宽网络，适用范围相对狭窄；HLS4ML[20] 虽支持主流框架模型转换，但其架构优化局限于单 FPGA 资源约束。综上所述，现有 FPGA 自动化框架的共同缺陷在于：其设计普遍局限于单设备部署场景，缺乏设备间通信接口的协同设计能力，同时未能建立多 FPGA 协同优化机制，因此难以满足大规模模型的分布式部署需求。

为此，已有研究开始探索在 FPGA 集群上实现神经网络加速。FPGA 集群架构根据互连结构可分为专用型与通用型 2 类。专用型架构通过定制互连实现特定模型的加速，例如文献 [15] 通过 SATA 电缆构建了由 6 个 FPGA 组成的集群，其实现的 AlexNet 和 VGG16 模型相比 CPU 和 GPU 分别获得了 21 倍和 2 倍的能效比提升。文献 [16] 通过 PCIe 将 16 块 FPGA 连接到主机，实现的 ResNet-152 模型在计算性能上相比已有工作获得 16.4 倍的提升。文献 [17] 在 SFP+ 连接的 2 块 ZCU102 FPGA 上部署 YOLO 模型，相对于单 FPGA 设计实现了 3.48 的加速比。这些架构虽然对特定模型进行了优化，但缺乏通用性和可扩展性。

通用型集群架构致力于提升系统的灵活性和可扩展性。其代表性工作包括微软 Brainwave[32]，其采用数据中心网络构建了包含 280 个 FPGA 的集群，用于加速搜索算法和深度学习模型，但闭源架构限制了其应用范围。Galapagos[33] 通过网络堆栈抽象实现了 FPGA 虚拟化，隐藏了底层通信细节，为软件编程及任务调度提供了灵活性。AIgean 框架[34] 进一步基于以太网构建了通用集群，实现了 AutoEncoder 与 ResNet-50 模型的加速，但其软件协议栈引入的通信延迟（>5 ms）严重制约了加速核流水线执行效率，且缺乏模型的自动化切分方法。此外，Elastic-DF[35] 针对多 Die FPGA（multi-die FPGA）设计资源平衡策略，在 ResNet-50 和 MobileNetV1 模型上分别获得了 44% 的延迟降低和 78% 的吞吐量提升，但其仅在少量 FPGA 上实现低位宽（1b）模型，且未考虑面向集群架构的优化。

综上所述，现有研究工作在 3 个方面存在不足：1）在集群架构方面，专用型架构的互连方式缺乏灵活性，而通用型互连方式通信效率较低；2）在加速框架方面，现有的面向单 FPGA 的加速框架无法直接应用于集群环境；3）缺乏自动化的任务划分和映射方法。针对上述问题，本文提出系统化的解决方案：基于高速互连网络的透明化集群架构、融合资源与通信开销优化的任务划分映射算法以及支持集群部署的全流程自动化工具链。

1.3 TH-Express 高性能互连技术

TH-Express[36-40] 高性能互连网络由网络路由芯片（network routing chip, NRC）和网络接口芯片（network interface chip, NIC）两种专用芯片构成。NRC 提供多端口高吞吐交换；NIC 支持短报文（mini-packet, MP）与远程直接内存访问（remote direct memory access, RDMA），并基于硬件逻辑实现可靠传输与流量控制。相较于传统 PCIe 与以太网，TH-Express 在拓扑灵活性与协议效率方面具有显著优势，适于构建大规模 FPGA 集群。现有研究尚未充分挖掘其在深度学习 FPGA 集群加速中的潜力，本文基于 TH-Express 构建 FPGA 集群系统，实现节点间高速通信，为大规模神经网络的集群化部署提供可靠网络基础。

2 FPGA 集群架构

现有 FPGA 集群架构主要可分为 3 类，其组织方式对计算效率具有决定性影响，如图 2 所示。协处理器架构[16] 将 FPGA 作为 CPU 附属设备，所有跨节点通信均需通过 CPU 进行中转。尽管这种架构能够兼容传统数据中心网络，但 FPGA 间的高通信延迟严重制约了协同计算效率。Catapult[41] 提出的 Bump-in-the-wire 架构，将 FPGA 嵌入 NIC。该架构虽可利用数据中心现有网络拓扑，但 FPGA 需同时处理计算与网络数据流，导致约 15% 逻辑资源的额外开销[32,41]。专用互连架构[15] 为特定模型定制 FPGA 互连网络结

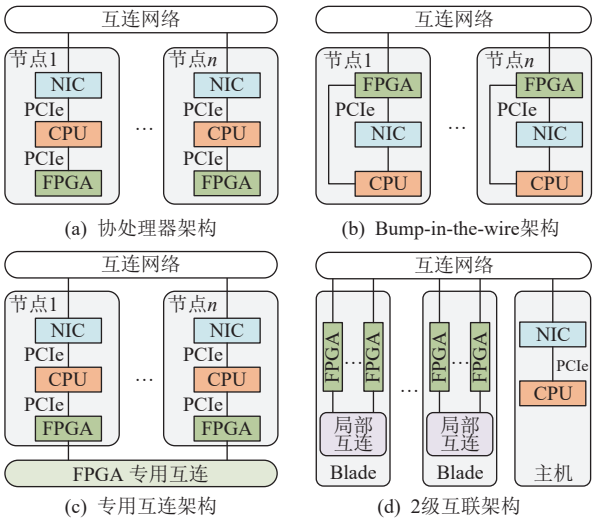


Fig. 2 Comparison of different cluster architectures

图 2 不同的集群架构对比

构, 能够获得极低的通信开销。然而其刚性拓扑结构导致扩展性受限, 且需为不同模型重新设计物理连接。此外, 由于 FPGA 互连网络与 CPU 等设备存在拓扑异构性, 实际部署时还需考虑跨架构网络结构通信所带来的性能损耗。

本文提出的基于 TH-Express 的 2 级互连集群架构(如图 2(d)所示)属于融合通用型与专用型拓扑优势的混合架构。该架构通过分层网络设计实现高效互连: 全局层采用光纤连接至高速路由芯片, 构建通用互连网络, 支持任意规模节点的灵活扩展; 局部层则在相邻 FPGA 节点间采用低延迟、高带宽的 LVDS 实现点对点专用通信。该混合架构通过分层设计融合了通用互连与专用互连的优势, 有效兼顾了系统通用性和通信效率的双重需求。虽然双模互连设计增加了硬件复杂度(如 FPGA 逻辑资源消耗与接口设计难度), 但通过本文提出的自动化工具链(详见第 4 节), 我们实现了端到端的全自动化部署流程, 有效解决了这些工程挑战。

此外, FPGA 与 CPU 通过 GTH 网络直接相连, 构成对等计算节点, 避免了传统架构中主从通信模式带来的性能瓶颈。在软件层面, 主机可通过用户态接口直接访问任意 FPGA 节点, 消除了传统层次化访问模式下的性能开销。同时, Shell/Role 架构将物理分散的 FPGA 资源虚拟化为单一逻辑设备, 并彻底隐藏了设备间的底层通信细节。在此架构下, 计算任务被拆分为多个阶段, 每个阶段由一个独立的加速核执行。系统采用统一编址方案为各加速核分配唯一地址, 并通过自动部署将其映射至指定 FPGA 节点, 同时动态配置路由表。推理时, 主机将计算请求封装为 TH-Express MP 报文, 经 NRC 路由至首节点。数据流驱动加速核计算, 流交换模块基于路由表自主将计算结果转发至下一目标加速核。后续加速核以相同机制自动传递数据, 最终结果经 NRC 网络返回主机。该架构通过数据流驱动与自主路由的有机结合, 实现了全程无需主机干预的自动化推理计算。

2.1 面向报文交换的 2 级互连架构

本文所提出的层次化 2 级互连架构如图 3 所示。该架构基于 GTH 全局网络与 LVDS 直接连接的协同设计, 构建了全局与局部 2 级通信模式。其中: 第 1 级采用基于 TH-Express 的 GTH 网络, 负责实现系统全局范围内的高效通信; 第 2 级则通过 LVDS 直接连接, 构建局部高带宽通信通道。2 级网络协同工作, 形成了层次化的通信拓扑结构, 从而在全局通信能力和局部数据传输效率之间实现优化平衡。

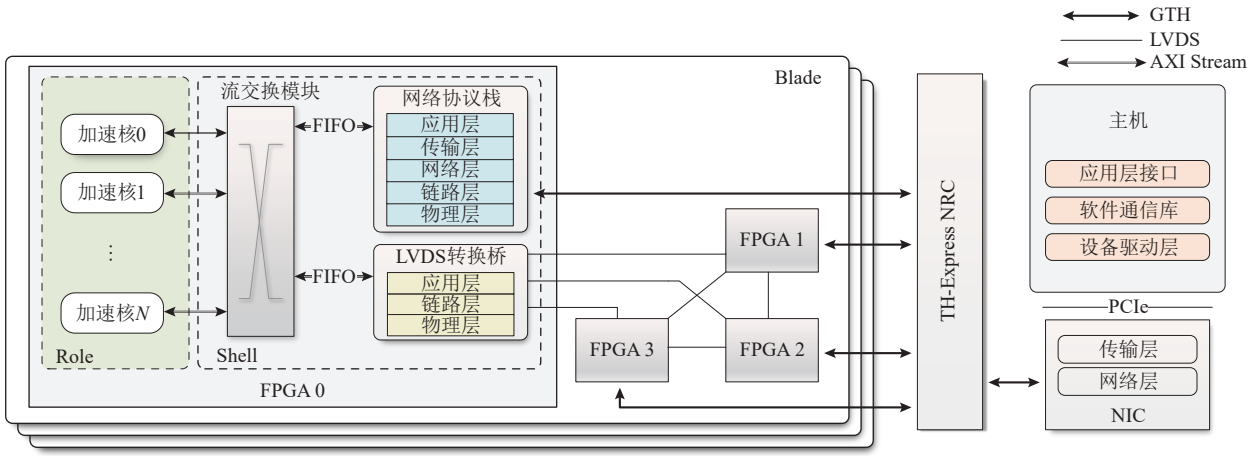


Fig. 3 TH-Express-based FPGA cluster architecture with two-level interconnect

图 3 基于 TH-Express 的 2 级互连 FPGA 集群架构

2.1.1 全局 GTH 互连

全局互连网络基于 NRC 与 GTH 高速串行链路构建, 采用胖树拓扑结构以支持系统的大规模扩展[39]。每个 NRC 芯片集成多个全双工 SerDes 端口, 支持灵活的通道速率配置, 具备高带宽传输能力。在路由机制方面, 采用基于分布式查找表的自适应路由策略, 各端口内置预定义路由表, 报文仅需携带目标物理地址即可实现自主寻址与高效转发。

在 FPGA 实现方面, 本文设计了专用协议栈, 以支持 FPGA 节点接入 TH-Express 互连网络, 并实现 AXI Stream 数据与网络报文的高效转换。该协议栈采用分层架构, 包括物理层、链路层、网络层、传输层与应用层。物理层集成多通道 SerDes 硬核, 支持可配置传输速率; 链路层通过数据校验确保可靠传

输；网络层实现地址解析和动态路由；传输层主要采用 MP 协议，适用于低延迟通信；应用层则提供 AXI Stream 接口，实现网络数据流与计算核之间的无缝对接。

2.1.2 局部 LVDS 互连

在 Blade 内部（4 个 FPGA 构成 1 个刀片），采用 LVDS 全互连架构实现板内 FPGA 间的低延迟、高带宽点对点通信。为提升传输效率，设计了 3 层精简协议栈，包括物理层、链路层与应用层。物理层集成 LVDS 收发器与时钟同步电路，提供高带宽传输能力；链路层实现多通道数据同步与基于信用的流控机制，确保数据可靠传输并避免接收端溢出；应用层则实现 AXI Stream 接口的无缝对接，支持反压传递以优化流量控制。该协议在满足低延迟要求的同时，保障了数据流的高效处理。

2.2 Shell/Role 架构

每个 FPGA 节点的设计分为静态的 Shell 层和动态的 Role 层。其中，Shell 层包含流交换模块、GTH 网络栈模块和 LVDS 转换桥模块，分别提供 GTH 和 LVDS 通信接口。Role 层则动态集成分配到该节点的所有加速核，形成可重构的任务执行单元。

流交换模块采用参数化设计，数据交换端口数量与位宽可根据系统需求灵活配置，其微架构如图 4 所示。各端口遵循 AXI Stream 协议，利用数据包中的 TDEST 字段携带目的地址信息。端口间通过交叉开关实现互连，由控制状态机统一管理连接状态，支持多端口并行传输。

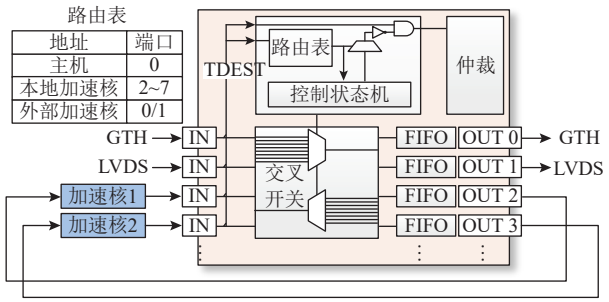


Fig. 4 Stream switch module microarchitecture and integration example with acceleration cores

图 4 流交换模块微架构及其与加速核连接示例

当数据包到达输入端口时，控制状态机依据 TDEST 字段查询转发路由表，确定目标输出端口。若目标端口处于空闲状态，交叉开关立即建立连接并开始数据传输；若端口忙碌，则通过 AXI Stream 协议的反压机制使输入端口等待。每个输出端口均设有 FIFO 缓冲区，可在连接建立阶段缓存首数据，并借助 AXI Stream 协议的反压机制避免溢出。当多个输入端口同时请求时，状态机通过仲裁机制选择其中之一优先传输。

本文设计的 Shell/Role 架构实现了 FPGA 通信硬件与加速器逻辑设计的解耦。集群中所有 FPGA 的 Shell 层保持统一设计，通过标准化流式接口为 Role 层提供透明化的集群通信能力。加速器的设计无需考虑 FPGA 底层通信细节，只需完成加速核端口与流交换模块的拼接，并配置对应的路由表项，即可实现加速核在集群上的部署。该过程通过自动化映射部署工具 AutoDeploy 实现。针对每个 FPGA 节点，该工具能够自动完成模块拼接、转发路由表配置及 FPGA 工程生成，随后执行综合、布局布线等操作，最终生成 FPGA 集群每个节点的位流，并实现位流的自动下载与部署。

2.3 软件通信栈

为了构建高效的 CPU-FPGA 通信机制，本文设计并实现了 3 层式软件通信栈。设备驱动层实现 NIC 硬件资源管理与内核态通信服务。软件通信库层提供面向应用的 NIC 直接访问接口，并采用连接协议保证数据传输的可靠性。应用层对数据传输的实现细节进行了封装，支持应用程序通过用户接口与任意 FPGA 节点建立直连通道。用户在准备好待计算的数据后，可直接调用接口函数将其发送到 FPGA 首节点，并等待计算完成后的结果返回。整个过程无需额外操作，显著降低了应用程序的开发复杂度。

3 神经网络自动化部署框架

基于 2 级互连的 FPGA 集群架构，本文提出了一种面向 FPGA 集群的端到端全流程自动化映射与部署框架 TH-FCAD，其架构如图 5 所示。该框架可将 PyTorch 或 TensorFlow 训练生成的神经网络模型自动映射部署至 FPGA 集群，其核心流程包含以下 4 个关键阶段：

1）模型解析与优化阶段。TH-FCAD 接收预训练神经网络模型描述文件（如 h5 格式），通过模型解析模块构建模型层次结构。以 ResNet-50 模型为例，能够精确识别模型包含的 53 个卷积层、50 个 ReLU 层以及跨层连接组件。基于扩展的 HLS4ML 框架，工具自动优化设计参数，平衡各层延迟，最终生成高效的流水线设计。

2）模型切分阶段。为实现模型在 FPGA 集群上的分布式部署，工具将原始模型切分为多个子模型，

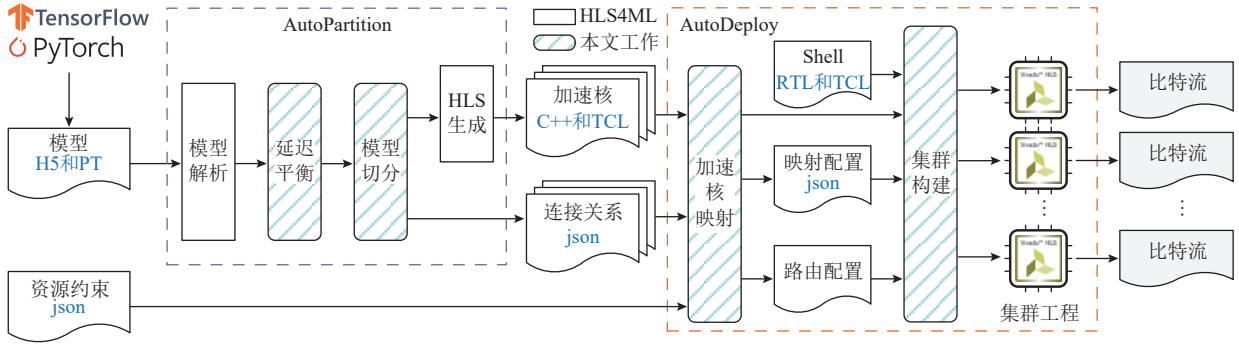


Fig. 5 TH-FCAD automated deployment framework

图 5 TH-FCAD 自动化部署框架

每个子模型被封装为一个加速核，并插入标准化接口以对接流交换模块。通过流交换模块，加速核间可以实现点对点的高效数据传输，从而支持在 FPGA 集群上的灵活部署与协同推理加速。

3)加速核映射阶段。模型切分得到的加速核连接图表示了核间数据传输结构。为优化集群通信效率，采用集群感知的加速核映射算法，获得加速核到 FPGA 的最佳映射，并生成包含 FPGA 分配信息、各 FPGA 流交换模块配置信息以及 NRC 路由表信息的映射配置文件。

4)集群构建阶段。工具为集群中的每个 FPGA 创建独立的 Vivado 工程，完成 Shell 模块实例化、Role 模块集成以及约束文件生成等关键步骤。将加速核 IP 与 Shell 设计无缝拼接，并调用 Vivado 工具链，最终生成可部署的比特流文件。

TH-FCAD 由 AutoPartition 和 AutoDeploy 这 2 个子模块构成。其中，AutoPartition 模块主要负责前 2 个阶段的工作，将模型描述文件转换成加速核的 HLS 实现。AutoDeploy 模块则实现了工作流程中后 2 个阶段的任务，负责多核设计的自动化划分与映射，并生成面向多 FPGA 集群的可执行位流文件。

3.1 延迟平衡与模型切分

本文开发的 AutoPartition 工具在 HLS4ML[20] 基础上实现了神经网络多核加速架构的自动生成，主要改进包括：1)提出了基于性能-资源模型的延迟平衡方法，可自动优化各层重用因子(reuse factor, *RF*)，解决了 HLS4ML 需手动调整 *RF* 的问题，实现延迟均衡与推理性能提升；2)提出模型切分方法，突破 HLS4ML 仅面向单 FPGA 设计的限制，支持多加速核协同推理，并实现 FPGA 集群的分布式部署能力。

3.1.1 延迟平衡

HLS4ML 架构中，卷积与全连接层均由矩阵向量乘(matrix-vector multiplication, MVM)部件实现，其规模与延迟由 *RF* 调节。*RF* 定义为完成 1 次 MVM 计算时单个 DSP 的复用次数。对于卷积层，单次等效规模为 $f_h \cdot f_w \cdot in_c \cdot out_c$ 的 MVM 计算。所需 DSP 数量为 $f_h \cdot f_w \cdot in_c \cdot out_c / RF$。其中，$f_h$，$f_w$，$in_c$，$out_c$ 分别为卷积核高度、宽度、输入通道数和输出通道数。

考虑数据位宽影响，DSP 消耗与 *RF* 的关系如式(1)所示。

$$R^{\mathrm{DSP}}(RF) = \gamma \cdot f_h \cdot f_w \cdot in_c \cdot out_c / RF \,, \tag{1}$$

其中系数 γ 为单个 DSP 处理字长与目标精度的比值。采用 8b 量化方案时，利用数据拼接优化，单个 DSP 可单周期并行完成 2 次 8b 乘法，此时 γ=0.5。

RF 直接影响层延迟。卷积层需执行 $out_h \cdot out_w$ 次 MVM 计算，其中 out_h，out_w 为输出特征图的高度和宽度。基于 Vivado HLS 综合仿真分析 MVM 时序行为，其时钟周期消耗包含初始化、数据读取及乘累加执行等阶段，由此得卷积层延迟模型如式(2)所示(单位：时钟周期)。

$$L(RF) = \begin{cases} out_h \cdot out_w \cdot (RF + 6), & RF > 1 \,, \\ out_h \cdot out_w \cdot 3, & RF = 1 \,。 \end{cases} \tag{2}$$

此外，为保证规则 DSP 阵列结构，*RF* 需满足式(3)约束。

$$(f_h \cdot f_w \cdot in_c \cdot out_c) \bmod RF = 0 \,。 \tag{3}$$

全连接层可视为 $f_h = f_w = out_h = out_w = 1$ 的特例，前述分析同样适用。

为最大化流水线执行效率，本文提出一种延迟平衡算法来自动优化各层 *RF*，其具体实现如算法 1 所示。该算法首先遍历计算层获取特征图尺寸并生成满足式(3)的 *RF* 候选集；随后计算各层在最小 *RF* 下的延迟，取其最大值为基准延迟 L_{base}；最后，将基准层 *RF* 设为最小值，调整其余层 *RF* 使其延迟逼近 L_{base}，并在候选集生成阶段实施资源约束检查。默认情况下，限制单层消耗的 DSP 资源不超过一块 FPGA 总资源的 80%，以实现性能与资源的最佳平衡。

算法 1. 延迟平衡算法。

输入：神经网络层集合 *Layers*，FPGA 可用 DSP 资源 C^{DSP}；

输出：重用因子配置 RF_{opt}。

① foreach $l_i \in Layers$ do

② 提取特征图尺寸；

③ $\mathcal{RF} \leftarrow$根据式(3)初始化候选RF集合；

④ $\mathcal{RF}^i_{\mathrm{cand}} \leftarrow \{rf \in \mathcal{RF} | R^{\mathrm{DSP}}(rf) \leqslant 0.8 \cdot C^{\mathrm{DSP}}\}$; /*过滤掉不满足 DSP 约束的候选 *RF**/

⑤ $L_i^{\min} \leftarrow \min\limits_{rf \in \mathcal{RF}^i_{\mathrm{cand}}} L_i(rf)$; /*根据式(2)计算该层的最小延迟*/

⑥ end for

⑦ $L_{\mathrm{base}} \leftarrow \max(L_i^{\min})$; /*确定基准延迟*/

⑧ foreach $l_i \in Layers$ do

⑨ $RF_i^* \leftarrow \arg\min\limits_{r \in \mathcal{RF}^i_{\mathrm{cand}}} |L_i(rf) - L_{\mathrm{base}}|$; /*选择最优 *RF* 值*/

⑩ end for

⑪ return $RF_{\mathrm{opt}} \leftarrow \{RF_1^*, RF_2^*, \cdots, RF_N^*\}$。

3.1.2 模型切分

原始的 HLS4ML 工具将神经网络整体转换为单一加速核，仅支持单 FPGA 部署，无法满足大规模神经网络在 FPGA 集群上的分布式加速需求。为此，本文提出一种模型切分方法，通过层次化切分将神经网络模型分解为多个子模型，每个子模型实现为一个独立加速核，并采用标准 AXI-Stream 接口实现高效的核间通信，从而支持在 FPGA 集群的灵活部署。如图 6 所示。

模型切分首先对神经网络进行层融合。基于其典型结构(包含卷积层、全连接层、激活层和池化层)，以计算密集的卷积层和全连接层为核心，将相邻的轻量级池化层和激活层合并。融合后的层级延迟和资源需求计算方法与前文一致。随后，将神经网络切分为多个子模型，每个子模型包含若干相邻的层并自动转化为对应加速核的 HLS 实现。

加速核采用细粒度流水线结构实现层间并行计算(如图 6 左下图)。核内各计算层均采用数据流驱动的工作模式，一旦接收到满足 MVM 计算所需的最小数据量时立即启动计算，从而形成极小启动间隔的计算流水线。加速核的延迟由其中延迟最大的层决定，即

$$L_{\mathrm{total}} = \max(L_1, L_2, \cdots, L_N) \text{。} \tag{4}$$

切分后的加速核采用点对点数据流通信机制，并在核间构建粗粒度流水线(如图 6 右下图)。具体实现方式为：在每个加速核的输出端加入数据流转换模块，将数据流统一封装为有效载荷为 512 b 的数据包，并通过 AXI-Stream 协议传输到下一级加速核。该机制支持输出与输入操作并行执行，实现核间流水线的并行计算。由于数据需要等待足够的计算结果写入缓存后才能进行打包传输，因此形成的流水线粒度相对较粗。

整体流水线的阶段数等于加速核的数量。为确保流水线效率，各传输路径需保持加速核数量均衡，以避免因路径长度不均导致的流水线阻塞。

对于仅包含链式连接的神经网络模型(没有分支连接)，本文提出一种基于资源约束的贪心合并策略实现模型切分：首先将每一层初始化为一个加速核并评估其 DSP 资源使用量；随后迭代合并资源消耗最少的相邻加速核对，约束条件为合并后 DSP 资

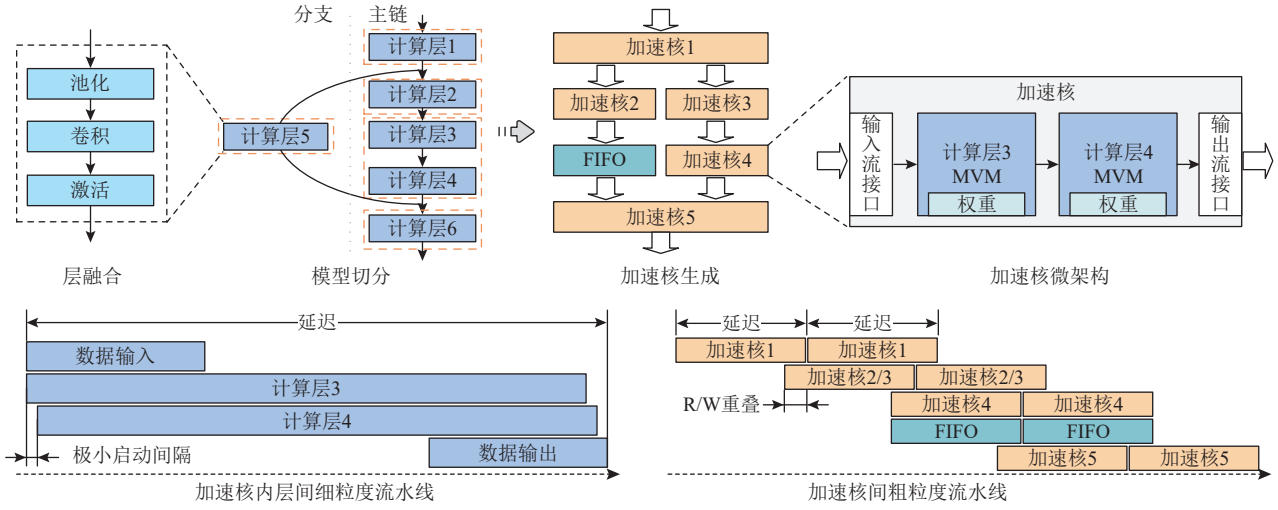


Fig. 6 Model partition acceleration kernel and its latency

图 6 模型切分加速核及其延迟

源不超过单块 FPGA 的α倍;重复上述过程,直到无可合并的加速核为止。

因此,参数α决定了加速核规模,其取值直接影响系统性能与后续加速核映射的灵活性。增大α可实现更细粒度流水线并降低端到端延迟,但若单个加速核的资源消耗过高,将限制其在集群中的灵活放置,导致后续映射算法难以通过核间组合优化资源利用率与通信开销。实验表明,将加速核的资源消耗控制在单块 FPGA 总资源的 30% 左右,既能保证核内包含多个层以实现细粒度流水线并行,又能为后续映射提供足够的组合空间(通常单块 FPGA 可容纳 3 个核),从而在系统性能与映射灵活性之间取得良好平衡。因此,本文设定α=0.3 作为经验阈值,后续实验结果也验证了该取值的有效性。需要说明的是,由于延迟平衡阶段已允许单层资源消耗最高可达 80%,因此切分过程中对于资源超过 30% 的层不再进行合并,而是直接将其封装为一个独立的加速核。

对于包含分支结构的神经网络(如 ResNet 的残差连接),首先将模型输入到输出包含层数最多的路径作为主链,其余路径作为分支,并将主链按照前述切分方法处理。而分支上的层数量少于主链,本文通过在分支末端引入 FIFO 缓存使其与主链保持相同流水线深度,从而消除流水线不均衡问题,确保核间流水线高效运行。然而,多分支汇聚可能引发死锁:若分支数据先于主链到达汇聚点,汇聚点因缺少主链输入而通过反压机制阻塞分支,分支 FIFO 填满后继续向上游传递反压,最终可能导致全局阻塞。死锁避免的关键在于确保分支 FIFO 具有足够的深度,以缓存主链延迟差内产生的全部数据。假设主链与分支延迟差为ΔL(含可能引入的跨 FPGA 通信延迟),分支输出吞吐量为T,则所需 FIFO 深度为$\Delta L \cdot T$。基于此,AutoDeploy 在完成映射后,根据实际延迟差自动计算各分支 FIFO 的深度需求,并额外增加 10% 的设计冗余。该设计与图拓扑无环性验证、信用流控协同作用,理论上可彻底避免死锁。我们通过仿真进一步验证了这一结论。

3.2 AutoDeploy 实现

AutoPartition 工具已经将完整的神经网络切分成多个延迟均衡的子模型,每个子模型包含若干层,并对应一个加速核实现。此时,已经具备手动部署条件,然而要实现高效的自动化部署仍面临两大关键挑战:1)如何将众多加速核合理映射到集群中的各 FPGA 节点是一个复杂的问题,需要综合考虑 FPGA 的资源约束、集群的通信开销等因素。传统手动映射方法难以实现最优权衡。2)尽管 Shell/Role 架构简化了通信设计,但针对集群的每个 FPGA 手动创建工程,并拼接加速核 IP 和 Shell 接口仍是一项复杂繁琐的任务。为此,本文开发了 AutoDeploy 工具,其核心技术包括:

1)集群感知的加速核映射算法。该算法综合考虑了 FPGA 资源约束和集群通信拓扑结构,能够自动生成最优映射方案,在满足资源限制的前提下,最小化通信开销,同时提升整体推理吞吐量和资源利用效率。

2)自动化集群构建。根据映射方案,工具自动为每个 FPGA 创建工程完成加速核 IP 与 Shell 架构的集成,并通过调用 Vivado 工具链生成可部署的比特流文件。这一过程有效隐藏了创建 FPGA 工程的复杂细节,提高了集群部署效率。

3.2.1 集群感知的加速核映射

加速核间的数据流连接可建模为有向无环图(directed acyclic graph, DAG),其在 FPGA 集群上的映射问题可转化为带资源约束的图划分问题。图划分问题属于 NP 难问题[42],其计算复杂度随节点规模与划分数量的增加呈指数级增长。对于小规模问题,可通过整数规划或枚举法获得精确解;而对于大规模场景,需采用启发式算法进行近似求解。

1)问题建模

加速核映射问题可被建模成带约束的图划分优化问题,具体形式化定义如下:

① 符号定义。以每个加速核作为图的节点,节点集合记为V。以加速核的数据流向作为边,边集合记为E。边权重w_{ij}表示加速核i到j的归一化通信数据量(单位为 B),具体而言,w_{ij}表示加速核i的输出特征图(也是j的输入特征图)字节大小除以归一化常量η,η取所有边输出特征图字节数的平均值,w_{ij}的计算过程如式(5)所述。

$$w_{ij} = \frac{B_{ij}}{\eta}, \tag{5}$$

其中

$$\begin{aligned} B_{ij} &= F_{\mathrm{w}}^{ij} \cdot F_{\mathrm{h}}^{ij} \cdot F_{\mathrm{c}}^{ij} \cdot D, \\ \eta &= \frac{1}{|E|} \sum_{\langle i,j \rangle \in E} B_{ij}, \end{aligned} \tag{6}$$

B_{ij}表示特征图的字节数,F_{w}^{ij},F_{h}^{ij},F_{c}^{ij}分别表示核i到j传输特征图的宽度、高度和通道数,D为单个数据元素的字节大小(8b 数据取$D=1$)。

② 加速核连接关系图记为$G(V,E)$。定义分配向

量$\boldsymbol{x}=(x_1,x_2,\cdots,x_n)$，其中$x_i\in\mathbb{N}_+$表示节点$i\in V$分配的FPGA编号。定义$S_k=\{i|x_i=k\}$为分配到FPGA的$k$的节点集合。在FPGA集群中，编号为$k$的FPGA属于第$\lfloor(k-1)/4\rfloor$组（每组包含4个FPGA），组内通过LVDS互连，组间通过GTH互连。

③ 输入。加速核连接关系图$G(V,E)$。节点i的资源消耗$r_i^{(t)}$和单个FPGA的资源上限$C^{(t)}$，其中$t\in\{\text{DSP,LUT,FF,BRAM}\}$。

④ 输出。划分方案$P=\{S_1,S_2,\cdots,S_K\}$，其中$S_k=\{i|x_i=k\}$表示划分到第k个FPGA上的节点集合。划分方案需满足$\bigcup S_k=V$且$S_p\cap S_q=\varnothing,(\forall p\neq q)$，其中$K$为所使用的FPGA数量（即$K=\max\limits_i x_i$）。

⑤ 约束条件。对于任意FPGA节点k以及资源类型$t\in\{\text{DSP,LUT,FF,BRAM}\}$，满足约束：

$$R_k^{(t)}=\sum_{i\in S_k}r_i^{(t)}\leqslant 0.8C^{(t)}\,, \tag{7}$$

即确保每个FPGA每种资源使用率均不超过80%。

⑥ 目标函数。本文研究是以最小化跨FPGA通信开销与硬件资源消耗作为综合优化目标。在系统拓扑特性约束下定义目标优化函数为：

$$\begin{aligned}W_{\mathrm{L}}&=\sum_{\langle i,j\rangle\in E,\lfloor(x_i-1)/4\rfloor=\lfloor(x_j-1)/4\rfloor,x_i\neq x_j}w_{ij}\,,\\ W_{\mathrm{G}}&=\sum_{\langle i,j\rangle\in E,\lfloor(x_i-1)/4\rfloor\neq\lfloor(x_j-1)/4\rfloor}w_{ij}\,,\\ Cost&=\min(\beta\cdot W_{\mathrm{L}}+(1-\beta)\cdot W_{\mathrm{G}}+\lambda\cdot K)\,,\end{aligned} \tag{8}$$

其中W_{L}表示通过组内LVDS互连的通信开销，W_{G}则表示通过全局GTH互连的通信开销，K为FPGA总数，作为惩罚项用于优化资源消耗。权重系数$\beta(0<\beta<1)$用于平衡组内与全局通信，由于LVDS互连（组内通信）性能优于GTH互连（全局通信），设置$\beta<0.5$以优先优化W_{G}；但β不宜过小，以免弱化对W_{L}的优化。我们对β在0.2~0.5区间内的取值进行了多次迭代尝试，发现在$\beta=0.4$时，可使用模拟退火算法求解得到显著的优化效果。在该配置下，全局通信开销W_{G}降低了24.66%，组内通信开销W_{L}降低了4.52%，取得了相对最优的权衡效果，并使总通信开销降低了13.62%，实现了对通信性能的整体优化。正则化参数λ用于调节资源消耗惩罚项K（即所需FPGA总数），以优化硬件资源利用率。其取值原则是使惩罚项$(\lambda\cdot K)$与通信开销项$(\beta\cdot W_{\mathrm{L}}+(1-\beta)\cdot W_{\mathrm{G}})$在数值量级上可比。由于$W_{\mathrm{L}}$和$W_{\mathrm{G}}$值均已进行归一化处理，直接取$\lambda=1$即可满足这一条件。

通过上述建模，加速核映射问题可转化为在资源约束下的图$G(V,E)$最优划分问题，以实现资源分配与通信代价的联合优化。

2）问题求解

本文采用模拟退火（simulated annealing, SA）算法求解FPGA集群上的加速核映射问题，求解过程如算法2所示。该算法以加速核连接关系图和各加速核的资源需求为输入，输出满足资源约束且通信开销最优的划分方案，具体步骤如下。

① 初始化。通过贪心法生成初始可行解。依次将每个节点$i\in V$分配至第1个满足资源约束条件的FPGA的k，即$\forall t,R_k^{(t)}+r_i^{(t)}\leqslant 0.8C^{(t)}$，若所有现有FPGA均不满足条件则创建新的FPGA。该过程持续至所有节点完成映射，确保每个FPGA的资源使用率不超过80%。

② 邻域解生成。通过合并子集和迁移节点2种随机操作生成新的邻域解。

合并子集时，随机选取FPGA的k_1和k_2（$k_1\neq k_2$）。若满足约束$\forall t,R_{k_1}^{(t)}+R_{k_2}^{(t)}\leqslant 0.8C^{(t)}$，则删除FPGA的$k_2$，并将所有$k_2$上的节点迁移至$k_1$，同时，更新资源信息：$R_{k_1}^{(t)}\leftarrow R_{k_1}^{(t)}+R_{k_2}^{(t)},R_{k_2}^{(t)}\leftarrow 0$。

迁移节点时，随机选择节点$i\in V$（当前位于的FPGA的k_1）。筛选出所有满足资源约束条件的FPGA的$k\neq k_1$，即$\forall k\neq k_1,R_k^{(t)}+r_i^{(t)}\leqslant 0.8C^{(t)}$。若现有FPGA均无法容纳该节点，则创建新的FPGA。随机选择一个候选FPGA的k_{new}，并将节点i迁移到k_{new}。相应地，更新节点分配关系$x_i\leftarrow k_{\text{new}}$，以及资源使用信息$R_{k_1}^{(t)}\leftarrow R_{k_1}^{(t)}-r_i^{(t)},R_{k_{\text{new}}}^{(t)}\leftarrow R_{k_{\text{new}}}^{(t)}+r_i^{(t)}$。

③ 接受准则。根据式（8）计算目标函数变化量，$\Delta Cost=Cost_{\text{new}}-Cost_{\text{old}}$，并依据概率$\exp(-\Delta Cost/T)$决定是否接受新邻域解。

④ 终止条件。当温度降至预设阈值$T_{\min}$时，停止迭代并返回记录的最优解。

算法2. 基于模拟退火的加速核映射算法。

输入：加速核连接关系图$G(V,E)$，节点的资源消耗$r_i^{(t)}$，单个FPGA的资源上限$C^{(t)}$；

输出：加速核到FPGA集群的映射方案$P_{\text{best}}=\{S_1,S_2,\cdots,S_K\}$。

① $P_{\text{best}}\leftarrow P\leftarrow GreedyPartition(G)$;

② $Cost_{\text{best}}\leftarrow Cost\leftarrow Cost(P)$;

③ $T\leftarrow 100\cdot Cost$;

④ while $T>T_{\min}$ do

⑤ for $1,2,\cdots,M$ do

⑥ $P'\leftarrow$ 随机生成邻域解;

⑦ $Cost'\leftarrow Cost(P')$;

⑧ $\Delta Cost \leftarrow Cost' - Cost$;
⑨ if $rand() < \exp(-\Delta Cost/T)$ then
⑩ $P \leftarrow P'$;
⑪ end if
⑫ if $Cost' < Cost_{best}$ then
⑬ $Cost_{best} \leftarrow Cost'$;
⑭ $P_{best} \leftarrow P'$;
⑮ end if
⑯ end for
⑰ $T \leftarrow 0.95 \cdot T$;
⑱ end while
⑲ return P_{best}。

3.2.2 自动化集群构建工具

本文实现的自动化集群部署工具 AutoDeploy 的架构如图 7 所示。该工具采用模块化设计与 Python-Tcl 协同工作机制，实现了从加速核到物理部署的全流程自动化。其中，Python 控制层负责高层逻辑实现，Tcl 驱动层负责底层物理实现。Python 控制层运行集群感知的加速核映射算法，生成加速核映射描述文件，并同步生成 FPGA 集群的 NRC 配置参数、Shell 模块配置以及加速核拼接的 Tcl 命令。Tcl 驱动层则负责动态创建多个 FPGA 工程结构，实例化加速核 IP 与 Shell 模块 IP，自动完成流交换模块的 AXI-Stream 端接口与加速核接口的拼接，实现协议转换模块与物理通道的绑定，并生成时序约束。最终通过调用 Vivado 工具链生成可部署的比特流文件。该混合架构在保持算法灵活性的同时，充分调用 Vivado 工具链的物理优化能力，实现了高效自动化部署。

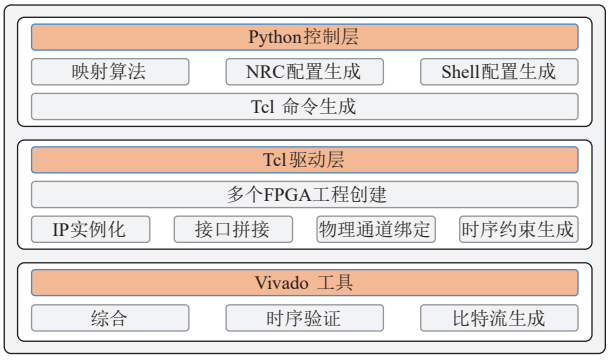


Fig. 7 Architecture of the automated cluster building tool

图 7 自动化集群构建工具架构

4 实验评估

4.1 实验平台

本文构建了基于 16 块 Virtex UltraScale+系列 FPGA 芯片的异构计算集群系统。系统中 4 个 FPGA 作为一个 Blade，采用 LVDS 的全互连实现板内通信，FPGA 和 NIC 均通过光纤与 NRC 互连。为进行对比实验，同时租赁了 Nvidia RTX 5090 云服务器平台。集群网络采用集成了 NRC 与 NIO 芯片的 TH-Express 高速互连架构，并搭载了我们开发的软件通信栈。开发环境基于 Vivado 2020.2 和 Vitis 2023 工具链构建。

4.2 集群互连性能及通信接口资源消耗分析

表 1 详细列出了集群中不同互连链路的性能测试结果。GTH 链路总带宽达 80 Gbps，端到端延迟为 1.56 μs。主机 CPU 与 FPGA 间的通信虽然采用相同的 GTH 物理链路，但由于软件通信栈的额外开销，延迟增加至 18.2 μs。LVDS 链路采用 8 组×8 通道×1.6 Gbps 的配置，最大带宽可达 100 Gbps。FPGA 内部的流交换模块，支持全端口的并行传输，其带宽由端口位宽和频率决定，而延迟则主要取决于仲裁延迟，经测试最大不超过 16 个时钟周期。

Table 1 Latency and Bandwidth of Cluster Links

表 1 集群链路的延迟和带宽

链路	延迟/μs	带宽/Gbps
Host-FPGA	18.2	80
FPGA-FPGA (GTH)	1.56	80
FPGA-FPGA (LVDS)	0.21	100
FPGA 内部	0.08	-

注：-表示不适用。

表 2 展示了 FPGA 板卡资源分布及 Shell/Role 架构资源消耗情况。其中，Shell 部分设计固定，各节点资源消耗基本一致。流交换模块资源消耗与端口数量正相关，表中数据基于 16 端口配置。实验数据显示，Shell 架构仅消耗约 5% 的片上资源。

Table 2 Resource Consumption of Shell Architecture

表 2 Shell 架构的资源消耗 %

模块	LUT	FF	BRAM	DSP
Shell	4.73	0.96	0.49	0
流交换模块	0.35	0.27	0	0

4.3 自动化框架有效性评估及与相关工作比较

为了验证本文提出的自动化框架在不同规模神经网络模型上的适用性，我们选取了 ResNet-8[43]，SqueezeNet[44]，ResNet-50[45] 三种不同规模的模型进行实验评估，所有实验均在 Batch-1 配置下进行，结果如表 3 所示。我们首先使用 ResNet-8 模型验证自动化框架的有效性。ResNet-8 模型规模较小，可以使用

Table 3 Experimental Results of Different Models

表 3 不同模型的实验结果

模型（计算操作数）	平台	框架	稀疏度/%	位宽/b	频率/MHz	延迟/ms	吞吐量/FPS
ResNet-8（25.4 MOP）	Intel i7-12700H	Keras	0	32		6.4	
	RTX 5090	Keras	0	32		1.7	
	1 个 FPGA	HLS4ML	0	16	156	**0.132**	7 575
	3 个 FPGA	本文	0	16	156	**0.051**	19 607
SqueezeNet（1.0 GOP）	Intel i7-12700H	Keras	0	32		8.5	
	RTX 5090	Keras	0	32		3.4	
	6 个 FPGA	文献 [46]	0	8	200	3.9	251
	4 个 FPGA	本文	0	8	125	**1.24**	806
ResNet-50（8.1 GOP）	Intel i7-12700H	Keras	0	32		56.3	
	RTX 5090	Keras	0	32		6.9	
	1 Alveo U250	文献 [27]	0	1	195	1.9	2 703
	1 Alveo U200	AGNA[47]	0	8	200	4.55	
	2 Alveo U280	Elastic-DF[35]	0	1	220	1.55	3 617
	2 Stratix 10 MX 2100	HPIPE[48]	85	16	400	1.75	2 621
	12 个 FPGA	AIgean[34]	0	8	200	1.9	660
	14 个 FPGA	贪心映射	0	8	125	**1.42**	704
	12 个 FPGA	SA 映射（本文）	0	8	125	**1.36**	735

原始 HLS4ML 工具在单个 FPGA 上实现，但需要增大参数 *RF* 以实现较高的资源复用，导致较大的推理延迟。相比之下，采用本文提出的自动化框架，将模型拆分至 3 个 FPGA 进行分布式部署，虽然引入了一定的通信开销，但能显著提升模型的整体推理性能。

具体而言，我们使用官方提供的 HLS4ML 工具对 ResNet-8 模型进行了手动优化。通过手动调整 *RF* 参数，成功实现了 ResNet-8 模型在单个 FPGA 上的部署。采用本文框架后，系统自动调整 *RF* 并切分模型结构，成功实现了模型在 3 个 FPGA 上的分布式部署。我们对比了 CPU 和 GPU 在相同任务下的性能表现。实验结果表明，单个 FPGA 版本相对 CPU 和 GPU 分别获得了 48.5 倍和 12.9 倍的加速效果。而 3-FPGA 分布式方案相对单个 FPGA 实现获得了 2.6 倍的性能提升。该结果充分证明了自动化框架在多个 FPGA 并行计算中的有效性。

为系统验证框架对不同规模网络的适应性，在 ResNet-8 验证基础上，我们进一步选取了模型规模适中但计算密集的 SqueezeNet 进行测试。实验结果表明，通过自动优化各层加速架构，本框架成功实现了 SqueezeNet 在 4 个 FPGA 芯片上的高效分布式部署，最终推理延迟仅为 1.24 ms，较 CPU 和 GPU 方案分别提升 6.9 倍和 2.7 倍。对比现有工作，文献 [46] 虽然在 6 个 FPGA 上部署了相同模型，并达到 200 MHz 工作频率，但由于缺乏延迟平衡策略，未能自动优化参数 *RF*，导致各层计算延迟不均衡，显著降低了流水线效率。与之相比，我们的方案在 FPGA 数量更少的情况下，仍实现了 3.1 倍的性能优势，充分证明了本文自动优化框架的有效性。

为进一步验证自动化框架在大规模神经网络模型上的性能表现，我们以 ResNet-50 模型为基准进行了实验。该模型包含 53 个卷积层和 1 个全连接层，在实验中，我们通过延迟平衡方法自动优化每层的 *RF*，并利用层次化切分工具将模型分解为 27 个独立的加速核，运行时钟频率设置为 125 MHz。每个加速核均可独立运行，整体延迟由最长的加速核延迟、网络通信延迟以及主机 CPU 与 FPGA 之间的传输延迟共同决定。对比实验中，贪心算法需消耗 14 个 FPGA，而本文提出的集群感知映射算法仅需 12 个 FPGA 即可实现部署。

测试结果表明，12 个 FPGA 的部署方案实现了 1.36 ms 的推理延迟，计算性能达到 5.95 TOPS（8.1 GOP·735 FPS）。与 CPU 和 GPU 相比，分别获得了 41.4 倍和 5.1 倍的加速效果。

文献 [27] 采用 1b 的权重量化策略在单个 FPGA 上部署 ResNet-50 模型，实现了 1.9 ms 的推理延迟。

其 1b 量化策略虽能通过细粒度流水线实现更高吞吐量，但其模型精度仅为 67.27%，低于本文 8b 量化模型 75% 的推理精度。AGNA[47] 采用 PEA 架构，受限于片外存储访问开销，在 U200FPGA 板卡上的推理延迟高达 4.55 ms，显著落后于本文 DataFlow 架构的低延迟表现，这验证了 DataFlow 架构在延迟优化上的优势。

我们进一步对比了其他现有的多 FPGA 加速框架的性能，Elastic-DF[35] 和 HPIPE 框架[48-49] 分别采用 1b 量化与 85% 稀疏度模型，通过轻量化设计在 2 块 FPGA 实现部署。这类方案本质仍聚焦小规模模型加速，虽能获得较高吞吐量，但延迟表现不及本文框架，并且产生较大的推理精度损失。作为当前 SOTA 研究，AIgean[34] 在 12 个 FPGA 以太网集群上实现了 1.9 ms 的纯 FPGA 模式推理延迟，但引入 CPU 通信后延迟骤增至 7 ms。相较而言，本文框架的性能相比其提升了 28.4%。

综上所述，本文提出的自动化框架在大规模神经网络加速中展现显著优势，尤其是在低延迟和高吞吐量方面，优于现有的相关工作。

4.4 集群感知映射算法性能分析

本文提出的集群感知的加速核映射算法(SA)成功实现了 ResNet-50 模型在 12 个 FPGA 上的高效部署。实验结果表明，和传统的贪心算法相比较，该优化方法表现出明显的优势：首先，贪心算法由于易陷入局部最优解，导致需要 14 个 FPGA 才能容纳所有加速核，且因通信带宽受限使得片间延迟增加至 1.42 ms；其次，优化算法通过全局启发式搜索策略，有效降低了 GTH 和 LVDS 链路的传输数据量，分别减少了 24.66% 和 4.52%，总体片外数据传输量减少 13.62%，如图 8 所示。

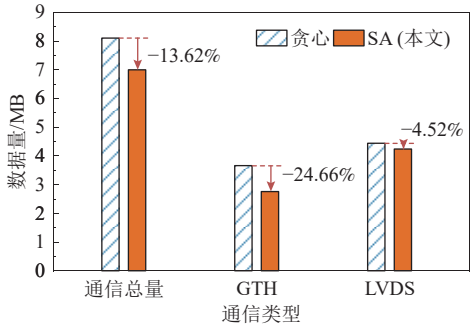


Fig. 8 Comparison of communication data volume with two mapping algorithms for ResNet-50

图 8 ResNet-50 在 2 种映射算法下的通信数据量对比

图 9 对比了 2 种映射算法的资源消耗情况，斜线填充条柱为贪心算法的 14 个 FPGA 实现，实心条柱为模拟退火算法的 12 个 FPGA 实现。从图 9 中可以看出：1)集群感知映射方法通过优化加速核划分，使 FPGA 平均资源利用率提升了 9.3%；2)在获得更高计算性能的同时，硬件成本降低 14.3%。实验结果表明，通过合理划分加速核并优化加速核映射，基于模拟退火算法的集群感知加速核映射方法能够更有效地利用有限的 FPGA 资源(特别是关键 DSP 资源)，在保持高性能的同时减少硬件开销。需要强调的是，本文研究的卷积神经网络加速任务属于计算密集型任务。在实现过程中，我们为提高系统性能，明确限定使用片上 DSP 实现卷积计算中的乘法操作，这一

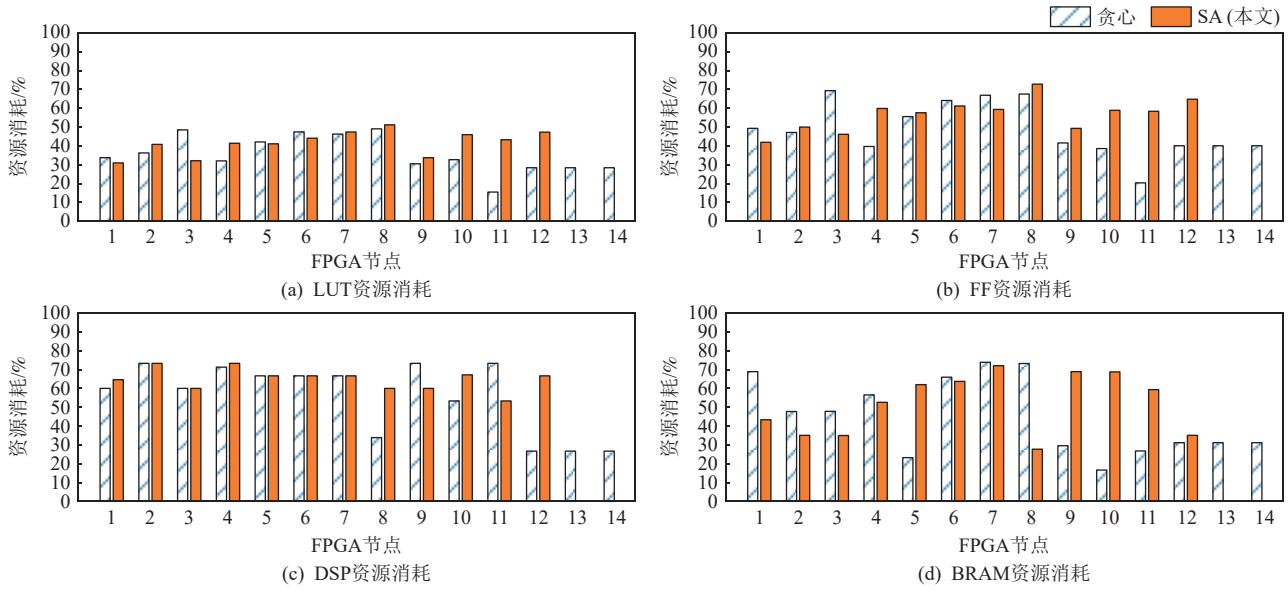


Fig. 9 Comparison of resource usage with two mapping algorithms for ResNet-50

图 9 ResNet-50 在 2 种映射算法下的资源消耗情况

策略显著降低了对 LUT 资源的依赖，导致 LUT 利用率维持在较低水平。此外，为确保时序收敛，我们将单个 FPGA 中各类资源（包括 DSP, LUT, FF, BRAM）的消耗上限设定为 80%。映射后，DSP 资源的峰值消耗达到 73.3%（平均 64.9%），明显高于其他资源。

为进一步验证本文所提出方法的有效性，我们将其与现有代表性多个 FPGA 划分方法进行深入比较。在现有工作中，Algean 框架[34] 的划分过程依赖人工经验进行任务分配，缺乏自动寻优机制，难以保证划分质量且无法适应不同模型与集群规模。HPIPE[48-49] 虽采用贪心策略实现了自动化划分，但其目标仅为满足单个 FPGA 资源约束，算法仅针对链式拓扑设计，难以扩展到更大规模集群；同时，贪心策略仅依据当前层的资源可用性做局部决策，无法从全局视角优化资源利用，易陷入局部最优。Elastic-DF[35] 采用整数线性规划求解划分问题，理论上可获得全局最优解，但其算法主要面向芯片内部多 Die 划分，优化目标设定为最小化跨 Die 切边数量。在多个 FPGA 场景下，该策略难以直接等价于通信开销最优，且无法区分不同物理链路（如 GTH 与 LVDS）的性能差异，因而难以适配异构互连的集群架构。

相比之下，本文提出的集群感知映射算法通过设置权重系数β（实验中β=0.4）区分 GTH 全局网络与 LVDS 局部链路的性能差异，有效避免了切边数量优化目标的局限性。与 HPIPE 的贪心策略不同，本算法基于模拟退火进行全局启发式搜索，能够在满足资源约束的前提下系统探索划分空间，获得更优的映射方案。

值得注意的是，本文方法着眼于系统级优化，通过优化加速核的物理部署位置减少跨节点通信总量。底层通信实现上，当前框架为每个连接预设固定时分复用（time division multiplexing, TDM）比率，虽简化了链路调度设计，却限制了物理链路通信潜力。近期研究 RTMF[50] 提出基于路由权重的竞争性 TDM 分配算法，通过精细调度 TDM 比率降低共享链路传输延迟。与本文相比，RTMF 侧重链路层面优化通信执行，而本文侧重系统层面减少通信需求，二者互为补充。未来工作可将二者结合，在集群感知映射算法中引入含 TDM 比率与带宽的精细链路模型，进一步降低关键路径传输延迟，实现系统级与链路级跨层次协同优化。

4.5 延迟平衡优化分析

在实验中，我们通过延迟平衡方法调整每层的重用因子，使各层的计算延迟和吞吐量基本保持一致，从而消除性能瓶颈并提升资源利用效率。图 10 对比了延迟平衡前后各层的延迟差异，在延迟平衡优化前，因各层特征图与卷积核计算规模不同导致延迟差异显著（最大值为 15.7 ms，最小值为 0.4 ms），这会降低资源利用率和流水线效率。在延迟平衡优化后，各层的延迟趋于一致（最大值为 1.26 ms，最小值为 0.8 ms），最大延迟仅为原来的 8%，差异缩小为原来的 3%，显著优化了系统性能。

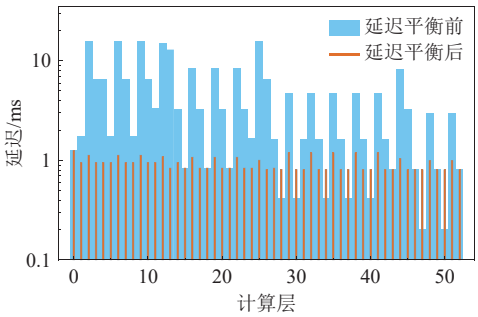


Fig. 10 Comparison of layer-wise latency before and after latency balancing in ResNet-50

图 10 ResNet-50 延迟平衡前后各层延迟对比

4.6 局部 LVDS 互连影响分析

本文提出的 2 级互连架构将每 4 个 FPGA 组成 1 个局部互连分组，组内通过 LVDS 实现高速直连通信。为评估局部 LVDS 互连对通信效率的提升效果，我们测量了不同局部互连规模下 FPGA 集群执行 ResNet-50 模型所需的总数据传输时间。

当局部互连规模为 1 个 FPGA 时，表示系统中仅使用全局 GTH 互连。局部互连规模为 4 个 FPGA 则对应本文默认架构。此外，为增强实验的普适性，我们还随机生成了多个具有神经网络结构特征的加速核连接图，数据传输需求也随机设定。每种配置下的通信时间取多次实验的平均值，结果如图 11 所示。图中“相对通信时间”定义为在特定局部互连规模下的 FPGA 与规模为 1 个 FPGA 时的通信时间比值。

实验结果表明，局部 LVDS 互连显著降低了通信延迟，尤其适用于具有局部连接特性的深度神经网络模型。在默认的 4 个 FPGA 互连的配置下，相对通信时间为 0.813，较无局部互连（即规模为 1 时）降低 18.7%。值得注意的是，随着局部互连规模的扩大，通信效率的提升呈现明显的边际递减趋势，然而，过大的局部互连规模不仅增加 LVDS 布线复杂度，还难以实现局部全互连网络。在实际设计中，需在互连规模、布线复杂度和通信效率之间寻求最优平衡。

4.7 硬件与算法贡献的消融分析

为明确 TH-FCAD 性能提升的来源，本文设计了

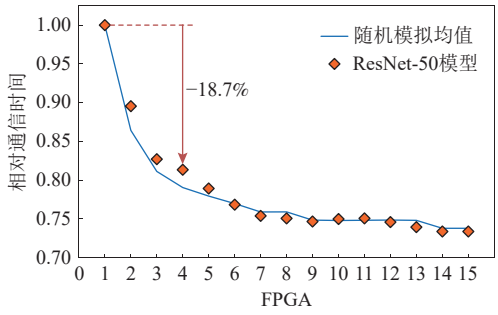


Fig. 11 Effect of local LVDS interconnect on communication time

图 11 局部 LVDS 互连对通信时间的影响

消融实验, 分别评估硬件平台与映射算法的独立贡献。如前文 4.5 节所述, 延迟平衡是确保多核流水线高效运行的前提。未使用延迟平衡时, 各层延迟差异高达 15 ms, 严重制约系统性能; 引入延迟平衡后, 各层延迟被有效均衡至 0.8~1.26 ms 范围内, 后续通信优化与资源调度才具备实际意义。因此, 所有消融实验均以延迟平衡为基础, 以保证实验设置的有效性。

在此前提下, 我们设置了 3 种实验配置: 配置 *a* 仅使用全局 GTH 互连模拟通用互连环境, 映射算法采用贪心策略; 配置 *b* 在配置 *a* 的基础上引入局部 LVDS 互连, 保留贪心映射算法; 配置 *c* 在配置 *b* 的基础上将映射算法替换为集群感知的 SA 映射算法。所有实验均在相同的 ResNet-50 模型和 Batch-1 配置下进行, 结果如表 4 所示。

Table 4 Comparison of Performance Across Different Hardware and Algorithm Configurations

表 4 不同硬件与算法配置下的性能对比

配置	互连方案	映射算法	FPGA 个数	延迟/ms
a	仅 GTH	贪心	14	1.47
b	GTH+LVDS	贪心	14	1.42
c	GTH+LVDS	SA	12	1.36

表 4 表明, 硬件与算法的协同优化带来了显著的性能增益。配置 *a* 仅依靠全局 GTH 互连和贪心映射, 在 14 个 FPGA 上实现了 1.47 ms 的推理延迟。与现有 AIgean 框架在以太网集群上实现的 1.9 ms 延迟相比, 配置 *a* 获得的 0.43 ms 提升主要归因于 TH-Express 高速互连硬件本身的低延迟特性。配置 *b* 将延迟进一步降低至 1.42 ms, 较配置 *a* 减少 0.05 ms, 这得益于局部 LVDS 互连带来的通信效率提升。配置 *c* 在此基础上采用集群感知的 SA 映射算法, 延迟降至 1.36 ms, 较配置 *b* 减少 0.06 ms, 同时所需 FPGA 数量从贪心算法的 14 个缩减至 12 个, 显著提升了资源利用率。

综合来看, TH-FCAD 相较于 AIgean 框架总延迟降低 0.54 ms。其中, 硬件层面(GTH 与 LVDS 互连)的低延迟特性贡献了约 88.9% 的延迟降低(0.48 ms/0.54 ms), 是系统性能提升的决定性因素; 而 SA 映射算法贡献了约 11.1% 的延迟降低(0.06 ms/0.54 ms)。此外, 算法还额外带来了硬件资源的节约。上述结果表明, TH-FCAD 的性能提升是以硬件低延迟特性为基石, 并通过算法创新进一步挖掘资源潜力与性能极限的协同优化结果。

4.8 死锁避免机制的仿真验证

本文构建了周期精确的 RTL 仿真环境, 以验证所提出 FIFO 深度设计方法的有效性以及系统的无死锁运行能力。针对可能发生死锁的分支汇聚结构, 我们配置了多组延迟差进行仿真测试。由于 ResNet-50 中分支路径通常比主链少 1~2 个加速核, 且单个加速核的计算延迟为 0.8~1.26 ms, 据此我们设定了 1.12 ms 与 1.84 ms 这 2 种典型延迟差。为进一步验证设计在更严苛条件下的鲁棒性, 额外增加了 2.68 ms (对应 3 个核延迟差)的边界测试用例在仿真环境中, 以系统能够支持的最大数据注入速率施加激励, 并在所有加速核的输入输出端部署性能监视器, 实时监控每个 FIFO 的瞬时占用率和最大占用深度。

表 5 汇总了不同延迟差配置下的仿真结果。结果显示, 在最大吞吐激励下, 各分支路径 FIFO 的最大占用深度均未超过设定值, 未出现写满现象, 且实测值与理论需求深度高度吻合。系统在整个运行周期内未发生死锁, 验证了所提出 FIFO 深度设计方法的有效性。

Table 5 FIFO Depth Analysis of Branch Merging Structures

表 5 分支汇聚结构 FIFO 深度分析

延迟差/ms	理论需求深度	设定深度	仿真最大占用深度	是否死锁
1.12	1 694	1 864	1 731	否
1.84	2 784	3 063	2 773	否
2.68	4 055	4 461	4 056	否

4.9 框架通用性与扩展性分析

本文提出的 TH-FCAD 自动化部署框架, 以“通信-计算解耦”与“分层抽象”为核心设计思想, 在体系结构层面实现了良好的硬件适应性与模型扩展性。当前实验基于天河高速互连网络集群展开, 并主要针对卷积神经网络进行了部署优化。然而, 得益于框架核心机制的通用性, TH-FCAD 具备支持更广泛的硬件平台与更大规模的模型(如 Transformer)的拓展

潜力。

4.9.1 硬件平台通用性

如 3.2 节所述，TH-FCAD 通过 Shell/Role 双层架构实现了通信与计算的解耦：Role 层完全聚焦于神经网络计算任务，并通过标准化的 AXI-Stream 接口与 Shell 层进行交互；Shell 层则负责封装所有底层通信细节，当前实现针对 TH-Express 网络进行了优化部署，集成了 GTH 网络栈和 LVDS 转换桥。该设计使得算法层面的贡献（如延迟平衡、模型切分和映射算法）与底层硬件无关，具备良好的通用性。

将 TH-FCAD 适配至通用以太网或 InfiniBand 集群时，仅需对 Shell 层及主机侧软件栈进行模块化替换，具体包括：1）替换 Shell 层中的网络协议栈模块。例如，将以太网 MAC/IP 核接入框架，负责将以太网报文转换为符合 Role 层要求的 AXI-Stream 协议数据流。2）相应地，主机侧的软件通信栈仅需将 TH-Express 专用接口替换为标准以太网 Socket 接口，同时将底层驱动替换为以太网驱动程序，无需改动上层通信逻辑。整个 Role 层（包含由 AutoPartition 生成的加速核）以及自动化工具链（如 AutoDeploy 的映射算法、流交换模块配置）均无需做任何修改。综上所述，TH-FCAD 是一个具备良好硬件适应性的通用框架，其设计思想为实现在不同互连架构上进行深度学习模型的分布式部署，提供了一套统一的自动化解决方案。

4.9.2 模型扩展性

TH-FCAD 不仅在硬件层面具备良好的适应性，其算法与架构设计也为支持更广泛的深度学习模型（如 Transformer）奠定了扩展基础。

在硬件层面，FPGA 集群采用的 2 级互连架构（全局通用互连+局部专用互连）兼具扩展性与高效性。全局互连支持集群规模的弹性扩展，能够灵活适应 Transformer 等大模型对大规模并行计算需求，确保计算资源可随模型规模线性增长。局部专用互连则为相邻的计算节点提供了低延迟、高带宽的数据传输通道，满足计算密集型模块间的高性能通信需求（如 Transformer 中矩阵乘法与 Softmax 操作间的数据交互）。此外，Shell/Role 设计实现了计算与通信的解耦，开发者仅需按照标准接口规范设计新型加速核（如注意力机制），即可将 Transformer 等模型集成到框架中部署。

在算法层面，框架的核心算法——延迟平衡、模型切分与映射算法——均独立于具体的硬件和网络结构，具备良好的通用性。延迟平衡方法仅需各层的计算特征与资源需求信息即可实现流水线均衡；模型切分策略以层为粒度，通过贪心合并与分支对齐策略，适用于层次化堆叠的神经网络；集群感知映射算法以加速核连接图为输入，其目标函数与约束条件仅涉及资源消耗与通信开销，不依赖于特定模型的计算语义。因此，这些算法为支持 Transformer 等复杂网络提供了基础保障。

然而，将 TH-FCAD 高效应用于 BERT, Llama 等超大模型仍面临若干关键挑战，这也构成了我们下一步明确的研究方向：

1）新型算子加速核库的构建。针对 Transformer 的核心算子（如多头注意力机制和 Softmax 等非线性运算），现有基于 HLS4ML 的算子库缺乏高效支持。未来将设计专用加速架构，探索 DataFlow 与 PEA 架构相融合的方案，实现高吞吐、低延迟的注意力计算。

2）更细粒度的模型切分策略。当前“按层切分”方法适用于计算量均衡的卷积神经网络，但 Transformer 模型计算高度密集（如 BERT-base 单次推理超 48 GOP），粗粒度切分易导致加速核过大或流水线气泡。未来将探索层内矩阵分块计算，将大型矩阵乘法拆分为子块并跨 FPGA 并行执行，同时设计高效的数据聚合机制以协调各分块间的中间结果同步。

3）存储管理策略。BERT 等模型参数量远超片上 BRAM 容量，长序列推理中 KV Cache 等中间结果存储需求激增。目前仅使用片上存储的策略不再可行，需研究分层存储策略，结合片上缓存与高效片外存储访问，并设计数据预取单元以隐藏访存延迟。此外，还需支持动态 KV Cache 的灵活管理，以适应可变长序列推理需求。

综上所述，TH-FCAD 框架在设计层面已为支持更大规模模型奠定了软硬件基础，但要充分发挥其性能优势，仍需在算子加速核、细粒度切分与存储管理等方面开展深入研究，这也正是我们后续工作的重点方向。

5 总　　结

本文针对神经网络日益增长的计算需求与现有计算架构的能效瓶颈，提出一种基于 FPGA 集群的高效加速方案。通过设计新型的 2 级互连 FPGA 集群架构、延迟平衡优化与模型切分方法，以及集群感知的加速核映射算法与自动化集群构建工具，本研究成功实现了神经网络在 FPGA 集群上的高性能计算。实验结果表明，所提出的方案在 ResNet-8, SqueezeNet,

ResNet-50 模型上实现了显著的性能提升, 优于 CPU, GPU 以及其他多 FPGA 实现。

本文所提出的 TH-FCAD 框架不仅实现了在 FPGA 集群上对神经网络的高效自动化部署, 更从体系结构层面构建了一套可扩展、可复用的方法论体系。其核心贡献在于: 通过 Shell/Role 架构实现通信与计算的解耦, 使得延迟平衡、模型切分与映射等算法具备硬件无关性; 通过 2 级互连与集群感知映射策略, 兼顾了通信效率与资源利用率; 通过自动化工具链, 将复杂的多 FPGA 部署流程封装为端到端的黑盒操作。上述设计使得 TH-FCAD 不仅适用于当前的 TH-Express 集群与卷积神经网络模型, 也可扩展支持以太网等通用互连, 并为实现 Transformer 等更大规模模型的自动化部署奠定了基础。因此, TH-FCAD 不仅在特定任务上具有性能优势, 还提供了一种面向 FPGA 集群的通用化、自动化部署范式, 为未来异构计算集群上的大规模智能模型部署提供了可借鉴的系统性解决方案。

作者贡献声明: 赵言亢和黎渊负责整体实验方案的设计与实现、论文整体架构的设计和撰写; 齐星云负责指导方法设计; 陆亿行、刘谱光、王强负责部分实验的设计与实施; 谢旻、彭林负责系统工程实现与测试验证; 赖明澈提出研究思路与整体方案指导, 并负责论文审阅与修改。赵言亢和黎渊为共同第一作者。

参 考 文 献

[1] Li Guoqing, Zhang Meng, Li Jiaojie, et al. Efficient densely connected convolutional neural networks[J]. Pattern Recognition, 2021, 109: 107610

[2] Li Zewen, Liu Fan, Yang Wenjie, et al. A survey of convolutional neural networks: Analysis, applications, and prospects[J]. IEEE Transactions on Neural Networks and Learning Systems, 2022, 33(12): 6999−7019

[3] Nechi A, Groth L, Mulhem S, et al. FPGA-based deep learning inference accelerators: Where are we standing?[J]. ACM Transactions on Reconfigurable Technology and Systems, 2023, 16(4): 60

[4] Hill M, Reddi V. Accelerator-level parallelism[J]. Communications of the ACM, 2021, 64(12): 36−38

[5] Cong J, Fang Zhenman, Lo M, et al. Understanding performance differences of FPGAs and GPUs[C]//Proc of the 26th Annual Int Symp on Field-Programmable Custom Computing Machines (FCCM). Piscataway, NJ: IEEE, 2018: 93−96

[6] Amin R, Obermaisser R. Towards resource efficient and low latency CNN accelerator for FPGAs: Review and evaluation[C]//Proc of the 3rd Int Conf on Embedded Systems and Artificial Intelligence (ESAI). Piscataway, NJ: IEEE, 2024: 1−10

[7] Mohaidat T, Khalil K. A survey on neural network hardware accelerators[J]. IEEE Transactions on Artificial Intelligence, 2024, 5(8): 3801−3822

[8] Curzel S, Ferrandi F, Fiorin L, et al. A survey on design methodologies for accelerating deep learning on heterogeneous architectures[J]. arXiv preprint, arXiv: 2311.17815, 2025

[9] Yuan Tian, Liu Weiqiang, Han Jie, et al. High performance CNN accelerators based on hardware and algorithm co-optimization[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2021, 68(1): 250−263

[10] Zeng Shulin, Liu Jun, Dai Guohao, et al. FlightLLM: Efficient large language model inference with a complete mapping flow on FPGAs[C]//Proc of the 32nd ACM/SIGDA Int Symp on Field Programmable Gate Arrays (FPGA'24). New York: ACM, 2024: 223−234

[11] Liu Jun, Zeng Shulin, Ding Li, et al. FlightVGM: Efficient video generation model inference with online sparsification and hybrid precision on FPGAs[C]//Proc of the 33rd ACM/SIGDA Int Symp on Field Programmable Gate Arrays (FPGA'25). New York: ACM, 2025: 2−13

[12] Zhang Xiaofan, Ye Hanchen, Wang Junsong, et al. DNNExplorer: A framework for modeling and exploring a novel paradigm of FPGA-based DNN accelerator[C]//Proc of the 2020 IEEE/ACM Int Conf on Computer Aided Design (ICCAD). Piscataway, NJ: IEEE, 2020: 1−9

[13] Wei Xuechao, Liang Yun, Li Xiuhong, et al. TGPA: Tile-grained pipeline architecture for low latency CNN inference[C]//Proc of the 2018 IEEE/ACM Int Conf on Computer-Aided Design (ICCAD). Piscataway, NJ: IEEE, 2018: 1−8

[14] Ye Hanchen, Jun H, Chen Deming. HIDA: A hierarchical dataflow compiler for high-level synthesis[C]//Proc of the 29th ACM Int Conf on Architectural Support for Programming Languages and Operating Systems (ASPLOS '24). New York: ACM, 2024: 215−230

[15] Zhang Chen, Wu Di, Sun Jiayu, et al. Energy-efficient CNN implementation on a deeply pipelined FPGA cluster[C]//Proc of the 2016 Int Symp on Low Power Electronics and Design (ISLPED). New York: ACM, 2016: 326−331

[16] Zhang Wentai, Zhang Jiaxi, Shen Minghua, et al. An efficient mapping approach to large-scale DNNs on multi-FPGA architectures[C]//Proc of the 2019 Design, Automation & Test in Europe Conf & Exhibition (DATE). Piscataway, NJ: IEEE, 2019: 1241−1244

[17] Jiang Weiwen, Sha E, Zhang Xinyi, et al. Achieving super-linear speedup across multi-FPGA for real-time DNN inference[J]. ACM Transactions on Embedded Computing Systems, 2019, 18(5s): 67

[18] Boutros A, Betz V. FPGA Architecture: Principles and progression[J]. IEEE Circuits and Systems Magazine, 2021, 21(2): 4−29

[19] Mittal S. A survey on modeling and improving reliability of DNN algorithms and accelerators[J]. Journal of Systems Architecture, 2020, 104(C): 101689

[20] Schulte J, Ramhorst B, Sun Chang, et al. HLS4ML: A flexible, open source platform for deep learning acceleration on reconfigurable

hardware[J]. ACM Transactions on Reconfigurable Technology and Systems, 2026, 19(2): 19

[21] Umuroglu Y, Fraser N, Gambardella G, et al. FINN: A framework for fast, scalable binarized neural network inference[C]//Proc of the 2017 ACM/SIGDA Int Symp on Field-Programmable Gate Arrays (FPGA'17). New York: ACM, 2017: 65−74

[22] Knapheide J, Stabernack B, Kuhnke M. A high throughput MobileNetV2 FPGA implementation based on a flexible architecture for depthwise separable convolution[C]//Proc of the 30th Int Conf on Field-Programmable Logic and Applications (FPL). Piscataway, NJ: IEEE, 2020: 277−283

[23] Wu Xiao, Ma Yufei, Wang Meiqi, et al. A flexible and efficient FPGA accelerator for various large-scale and lightweight CNNs[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2022, 69(3): 1185−1198

[24] Yu Yunxuan, Zhao Tiandong, Wang Kun, et al. Light-OPU: An FPGA-based overlay processor for lightweight convolutional neural networks[C]//Proc of the 28th ACM/SIGDA Int Symp on Field-Programmable Gate Arrays (FPGA'20). New York: ACM, 2020: 122−132

[25] Venieris S, Bouganis C. fpgaConvNet: A framework for mapping convolutional neural networks on FPGAs[C]//Proc of the 24th IEEE Annual Int Symp on Field-Programmable Custom Computing Machines (FCCM). Piscataway, NJ: IEEE, 2016: 40−47

[26] Venieris S, Bouganis C. Latency-driven design for FPGA-based convolutional neural networks[C]//Proc of the 27th Int Conf on Field Programmable Logic and Applications (FPL). Piscataway, NJ: IEEE, 2017: 1−8

[27] Petrica L, Alonso T, Kroes M, et al. Memory-efficient dataflow inference for deep CNNs on FPGA[C]//Proc of the 2020 Int Conf on Field-Programmable Technology (ICFPT). Piscataway, NJ: IEEE, 2020: 48−55

[28] Huang Yingbing, Wan L, Ye Hanchen, et al. Invited: New solutions on LLM acceleration, optimization, and application[C]//Proc of the 61st ACM/IEEE Design Automation Conf. New York: ACM, 2024: 1−4

[29] AMD/Xilinx. Vivado design suite user guid high-level synthesis (UG902)[EB/OL]. (2021-05-04)[2025-11-28]. https://docs.amd.com/v/u/en-US/ug902-vivado-high-level-synthesis

[30] AMD/Xilinx. Xilinx Vitis AI overview[EB/OL]. (2023-09-28)[2025-11-28]. https://docs.amd.com/r/en-US/ug1414-vitis-ai/Vitis-AI-Overview

[31] Blott M, Preußer T, Fraser N, et al. FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks[J]. ACM Transactions on Reconfigurable Technology and Systems, 2018, 11(3): 16

[32] Chung E, Fowers J, Ovtcharov K, et al. Serving DNNs in real time at datacenter scale with project brainwave[J]. IEEE Micro, 2018, 38(2): 8−20

[33] Tarafdar N, Eskandari N, Sharma V, et al. Galapagos: A full stack approach to FPGA integration in the cloud[J]. IEEE Micro, 2018, 38(6): 18−24

[34] Tarafdar N, Di G, Harris P, et al. AIgean: An open framework for deploying machine learning on heterogeneous clusters[J]. ACM Transactions on Reconfigurable Technology and Systems, 2022, 15(3): 23

[35] Alonso T, Petrica L, Ruiz M, et al. Elastic-DF: Scaling performance of DNN inference in FPGA clouds through automatic partitioning[J]. ACM Transactions on Reconfigurable Technology and Systems, 2022, 15(2): 15

[36] Gan Xinbiao, Tang Qian, Xiong Feng, et al. TianheStar: Orchestrating sssp applications on Tianhe supercomputer[C]//Proc of the 24th IEEE Int Symp on Cluster, Cloud and Internet Computing (CCGrid). Piscataway, NJ: IEEE, 2024: 534−542

[37] Wang Ruibo, Lu Kai, Chen Juan, et al. Brief introduction of TianHe exascale prototype system[J]. Tsinghua Science and Technology, 2021, 26(3): 361−369

[38] Gan Xinbiao, Zhang Yiming, Zeng Ruigeng, et al. XTree: Traversal-based partitioning for extreme-scale graph processing on supercomputers[C]//Proc of the 38th IEEE Int Conf on Data Engineering (ICDE). Piscataway, NJ: IEEE, 2022: 2046−2059

[39] Pang Zhengbin, Xie Min, Zhang Jun, et al. The TH express high performance interconnect networks[J]. Frontiers of Computer Science, 2014, 8(3): 357−366

[40] Liao Xiangke, Pang Zhengbin, Wang Kefei, et al. High performance interconnect network for tianhe system[J]. Journal of Computer Science and Technology, 2015, 30(2): 259−272

[41] Caulfield A, Chung E, Putnam A, et al. A cloud-scale acceleration architecture[C]//Proc of the 49th Annual IEEE/ACM Int Symp on Microarchitecture (MICRO). Piscataway, NJ: IEEE, 2016: 1−13

[42] Çatalyürek Ü, Devine K, Faraj M, et al. More recent advances in (Hyper) graph partitioning[J]. ACM Computing Surveys, 2023, 55(12): 253

[43] Banbury C, Reddi V, Torelli P, et al. MLPerf tiny benchmark[J]. arXiv preprint, arXiv: 2106.07597, 2021

[44] Iandola F, Han Song, Moskewicz M, et al. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size[J]. arXiv preprint, arXiv: 1602.07360, 2016

[45] He Kaiming, Zhang Xiangyu, Ren Shaoqing, et al. Deep residual learning for image recognition[C]//Proc of the 2016 IEEE Conf on Computer Vision and Pattern Recognition (CVPR). Piscataway, NJ: IEEE, 2016: 770−778

[46] Mageiropoulos E, Chrysos N, Dimou N, et al. Using HLS4ML to map convolutional neural networks on interconnected FPGA devices[C]//Proc of the 29th IEEE Annual Int Symp on Field-Programmable Custom Computing Machines (FCCM). Piscataway, NJ: IEEE, 2021: 277−277

[47] Ding Yuhao, Wu Jiajun, Gao Yizhao, et al. Model-platform optimized deep neural network accelerator generation through mixed-integer geometric programming[C]//Proc of the 31st IEEE Int Symp on Field-Programmable Custom Computing Machines (FCCM). Piscataway, NJ: IEEE, 2023: 83−93

[48] Ibrahim M, Zhao Zhipeng, Hall M, et al. Extending data flow architectures for convolutional neural networks to multiple FPGAs[C]//Proc of the 2023 Int Conf on Field Programmable

Technology (ICFPT). Piscataway, NJ: IEEE, 2023: 132−141

[49] Hall M, Betz V. From tensorflow graphs to LUTs and wires: Automated sparse and physically aware CNN hardware generation[C]// Proc of the 2020 Int Conf on Field-Programmable Technology (ICFPT). Piscataway, NJ: IEEE, 2020: 56−65

[50] Liang Shiyan, Lin Jingui, Liu Dongwei, et al. RTMF: Routing based on TDM for multi-FPGA system[J]. ACM Transactions on Design Automation of Electronic Systems, 2025, 30(5): 81



Zhao Yankang, born in 2000. PhD candidate. His main research interests include hardware acceleration and computer architecture.

赵言亢, 2000 年生。博士研究生。主要研究方向为硬件加速、计算机体系结构。



Li Yuan, born in 1984. PhD, associate professor. His main research interests include high performance computer architecture, high speed interconnection network, and artificial intelligence.

黎　渊, 1984 年生。博士，副研究员。主要研究方向为高性能计算机体系结构、高速互连网络、人工智能。



Qi Xingyun, born in 1979. PhD, associate professor. His main research interests include high performance computer architecture and high speed interconnection network.

齐星云, 1979 生。博士，副研究员。主要研究方向为高性能计算机体系结构、高速互连网络。



Lu Yihang, born in 1999. Master. His main research interests include computer architecture and artificial intelligence.

陆亿行, 1999 年生。硕士。主要研究方向为计算机体系结构、人工智能。



Liu Puguang, born in 1995. PhD, assistant professor. His main research interests include algorithm accelerator, reconfigurable computing, and high-performance network.

刘谱光, 1995 年生。博士，助理研究员。主要研究方向为算法加速、可重构计算、高性能网络。



Wang Qiang, born in 1989. PhD, assistant professor. His main research interests include computer architecture, high speed interconnection network, and artificial intelligence.

王　强, 1989 年生。博士，助理研究员。主要研究方向为体系结构、高速互连网络、人工智能。



Xie Min, born in 1972. PhD, professor. His main research interest includes parallel and distributed computing.

谢　旻, 1972 年生。博士，研究员。主要研究方向为并行与分布式计算。



Peng Lin, born in 1979. PhD, associate professor. His main research interest includes high performance computing.

彭　林, 1979 年生。博士，副研究员。主要研究方向为高性能计算。



Lai Mingche, born in 1982. PhD, professor. His main research interests include computer architecture, high performance interconnection, and VLSI design.

赖明澈, 1982 年生。博士，研究员。主要研究方向为体系结构、高性能互连、超大规模集成电路设计。