计 算 机 研 究 与 发 展
Journal of Computer Research and Development
https://crad.ict.ac.cn
62(6): 1363−1379, 2025

大规模异构一致性融合计算系统的性能建模与优化

李仁刚[1] 唐轶男[2] 郭振华[2] 王 丽[2] 宗 瓒[1] 杨广文[1]
[1](清华大学计算机科学与技术系 北京 100084)
[2](浪潮电子信息产业股份有限公司 济南 250101)
(lirg@ieisystem.com)

Performance Modeling and Optimization for Large-Scale Heterogeneous Consistency Integrated Computing System

Li Rengang[1], Tang Yinan[2], Guo Zhenhua[2], Wang Li[2], Zong Zan[1], and Yang Guangwen[1]
[1](*Department of Computer Science and Technology*, *Tsinghua University*, *Beijing* 100084)
[2](*IEIT SYSTEMS Co.*, *Ltd.*, *Jinan* 250101)

Abstract With the widespread adoption and development of large-scale artificial intelligence applications, the demand for computing power in artificial intelligence from both industry and academia is increasing. Heterogeneous consistency integrated computing systems, which combine heterogeneous computing technology with cache consistency technology, are gradually becoming an important solution for building intelligent computing centers in the future. However, due to the immaturity of heterogeneous computing and consistency interconnect technologies, it is hard for existing research to model the performance of such systems, making it difficult for researchers to evaluate construction schemes, predict computing performance, and assess system optimization methods at a low cost. We propose HCSim, a performance modeling tool for heterogeneous consistency integrated computing systems, addressing challenges in modeling system topology and inaccuracies in workload modeling within consistency systems. HCSim provides researchers with a flexible, low-cost, and efficient modeling and simulation tool for evaluating interconnect topologies and AI computing tasks. Using HCSim, we model a heterogeneous consistency integrated computing system with thousands of accelerators, and simulate the data-parallel distributed training task of the LLAMA2-13B large language model (LLM) on this system, exploring the impact of variables such as heterogeneous computing power distribution, bandwidth, latency, and task scale on system performance and task execution efficiency. Furthermore, we also design optimization strategies for the communication issues in heterogeneous consistency integrated computing systems and validate the effectiveness of these strategies using HCSim. The simulation results show that HCSim not only meets the performance modeling needs of heterogeneous consistency integrated computing systems, but can also be applied to evaluate and verify optimization strategies for such systems.

Key words artificial intelligence; high performance computing; modeling and simulation; distributed computing; cache consistency; heterogeneous computing

摘 要 随着大规模人工智能应用的普及与发展,工业界和学术界对于人工智能算力的需求逐渐提升,结合了异构计算技术与缓存一致性技术的异构一致性融合计算系统逐渐成为未来构建智算中心的重要解决方案.然而,由于异构计算和一致性互连技术尚不成熟,现有工作难以实现对该系统进行性能建模,导致研究者无法以低成本完成异构一致性融合计算系统的建设方案评估、计算性能预测以及系统优化方法

收稿日期: 2025-03-01; **修回日期**: 2025-04-08
通信作者: 唐轶男(tangyinan@ieisystem.com)

评测等工作. 提出了一种面向异构一致性融合计算系统的性能建模工具 HCSim, 解决了现有建模仿真研究中对该系统拓扑架构建模困难、对一致性系统中工作负载建模不准确等问题, 为研究者提供了一个可灵活建模、评估互连拓扑与 AI 计算任务的低成本、高效建模仿真工具. 利用 HCSim, 建模了千卡互连的异构一致性融合计算系统, 并在该系统上模拟了 LLAMA2-13B 大语言模型 (large language model, LLM) 的数据并行分布式训练任务, 探究了异构算力分布、带宽、时延和任务规模等变量对系统性能与任务执行效率的影响. 进一步地, 针对异构一致性融合计算系统的通信问题, 设计了相应的优化方案, 并利用 HCSim 进行了效果验证. 仿真结果说明 HCSim 不仅能够满足异构一致性融合计算系统的性能建模需求, 同时也可以被应用于评估、验证异构一致性融合计算系统的优化方案.

关键词 人工智能; 高性能计算; 建模仿真; 分布式计算; 缓存一致性; 异构计算

中图法分类号 TP311.13

DOI: 10.7544/issn1000-1239.202550120 **CSTR:** 32373.14.issn1000-1239.202550120

随着人工智能(AI)技术的快速发展, 大语言模型(large language model, LLM)[1]、多模态大模型[2]等先进 AI 模型不断涌现, 工业界和学术界对 AI 计算的算力需求也在持续攀升. 具体而言, 一方面, 当前 AI 的训练与推理不仅对算力设备的数量提出了更高要求[3], 同时对算力性能的特性也有不同需求[4]. 因此, 基于异构计算的计算系统已成为当今业界构建智算集群的主要方式. 另一方面, 内存限制成为了制约 AI 计算的重要问题[5]. 对于训练过程, 现有计算架构已无法支撑该过程中产生的激活、优化器状态等内存数据, 因此才产生了如 Zero[6]、重计算[7]、数据并行训练[8]等折中的内存瓶颈缓解技术. 对于推理过程, KVcache[9]、MoE[10]等技术也对现有内存容量带来了巨大挑战. 缓存一致性技术被认为有望解决计算卡内存受限的问题[11], 诸如英伟达等厂商均提出了高效的缓存一致性互连技术[12-13], 形成紧耦合的融合计算系统[14], 以提升算力资源在并行 AI 计算中的协同执行效率. CXL 协议[15]的提出也论证了一致性互连技术对内存池化、管理、调度等方面的优势. 因此, 在未来, 结合异构计算与缓存一致性互连技术的异构一致性融合计算系统, 将成为 AI 算力基础设施的重要发展方向.

异构一致性融合计算系统结合类型多样的异构算力, 如 CPU、GPU、FPGA 等, 不仅能够充分发挥不同计算架构的优势, 还能够通过高效的内存管理和缓存一致性协议协调不同异构算力之间的数据访问与处理. 然而, 由于异构计算和一致性互连等关键技术尚未完全成熟, 异构一致性融合计算系统的性能预测与评估仍然是当前亟待解决的问题. 例如, 如何在建设异构一致性融合计算系统之前以低成本快速地估算其性能, 如何评测 LLM 训练任务在该系统中的执行效率, 以及如何发现并优化计算系统中的性能瓶颈, 都是异构一致性融合计算系统研究中待解决的核心挑战.

但是, 在实际应用中, 该计算系统的性能涉及算力的分布、算力与内存的互连架构、任务负载的部署方式、互连拓扑的时延带宽参数等多个复杂变量和指标, 因此在实际硬件系统中进行测评的难度和成本往往较高, 难以实现. 为此, 使用建模与仿真方法对异构一致性融合计算系统进行性能评估, 成为一种较为理想的解决方案.

然而, 现有的建模与仿真研究大多集中于同构计算系统和非一致性计算系统, 难以有效地对异构计算、一致性互连等关键技术进行建模. 目前, 建模异构一致性融合计算系统时, 主要面临 2 个挑战:

1)异构一致性融合计算系统的拓扑架构建模具有较大难度. 模型中不仅需要构建异构算力、内存设备等各类器部件之间的互连拓扑, 还需考虑异构算力之间的计算性能差异. 以往的建模与仿真工作往往难以同时满足这些需求.

2)引入缓存一致性协议后, 计算负载的执行方式会与传统的非一致性系统有所不同. 尽管一致性系统解决了算力主存空间的限制, 但算力对远程内存的直接访问开销成为需要特别考虑的因素[16], 这也使得以往的工作难以进行合理建模.

为了解决以上问题, 本文研究提出了一种面向异构一致性融合计算系统的性能建模工具 HCSim (heterogeneous coherent simulator). HCSim 集成了成熟的分布式系统仿真器 SimGrid[17-18], 充分利用其互连拓扑描述能力以及 SimGrid-S4U[19]的高效网络仿真能力, 从而有效解决了异构一致性融合计算系统的拓扑架构建模难题. 此外, HCSim 针对一致性系统的

工作特性，设计了结合内存访问特性的负载建模与负载运行模拟方法，使用户能够模拟一致性系统下的工作负载执行方式，并生成可观测的执行轨迹. 进一步地，基于所设计的 HCSim，本文研究对异构一致性融合计算系统与 LLM 训练任务进行了建模，探究了多种变量参数对该系统性能的影响，并提出优化方案以缓解该系统中的通信瓶颈，提高计算系统的整体性能. 本文的贡献有 3 点：

1）提出了面向异构一致性融合计算系统的性能建模工具 HCSim，解决了拓扑架构建模困难以及一致性计算系统负载执行方式存在差异的问题，为异构一致性融合计算系统的性能建模、评估和优化提供了高效的仿真分析工具.

2）基于 HCSim 构建了在异构一致性融合计算系统下执行 LLM 训练负载的仿真，并探究了异构算力分布、并行计算规模、时延、带宽等参数对计算系统性能的影响.

3）面向异构一致性融合计算系统中的通信性能优化问题，提出了一致性下的 ring-allreduce 参数同步方法，并使用 HCSim 验证了该优化方式对计算系统性能的提升.

本文研究旨在面向新型异构一致性融合计算系统，提出高效、可用的建模仿真分析工具，为工业界和学术界提供低成本的新型计算系统性能与优化方案的评估手段，从而助力新一代 AI 基础设施与 AI 应用的创新与发展.

1 相关工作

1.1 高精度的计算、内存、网络仿真器

计算系统通常涉及处理器计算、内存访存和网络通信等过程[20]，以往已有许多成熟的研究能够对上述 3 种过程进行高精度的建模与仿真. 对于计算过程的建模与仿真，经典的体系结构仿真器包括高度可配置的开源微架构仿真器 Gem5[21]，兼具指令级和全系统仿真的操作系统开发与嵌入式系统仿真器 QEMU[22]，以及专门用于 GPGPU 计算和并行程序性能分析与优化的 GPGPU-Sim[23] 等；对于内存访存过程的建模与仿真，较为成熟的仿真器包括专注于 DRAM 访问时序和性能分析的 DRAMSim3[24]，以及支持较新内存标准（如 DDR4、HBM）的仿真器 Ramulator[25] 等；在网络仿真器方面，较为广泛使用的是离散事件网络仿真器 NS-3[26]，还有支持无线网络、物联网、车联网和数据中心网络的模块化仿真器 OMNeT++[27] 等.

然而，以上仿真器通常面临计算复杂度高和仿真耗时较大的问题. 例如，使用 GPGPU-Sim 仿真单个 GPU 计算任务可能需要长达几天的时间[28]，而 NS-3 仿真 64 个节点执行 1MB 的 all-reduce 操作可能需要 20 min 以上[29]. 因此，直接使用这些成熟的高精度仿真器，往往难以在有效时间内对规模较大的计算系统进行仿真与分析，无法满足建模异构一致性融合计算系统的需求.

1.2 面向分布式 AI 计算系统的性能建模

随着分布式 AI 计算技术的飞速发展，业界和学界涌现了一系列面向分布式 AI 计算系统与 AI 计算任务的性能建模仿真研究，表 1 中展示了近几年提出的相关工作.

Table 1 Performance Modeling Work for Distributed AI Computing Systems

表 1 面向分布式 AI 计算系统的性能建模工作

工作名称	工作简介
AMPeD[30]	针对 Transformer 神经网络架构的分布式训练任务进行数学建模的研究.
Calculon[31]	针对 LLM 的分布式训练过程进行数学建模的工作.
Paleo[32]	早期使用数学模型与有向无环图（DAG）模型建模数据并行、模型并行等 AI 分布式训练任务的研究.
FlexFlow[33]	一个开源的深度学习编译器和运行时系统，其中提供了一个基于图模型的模块，用于建模与优化分布式 AI 计算任务.
Daydream[34]	使用细粒度的基于图的模型描述分布式训练，主要针对基于 all-reduce 的分布式并行任务进行建模与仿真.
Dpro replayer[35]	基于图模型预测和优化数据并行分布式训练的性能.
DistSim[36]	用于预测每个参与分布式训练的算力节点的详细工作时间线.
DistIR[37]	一种基于 MLIR[38] 的中间表示，用于以伪代码的形式模拟分布式训练的执行过程.
Proteus[39]	使用策略树描述并行策略，并依赖执行图来建模分布式训练. 该研究考虑了分布式训练的运行时特征.
TAG[40]	使用图模型建模并预测分布式训练任务的耗时，同时基于图神经网络搜索并优化分布式训练的执行策略.
SMSG[41]	一种无需进行实际性能采集分析（profiling）的分布式训练数学建模方法，仅需输入 AI 加速器的实际计算性能和系统通信能力，即可对分布式系统中的 AI 训练任务进行建模.
Astra-sim[29,42]	一个帮助研究人员理解并探索分布式训练中多种软硬件协同设计空间的模拟器.

然而，这些已有的研究对于异构计算与一致性互连技术的支持能力较为有限. 具体而言，现有工作面临的主要问题包括以下 4 点：

1）通常假设所有算力的计算能力一致，难以模拟异构算力差异对计算系统性能的影响.

2）依赖于在特定硬件上运行 AI 模型训练的前几个训练步骤，并通过性能分析工具获取执行轨迹，导致应用成本较高，无法适用于尚未实际部署的计算系统.

3)对于计算系统中互连拓扑建模的灵活性较弱，无法描述异构一致性融合计算系统中复杂的互连架构.

4)缺乏对一致性建模仿真的支持，无法建模一致性条件下 AI 计算任务的工作负载与执行方式.

由此可见，现有的建模仿真研究难以解决上述拓扑架构建模困难和引入一致性后计算负载建模困难的两大挑战，难以实现对异构一致性融合计算系统的性能建模，也无法在缺乏实际软硬件的情况下评估系统优化方案的效果. 这些正是本文研究所提出的 HCSim 所要解决的问题.

1.3 相对精度的系统性能建模

随着计算系统的变量、规模和复杂性逐渐增加，保证性能建模与仿真工作高精度的技术难度不断提升，搭建实际计算系统软硬件平台对仿真器进行校准的成本也在逐步增加，这使得建模与仿真工具的设计与实现变得愈加困难. 因此，近年来，许多建模与仿真工作不再专注于提升仿真的绝对精度，而是更多地关注模拟计算系统的功能与原理，并输出用于参考的相对指标.

例如，SMSG[41] 在其研究中指出，重要的不是估计分布式训练的实际执行时间，而是对比不同策略的相对执行耗时. 在 Scale-sim[43] 仿真器中，研究人员仿真了脉冲阵列微架构[44] 的工作流程，但其仿真器仅输出了仿真中定义的周期(cycle)，并没有将仿真输出结果与真实系统进行对比. 同样地，在近期提出的 Astra-sim[29,42] 中，研究者也仅使用仿真中定义的时间对比了不同计算系统软硬件协同设计方式的差异，未探究仿真器建模的具体精度.

对于异构一致性融合计算系统的性能建模与仿真，由于异构软硬件之间通常存在兼容性问题，实际部署计算系统并对仿真进行校准需要付出巨大的工程量与成本，这与建模仿真的初衷相悖. 因此，与诸多近期的研究工作一致，本文研究提出的 HCSim 采用相对精度，旨在对比不同算力与内存的互连架构、任务负载的部署方式、互连拓扑时延带宽等多种变量对系统性能的相对影响，从而为异构一致性融合计算系统的设计与优化提供指导建议.

2 HCSim 的设计实现与工作原理

HCSim 的建设目标是对异构一致性融合计算系统的性能进行建模，解决以往建模仿真工作对该系统进行建模的挑战. 根据输入的计算系统互连架构和 AI 计算负载，HCSim 用于模拟仿真 AI 计算负载的执行流程，最终输出 AI 计算负载在计算系统中的执行耗时和运行轨迹，为研究者在异构一致性融合计算系统的设计与优化过程中提供相对指标的参考.

2.1 HCSim 的架构

为了实现上述目标，本文研究提出 HCSim 的系统架构如图 1 所示，主要包括 3 个核心层的设计：

1)平台层. 用于接收和解析用户输入的异构一致性融合计算系统的互连拓扑配置，并支持用户对互连拓扑以及拓扑中的算力、内存节点进行自定义，进而对算力节点、内存节点、拓扑结构以及节点之间的互连进行建模与仿真.

2)负载层. 接收用户定义的 AI 计算任务描述，并将计算任务转化为 HCSim 所定义的负载图，用于描述计算系统中的分布式 AI 计算任务. 随后，负载层根据转化后的负载图，驱动执行层对计算任务的执行过程进行仿真模拟，并在执行过程中不断输出细粒度的执行轨迹.

3)执行层. 在负载层的驱动下，执行层用于仿真 AI 计算负载在平台层所定义的计算系统中的执行过

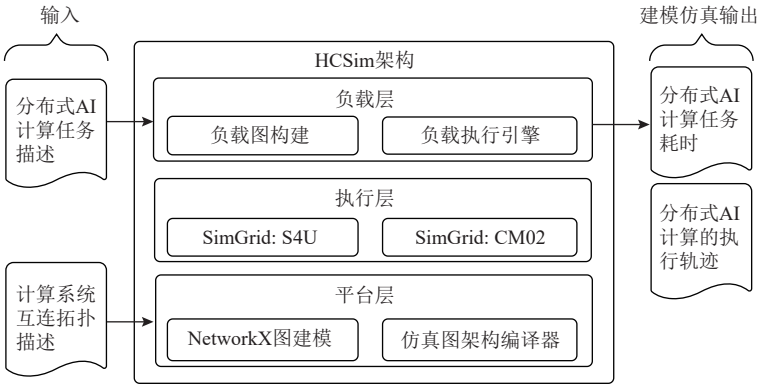


Fig. 1 System architecture of HCSim

图 1 HCSim 的系统架构

程. 该层基于 SimGrid[17] 实现，包括通信建模和计算建模 2 个部分，能够模拟算子级别的执行流程，输出 AI 计算任务的细粒度执行耗时.

HCSim 依赖于这 3 层的相互协作，实现对异构一致性融合计算系统的性能建模.

2.2 异构一致性融合计算系统的拓扑架构建模

为了满足对异构一致性融合计算系统拓扑架构进行建模的需求，HCSim 的平台层支持灵活自定义算力节点、内存节点和互连拓扑，并支持定义相关的核心参数. 平台层所定义的互连拓扑会被传递至执行层所集成的 SimGrid 仿真器[17]，从而实现计算系统互连拓扑的模型构建.

与绝大多数建模工作类似，HCSim 仿真平台采用图的方式对互连拓扑进行建模. 表 2 展示了 HCSim 在平台层支持构建的图节点与边等元素，以及它们相应的属性参数. 在 HCSim 中，所构建的图上的节点分为交换节点、异构算力节点和内存节点. 其中，交换节点主要用于描述计算系统中的数据交换节点，如 PCIE Switch, NVSwtich, L1 Switch, ToR Switch 等，其节点上没有必要的属性. 异构算力节点用于描述计算系统中的异构算力，这些节点需要定义与计算能力相关的属性信息. 对于内存节点，HCSim 支持定义内存的容量，也可以将内存设置为无穷大. 对于所构建图中用于连接节点的边，在 HCSim 中设计用边来表示节点之间的物理链路连接，因此需要定义带宽、时延等与数据传输相关的属性信息.

Table 2 Element Definitions of Topology Graph

表 2 拓扑图的元素定义

元素名称	属性名	属性含义
交换节点		无需定义属性
异构算力节点	fp16 算力/ FLOPS	fp16 计算精度下的计算性能
	fp32 算力/ FLOPS	fp32 计算精度下的计算性能
内存节点	内存容量	包含可用的内存空间有多少
边	时延	所连接节点之间的通信时延
	带宽	所连接节点之间的通信带宽

基于平台层的设计，图 2 展示了 HCSim 可以构建的异构一致性融合计算系统拓扑的抽象表示. 在一致性互连技术的帮助下，对于任意异构算力(图 2 中的 XPU)，不仅可以直接以高带宽低时延访问本地主存，还可以通过一致性互连网络直接访问远端其他 XPU 的内存或扩展内存，实现缓存一致性. 在图 2 中，HCSim 可以自由修改计算系统和一致性互连网络的互连拓扑，也可以自由定义每个异构算力节点的计算能力、每个内存节点的存储容量，从而实现对异构一致性融合计算系统拓扑架构的灵活建模. 另外，值得注意的是，HCSim 对异构算力的定义要求较低，这使 HCSim 不仅可以建模常用的异构算力，而且可以对诸如 FPGA, ASIC 等定制化算力进行性能建模. 研究者只需通过测试的方式获得异构算力的计算能力、内存容量等信息，便可以利用 HCSim 对不同的异构算力开展建模仿真.

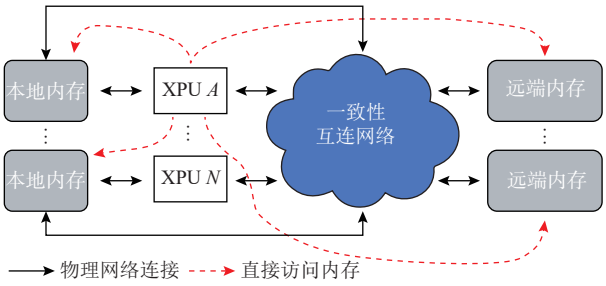


Fig. 2 Workload graph comparison

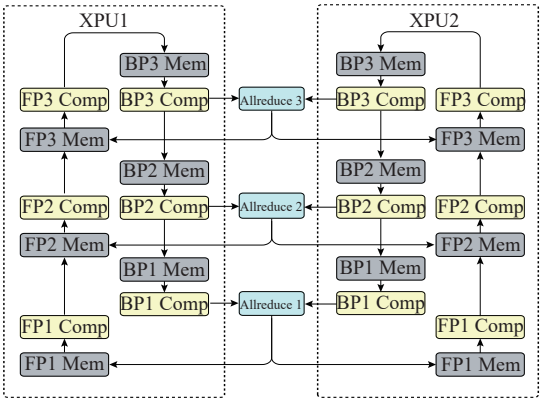
图 2 工作负载图对比

2.3 一致性系统中的 AI 计算负载建模

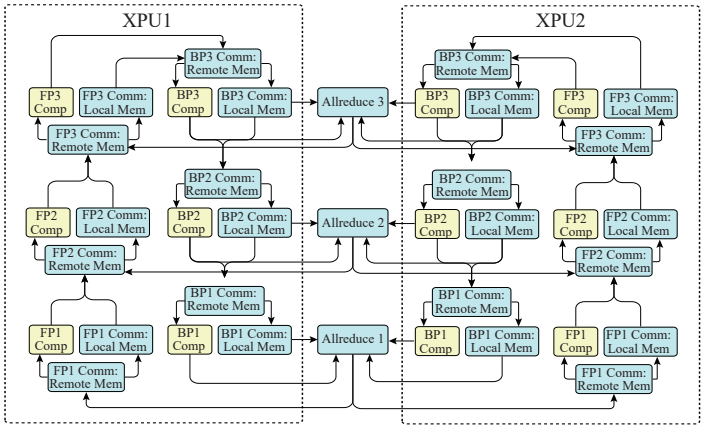
在异构一致性融合计算系统中，由于 XPU 可以直接对非本地内存进行访存操作，因此 AI 计算负载的工作方式与传统非一致性计算系统存在差异，这也导致以往的建模仿真工作难以直接应用.

以往绝大多数建模仿真工作并未考虑内存对计算负载的影响. 即便较新的 Astra-sim 2.0[29] 提出了面向分离式内存的负载模型，它仍然难以有效描述异构一致性融合计算系统下的负载计算流程. 图 3(a)展示了基于 Astra-sim 2.0 的负载模型构建的计算任务负载图，其中描述了 2 个异构算力采用数据并行训练带有 3 个神经层的神经网络. 在图 3(a)中，每个顶点代表一个执行过程，边代表执行过程之间的依赖关系，即对于图中的任一个顶点，只有所有指向该顶点的顶点完成后，该顶点才能启动执行. 图 3(a)中灰色顶点代表访存的通信过程，黄色顶点代表计算过程，蓝色顶点代表集合通信过程，FP 代表前向传播，BP 代表反向传播，FP1 代表第一个神经层的前向传播过程，以此类推. 可以看出，尽管 Astra-sim 2.0 考虑了访存过程对 AI 计算任务的影响，但其建模中仍然仅模拟了将远端内存搬运到本地显存，再进行计算的过程，这与一致性系统的实际情况不符.

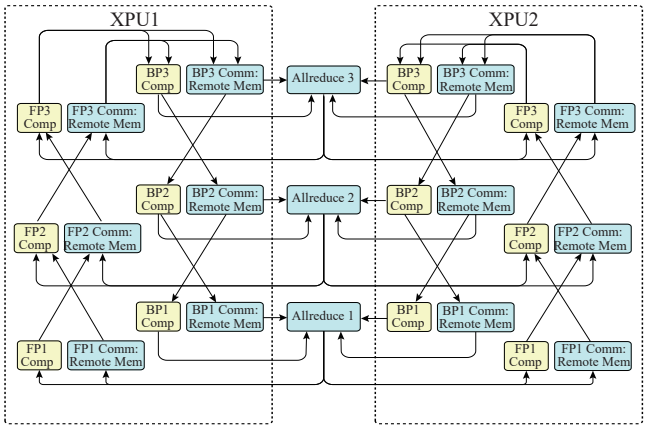
图 3(b)和图 3(c)分别展示了 HCSim 的负载层为上述相同 AI 计算任务在非一致性系统和一致性系统中构建的负载图，其中黄色顶点代表计算过程，蓝色顶点代表通信过程. 在一致性系统中，由于 XPU



(a) Astra-sin 2.0的负载图



(b) HCSim的非一致性系统负载图



(c) HCSim 的一致性系统负载图

Fig. 3 Abstract representation of the heterogeneous consistency integrated computing systems topology in HCSim

图 3 HCSim 中异构一致性融合计算系统拓扑的抽象表示

可以直接访问远端内存，因此远端访存过程可以与计算过程形成流水线执行. 在这种情况下，神经层的计算耗时可能会受到异构算力计算能力的限制，也可能会受到访存速度的限制. 因此，如图 3(c)所示，HCSim 构建的负载图将计算过程与访存过程描述为并行执行，可以近似表达访存与计算的流水执行过程，进而挖掘每个神经层在一致性系统中的性能瓶颈，以实现更准确地表达 AI 计算负载在一致性系统中的执行流程. 相比之下，3(b)则展示了 HCSim 所建模的非一致性系统中异构算力先进行内存搬运，再进行主存访存与计算的过程.

表 3 展示了 HCSim 的负载图中所有顶点以及可定义的属性. 可以看出，除了需要在负载图中绑定计算顶点和通信顶点的执行位置之外，HCSim 的任务负载图与互连拓扑图几乎解耦，这使得 HCSim 的使用者能够更快捷地定义不同的计算任务负载. 此外，区别于以往建模工作中常用的有向无环图(DAG)，HCSim 采用有向有环图对 AI 计算负载进行建模，从而帮助 HCSim 更准确地模拟 AI 训练任务反复迭代循环的过程. 对于 AI 推理计算、分布式数据分析等其他计算负载，使用者也可以在 HCSim 中用同样的方式进行定义与仿真模拟.

Table 3 Element Definitions in the Workload Graph

表 3 负载图的元素定义

元素名称	属性名	属性含义
计算顶点	计算量/ FLOPs	该计算顶点所代表的计算任务的计算量
	访存量	该计算顶点所执行的任务需要的访存数据量
	执行位置	该计算顶点所执行的异构算力位置
	是否为起始顶点	该计算顶点是不是计算任务的起始顶点
通信顶点	通信量	该通信顶点的总通信量
	通信方式	该通信顶点的通信方式（例如集合通信、点对点通信）
	通信范围	根据通信顶点的通信方式，定义源节点、目的节点，或通信节点范围等
	是否为起始顶点	该通信顶点是不是计算任务的起始顶点
连接顶点的边	是否需要在第 1 次执行考虑	在计算任务第 1 次执行迭代时，不需要考虑有些边到顶点执行的依赖关系中

2.4 HCSim 的工作流程与实现方式

基于构建的互连拓扑和负载图，HCSim 可以对分布式 AI 计算进行建模与仿真. 整体的工作流程如图 4 所示，图中展示的每个具体步骤以及实现方式将在本节中逐一进行介绍.

1）互连拓扑图与负载图生成

如图 4 中的①所示，为了构建 2.2 节和 2.3 节中描述的互连拓扑图和负载图，HCSim 在负载层和平台层分别构建了负载图描述与生成模块、互连拓扑描述与生成模块，并集成了 NetworkX[45] 的接口. NetworkX 是一个非常实用的 Python 库，专门用于创建、操作和研究图与网络. 该库提供了丰富的功能，使得用户能够方便地构建有向图或无向图，添加或删除节点和边，或为这些节点和边赋予属性. 此外，NetworkX 还支持多种图的遍历算法，如深度优先搜索和广度优先搜索，这使得在复杂的网络结构中查找路径、修改路由等方面变得更加简便. 另一个优势是，NetworkX 提供了与 Matplotlib 集成的绘图工具，使得使用者能够将构建的网络拓扑可视化，从而直观地验证所构建的拓扑是否符合需求. 借助 NetworkX，HCSim 的使用者只需按照表 2 和表 3 的要求定义图与属性，即可将所生成的互连拓扑图、负载图作为 HCSim 建模与仿真的输入.

2）NetworkX-SimGrid 拓扑编译

NetworkX 本身只能进行图的构建与分析，并不具备仿真能力. 因此，如图 4 中的②所示，HCSim 在平台层进一步构建了 NetworkX-SimGrid 编译器，使得 NetworkX 构建的图模型能够被 SimGrid 加载，并通过 SimGrid-S4U[18] 模型进行仿真执行.

具体来说，NetworkX-SimGrid 编译器的功能是遍历 NetworkX 所生成的图模型中的节点和边，并将它们逐一转换为 SimGrid 支持的 XML 文件格式. 同时，编译器还会使用 NetworkX 中的 Dijkstra 算法计算所有节点之间的最短路径，并将这些最短路径作为路由信息，与图模型一同存储为 SimGrid 可读的 XML 文件. 此外，路由信息也可以由用户自行定义，从而进一步提升仿真的灵活性.

3）负载执行引擎

如图 4 中的③所示，在 HCSim 的负载层中，负载执行引擎的作用是结合所构建的负载图，驱动执行层进行仿真，从而完成对 AI 分布式计算任务的模拟执行.

具体而言，负载执行引擎采用事件驱动的方式进行仿真. 如图 5 所示，负载执行引擎首先将负载图中的所有起始顶点（表 3 中定义为起始顶点的所有顶点）放入一个执行容器中，并启动这些顶点的执行，将其送入执行层进行仿真. 如果执行顶点是计算顶点，该顶点的计算任务信息将被发送到执行层的计算模型进行仿真；如果执行顶点是通信顶点，该顶点则会被发送至执行层的网络模型进行仿真. 随后，负载层还包含一个事件触发器. 一旦执行容器中的某个顶点完成计算或通信，事件触发器会被激活，从执行容器中移除该完成的顶点，并检查该完成顶点在负载图中所指向的所有顶点是否满足所有依赖. 如

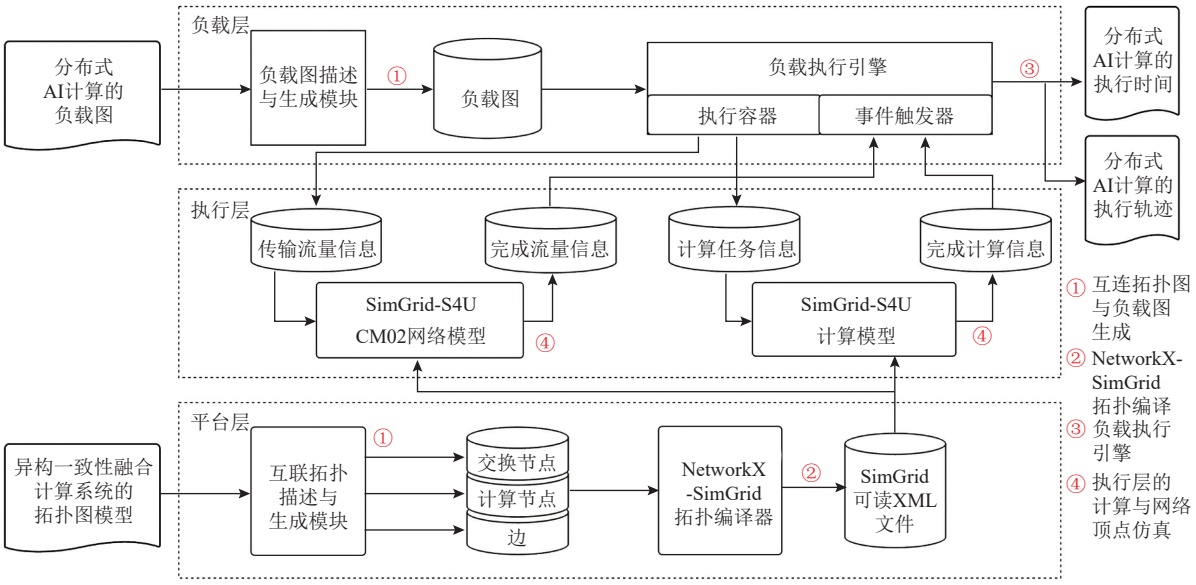


Fig. 4 Workflow and implementation method of HCSim

图 4 HCSim 的工作流程与实现方式

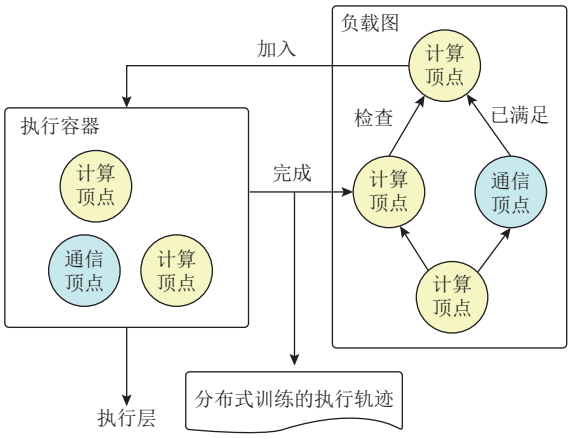


Fig. 5 Schematic diagram of load execution engine

图 5 负载执行引擎的示意图

果有满足所有依赖条件的顶点,这些顶点将被加入到执行容器中. 这样,负载图便可以在执行容器中不断循环执行,并根据仿真平台使用者定义的执行迭代次数生成相应的执行轨迹,直到所有初始顶点被重新加入容器的次数达到定义的迭代次数. 最后,负载执行引擎将收集并汇总整个模拟执行过程,并将分布式 AI 计算的执行时间与执行轨迹返回给仿真平台的使用者.

4)执行层的计算与网络顶点仿真

如图 4 中的④所示,当接收到计算顶点或通信顶点的信息时,执行层需要结合平台层的信息对这些顶点进行模拟. 对于计算顶点,HCSim 直接使用 SimGrid 中的计算模型进行耗时计算,即:

$$t_{\text{comp}} = \frac{\text{FLOPs}}{\text{FLOPS}}, \tag{1}$$

其中,FLOPs(floating point operations)是计算顶点的浮点运算次数,FLOPS(floating point operations per second)是计算顶点所分配的异构算力的计算能力. 对于通信节点,HCSim 采用 SimGrid 的 CM02 模型[46],并设置 TCP-gamma 为 0,从而获得经典的通信耗时模型:

$$t_{\text{comm}} = t_{\text{latency}} + \frac{L_{\text{data size}}}{H_{\text{bandwidth}}}. \tag{2}$$

值得注意的是,以上所有 SimGrid 模型均采用线性最大-最小求解器进行模拟,即不断根据输入到仿真器中的执行任务的执行速度和剩余工作量计算最早完成的操作,并更新求解器与其他未完成的执行任务. 基于这种仿真方式,SimGrid 可以更好地权衡仿真精度与仿真速度,从而高效地仿真分布式计算系统的执行流程.

一旦有顶点执行完毕,执行层会转发 SimGrid 输出的完成信息,并将其反馈给负载执行引擎的事件触发器. 最终,负载执行引擎根据收到的反馈信息,输出 AI 计算任务的执行时间与执行轨迹,完成对异构一致性融合计算系统的性能建模.

3 仿真与分析

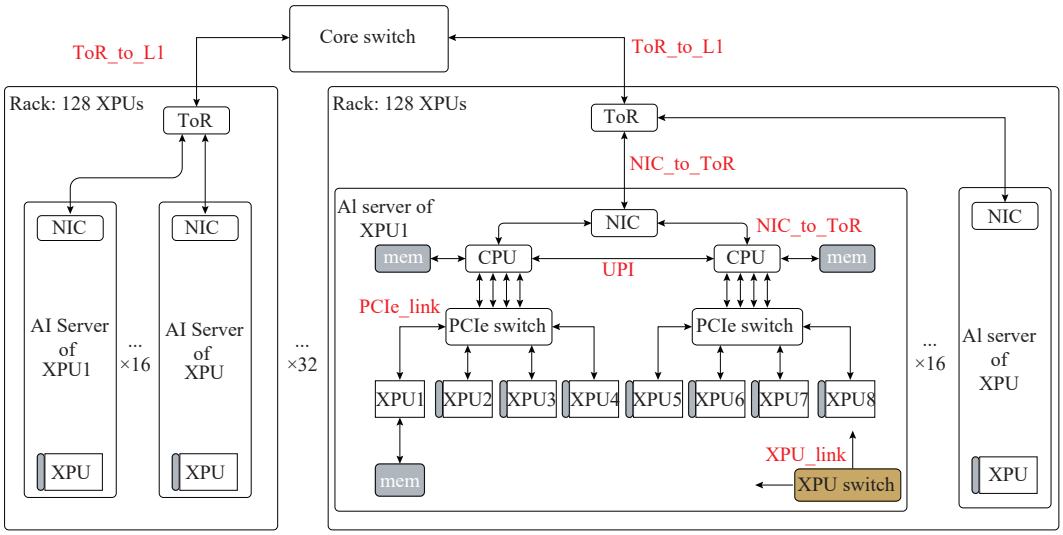
借助 HCSim 的仿真能力,本节从计算系统互连架构、异构算力分布、任务规模,以及计算系统互连拓扑的时延、带宽等参数出发,研究与分析了异构一致性融合计算系统的构建方式与性能瓶颈. 随后,针对异构一致性融合计算系统的通信瓶颈问题提出了优化方法,并使用 HCSim 仿真验证了该优化方法的效果.

3.1 仿真配置

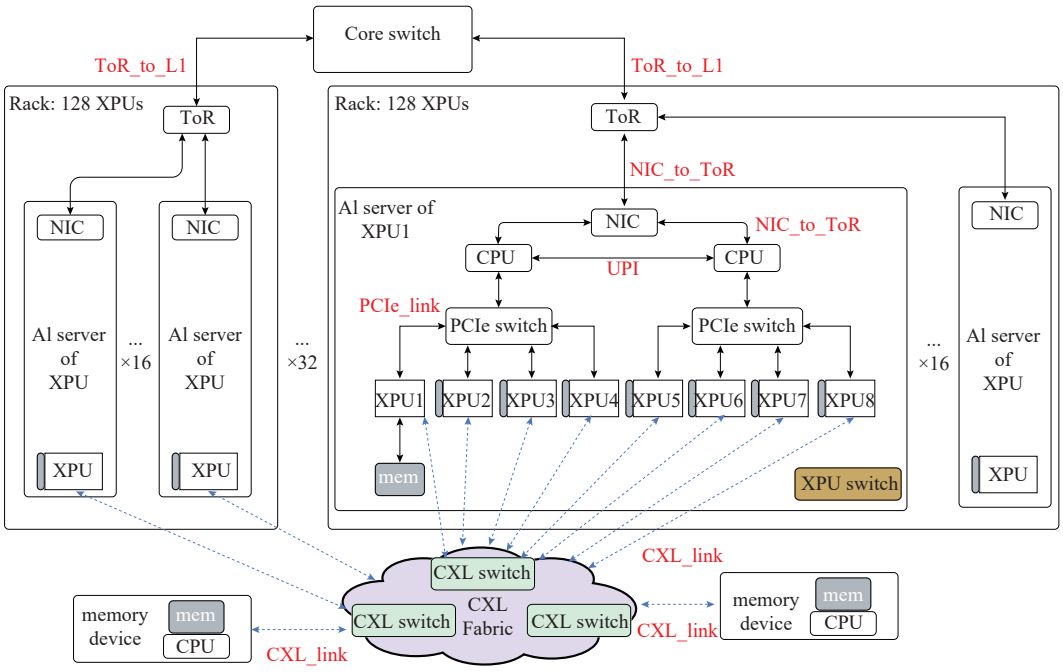
1)互连拓扑配置

为了更好地对比与分析引入一致性数据访问之前计算系统的局限性,以及引入一致性数据访问后计算系统的优势,仿真首先构建了传统未引入一致性数据访问的互连拓扑. 如图 6(a)所示,在构建的传统互连拓扑中,服务器内的互连架构借鉴了 IEIT 5468 服务器的架构[47]. 该架构采用 Balance 拓扑,每颗 CPU 下连接 1 个 PCIe switch,每个 PCIe switch 连接 4 块异构算力(XPU). 此外,考虑到诸如英伟达等厂商提供了 NVSwitch[48] 等高效的异构算力互连能力,本文研究在该拓扑中引入了 XPU switch,以表达算力间的高效互连能力. 基于 XPU switch,服务器内的异构算力之间可以实现比 PCIe switch 更强大的互连互访能力. 对于服务器之间的互连架构,整体计算系统的拓扑采用 leaf-spine 架构. 具体来说,所有服务器对外的互连采用以太网互连,且同一个机架上的所有服务器会被连接到机顶交换机(ToR switch)上. 所有机顶交换机最终连接到一个核心交换机(Core switch)上,从而实现所有服务器和异构算力之间的互连互访. 由于缺乏一致性互连技术,扩展内存只能与服务器中的 CPU 互连. 本文研究构建了包含 4 096 个 XPU 的计算系统,其中共有 32 个 ToR,每个 ToR 包含 16 个服务器,每个服务器中挂载 8 个 XPU,并设定每个 ToR 内的 128 个 XPU 类型相同.

在图 6(a)的拓扑基础之上,进一步构建了图 6(b)所示的引入一致性数据访问技术 CXL[49-51] 后的互连拓扑. 该拓扑与图 6(a)中拓扑的主要区别在于,该拓扑中额外引入了基于 CXL 3.0[15] 特性的 CXL switch,构建了一致性互连网络(CXL Fabric),用于建模一致性数据访问. 在该系统架构中,一致性互连网络不仅



(a) 未引入一致性互连技术的互连拓扑架构



(b) 引入了一致性互连技术CXL后的互连拓扑架构

Fig. 6 Architecture diagram of interconnect topology without and with the introduction of the consistent interconnect technology CXL

图 6 未引入一致性互连技术与引入了一致性互连技术 CXL 的互连拓扑架构图

为异构算力之间(CXL Type 2 设备)的内存互访提供了快速通路，而且为异构算力提供了额外的内存扩展(CXL Type 3 设备). 如图 6 所示，所有通过蓝色箭头连接的内存设备之间都可以使用高速的 CXL 链路进行一致性访问. 此外，由于 CXL switch 具备级联能力，因此每个机架中的 CXL switch 还可以进行级联扩展. 本文研究基于 CXL 3.0 的能力，将 4 096 个 XPU 通过 CXL switch 互连. 外置的扩展内存也可以直接连接到一致性互连网络中，供 XPU 使用.

为了公平对比图 6(a)与图 6(b)的拓扑，仿真中设计为 2 个拓扑中的每个 XPU 都配置了可以无阻塞访问的扩展内存. 对于未引入一致性数据访问的互连拓扑，如图 6(a)所示，仿真配置了 CPU 与 PCIe 之间的 4 条 PCIe 连接，保证每个 XPU 有充足的带宽访问 CPU 下的外置内存. 对于引入一致性数据访问的互连拓扑，如图 6(b)所示，仿真为每个 XPU 都配置了唯一可访问且不冲突的 CXL Type 3 内存设备. 通过以上配置，在 XPU 主存不足时，仿真中每个 XPU 都可以无阻塞地访问扩展内存，并按照负载图的定义执行相应的数据读取等过程.

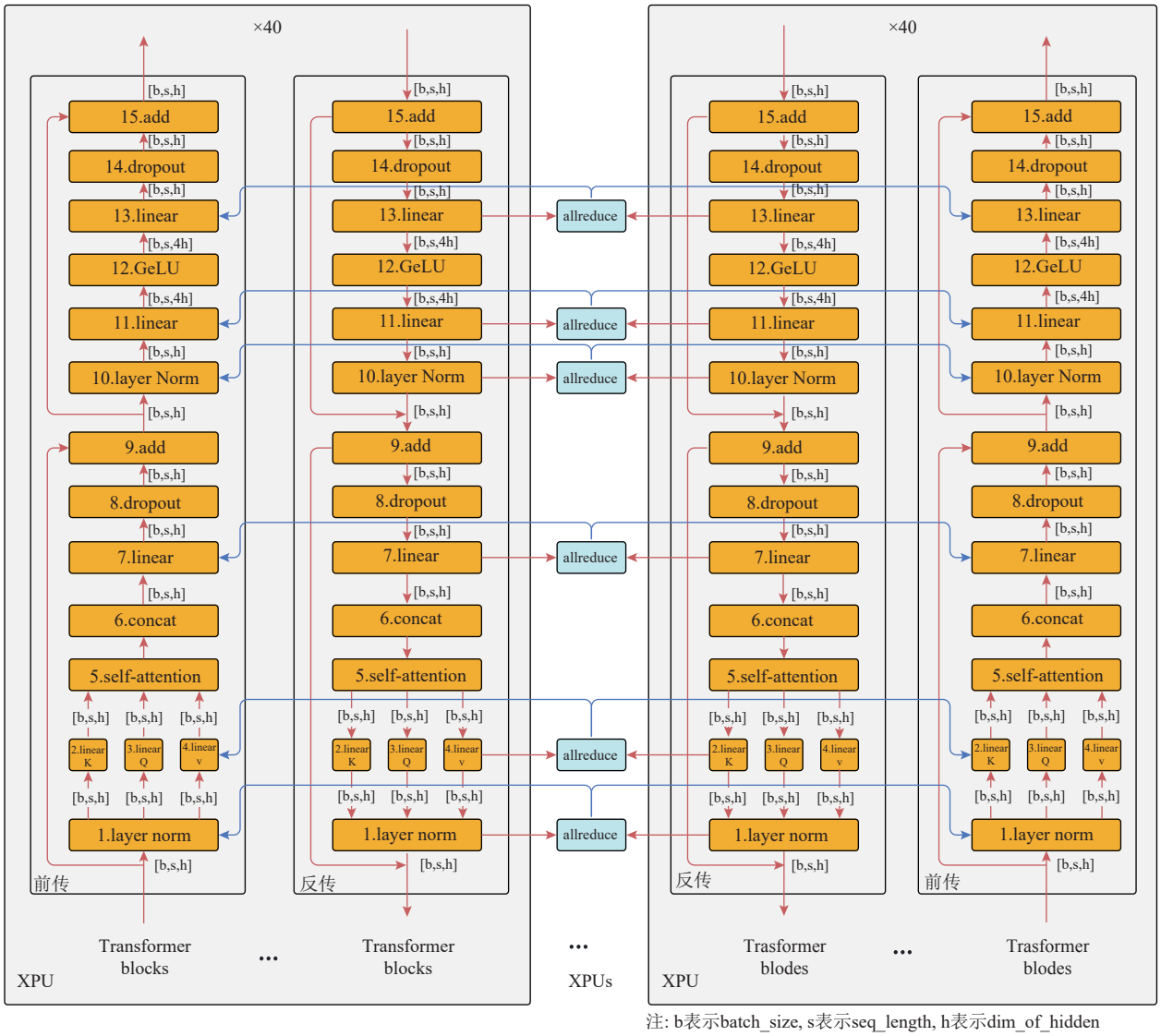
2)AI 计算负载配置

本文研究通过建模与分析异构一致性融合计算系统在处理 LLM 训练任务时的性能表现评估该计算系统的性能,因此建立了如图 7 所示的 LLAMA2-13B[52] 的负载图. 由于训练 LLAMA2-13B 的主要耗时集中在计算其模型架构中的 40 个 LLaMA decoder,因此该负载图描述了这些 LLaMA decoder 的并行训练过程. 其中,在 XPU 本地内存有剩余时,橙色顶点代表在本地内存的计算过程;在 XPU 本地内存不足时,对于非一致性系统,橙色顶点代表如图 3(b)所示的先内存搬运再计算过程,而对于一致性系统,橙色顶点代表如图 3(c)所示的直接访问远端内存进行计算过程. 为了构建 HCSim 的负载图,本文研究根据矩阵乘法、self-attention、layer normalization、concat、dropout、GeLU 等神经层的计算特征,估计了如表 4 所示的每个神经层的计算量和访存量,并将其记录到负载图中相应的计算与通信顶点中,其中本文研究使用 FP32 训练精度. 然后,结合每个神经层在反向传播过程中所需同步的梯度数据量,将其记录到表示参数同步过程的相应通信顶点中,从而完成了 LLAMA2-13B 负载图的所有参数配置,使其能够被 HCSim 读取并用于模拟执行.

3)仿真参数

由于现有仿真工作缺乏对异构一致性融合计算系统中互连拓扑、计算负载的合理建模,因此,本文研究的仿真聚焦于探究 HCSim 本身的仿真能力.

对于拓扑架构和 AI 计算负载的其他参数,仿真中的设定如表 5 所示,其中参考了当前构建计算集群的主流性能参数. 对于表 5 里未标记为变量的参数,在仿真中配置为定值,不会发生变化. 对于标记为变



注: b表示batch_size, s表示seq_length, h表示dim_of_hidden

Fig. 7 Workload graph for distributed training of LLAMA2-13B

图 7 LLAMA2-13B 的分布式训练负载图

Table 4 Computational Load and Memory Access of Each Neural Layer in LLaMA Decoder

表 4 LLaMA Decoder 中每个神经层的计算量与访存量

神经层	前传计算量/GFLOPs	前传访存量/GB	反传计算量/GFLOPs	反传访存量/GB
layer norm	可忽略	0.000 02+0.039×b	可忽略	0.000 15 + 0.156b
linear (K)	b×195	0.048 8+0.039×b	b×390	0.39 + 0.156b
linear (Q)	b×195	0.048 8+0.039×b	b×390	0.39 + 0.156b
linear (V)	b×195	0.048 8+0.039×b	b×390	0.39 + 0.156b
self-attention	b×312.5	3.242×b	b×625	0.234b
concat	可忽略	可忽略	可忽略	可忽略
linear	b×195	0.048 8+0.039×b	b×390	0.39 + 0.156b
dropout	可忽略	0.019 5×b	可忽略	0.019 5
add	可忽略	0.078×b	可忽略	0.039
layer norm	可忽略	0.000 02+0.039×b	可忽略	0.000 15 + 0.156b
linear	b×780	0.195 + 0.039×b	b×1 560	1.562GB + 0.39b
GeLU	可忽略	0.156×b	可忽略	0.078
linear	b×780	0.195 + 0.156×b	b×1 560	1.562GB + 0.39b
dropout	可忽略	0.019 5×b	可忽略	0.019 5
add	可忽略	0.078×b	可忽略	0.156

量的参数，如 CXL_link 的时延、带宽等，仿真中将对其进行调整，用于对比不同参数对系统性能和任务执行效率的影响. 仿真使用的服务器配置了 Intel Xeon Platinum 8168 @ 2.70 GHz 处理器，内存为 DDR4 2 666 MHz.

3.2 仿真结果分析

1）对比引入/不引入一致性的计算系统性能

首先，给定 CXL_link 的带宽与时延分别为 128 GB/s 和 200 ns，本文研究对比了在使用不同数量的算力节点时，引入或不引入一致性的计算系统性能. 本文研究通过整个计算系统每秒处理的 batch 数量来

Table 5 Simulation Parameter Settings

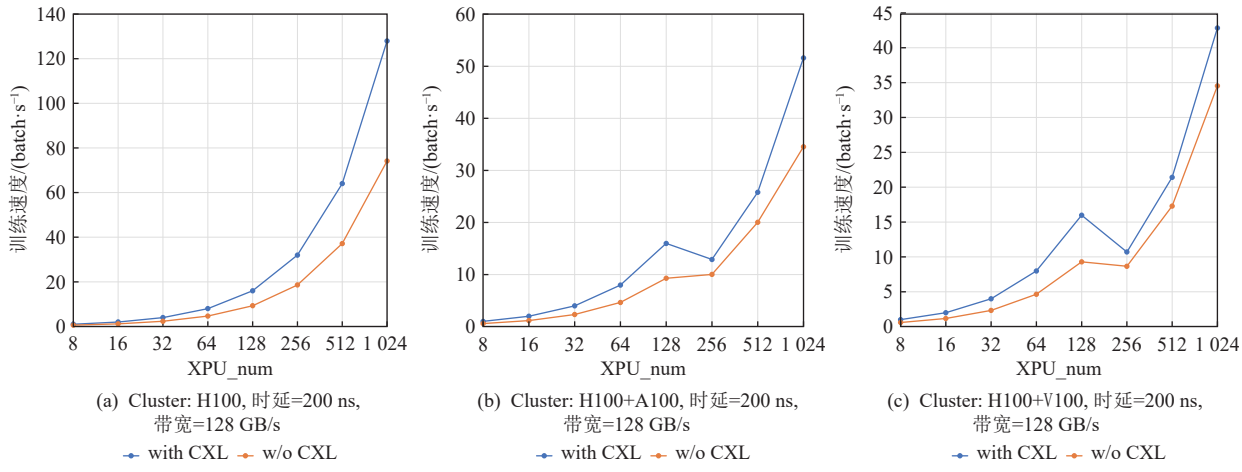
表 5 仿真参数设定

异构一致性融合计算系统			LLaMA2-13B 任务配置	
算力节点	带宽	时延		
ToR_to_L1	12.5 GB/s	5 μs	训练时每个 XPU 输入的 batch_size	1
NIC_to_ToR	12.5 GB/s	5 μs	seq_len	4 096
UPI	62.4 GB/s	100 ns	隐藏层维度	5 120
PCIE_link	128 GB/s	250 ns	使用算力节点数量（scale）	变量
XPU_link	900 GB/s	100 ns		
CXL_link	变量	变量		

对比不同参数配置下的系统性能，并针典型的英伟达 H100、A100、V100 算力，仿真了 H100 同构计算系统和 H100+A100，H100+V100 两种异构计算系统. 对于 FPGA，ASIC 等异构算力的仿真，也可以通过在 HCSim 中自定义算力节点实现建模仿真. 图 8 展示了仿真结果. 仿真记录 HCSim 仅用 756.6 s 便完成了包含 1 024 个算力节点的一个训练迭代的仿真，这也充分说明了 HCSim 在对异构一致性融合计算系统性能进行仿真时具有优秀的执行效率.

从图 8 看出，无论是在同构还是异构的情况下，引入一致性后的计算系统性能明显优于不引入一致性的计算系统性能. 为了分析性能差异的原因，本文研究提取了如图 9 所示的仿真输出执行轨迹，其中包含了 HCSim 中每个计算过程和通信过程在完成时的仿真时间. 经过对执行轨迹的分析，以上性能差异的原因主要包括 2 点：

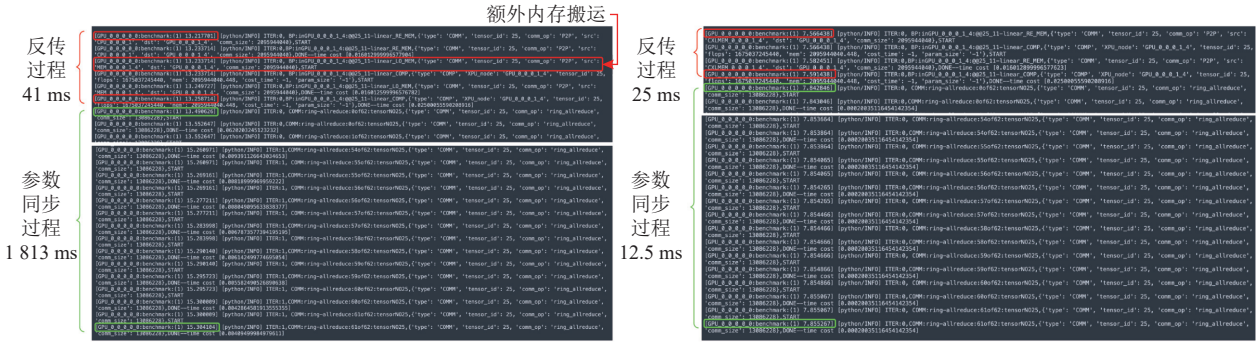
①在不引入一致性的计算系统中，一旦 XPU 本地内存不足，就需要引入额外的内存搬运过程，导致训练耗时增加. 相比之下，从图 9(b)的反向传播过程



(a) Cluster: H100, 时延=200 ns, 带宽=128 GB/s (b) Cluster: H100+A100, 时延=200 ns, 带宽=128 GB/s (c) Cluster: H100+V100, 时延=200 ns, 带宽=128 GB/s

Fig. 8 Simulation result on the computing system performance with and without introducing consistency

图 8 对比引入/不引入一致性的计算系统性能的仿真结果



(a) w/o CXL (b) with CXL

Fig. 9 Execution trace of HCSim for the same operator output

图 9 HCSim 对同一个算子输出的执行轨迹

轨迹可以明显看出，由于一致性互连技术 CXL 的引入，反传过程可以省略如图 3 所示的额外本地内存搬运步骤，从而节约训练耗时.

②对比图 9(a)中的参数同步过程轨迹可以发现，在不引入一致性的计算系统中，参数同步过程明显耗时更长，这是由于在该系统中异构算力节点之间的梯度同步仍需要通过以太网进行，效率较低. 相比之下，如图 6 所示，引入一致性后算力之间可以通过一致性互连网络进行通信，从而显著提升参数同步过程的执行效率.

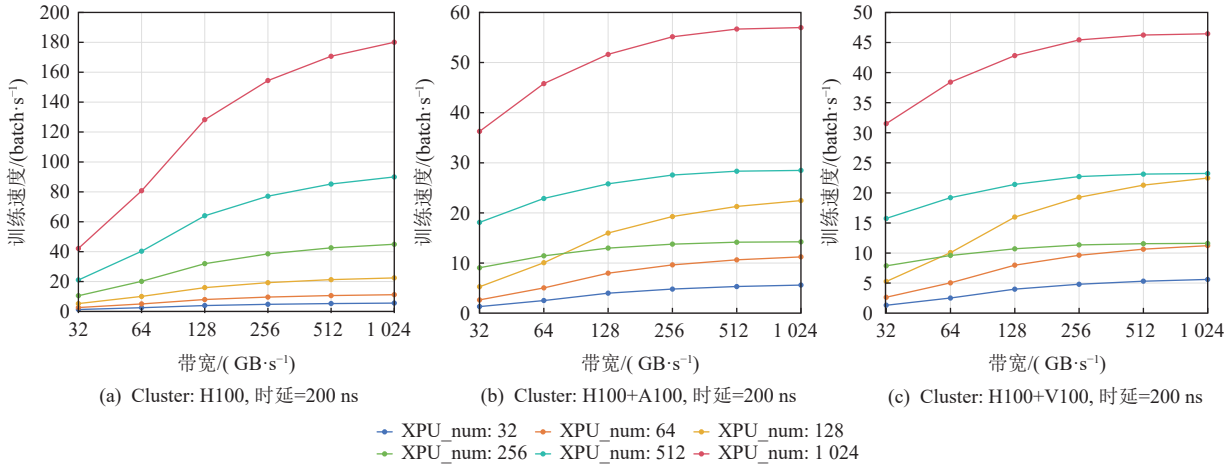
另外，值得注意的是，对于 H100+A100、H100+V100 两种异构计算系统，在节点数量从 128 增加至 256 时，计算系统的性能出现了明显跳变. 这应该是由于在每个 ToR 内的 XPU 类型相同的仿真条件下，在 128 个节点以下时，计算系统仍然保持着同构. 而当节点数量从 128 提升至 256 时，系统引入了低性能的异构计算卡，降低了系统的整体计算性能. 这个仿真结果充分说明了对于异构计算系统，一味地增加 XPU 的数量并不一定会带来计算系统性能的提升，还需考虑 XPU 之间是否存在计算性能的差异.

2)探究带宽对计算系统性能的影响

进一步地，本节探究带宽对异构一致性融合计算系统性能的影响. 仿真采用图 6(b)中的互连架构，并固定 CXL_link 的时延为 200 ns，通过修改 CXL_link 的带宽与算力节点规模，对 LLAMA2-13B 的并行训练进行仿真，并观察不同计算系统性能的变化. 图 10 展示了这部分的仿真结果，经过分析可以得到以下结论：

①在算力节点规模不变时，随着 CXL_link 一致性互连带宽的提升，计算系统的性能会逐渐提升，但提升的效果逐渐减缓. 这是因为带宽的提升缓解了带宽瓶颈，直接加速了内存访问与参数同步过程，从而提升了计算系统的整体性能. 而随着带宽的增加，系统的瓶颈逐渐从带宽转移到了算力，最终导致提升效果不再显著.

②在带宽不变的情况下，参与计算任务的 XPU 数量的增加显著提升了计算系统的性能. 这应该是由于一致性互连为 XPU 之间带来了高效的通信能力，



(a) Cluster: H100, 时延=200 ns (b) Cluster: H100+A100, 时延=200 ns (c) Cluster: H100+V100, 时延=200 ns

Fig. 10 Impact of bandwidth on the performance of heterogeneous consistency-integrated computing systems

图 10 带宽对异构一致性融合计算系统性能的影响

提升了计算系统的可扩展性.

③引入异构算力同样会带来计算系统性能的波动，导致提升系统带宽的性能收益减弱.

3)探究时延对计算系统性能的影响

固定 CXL_link 带宽为 128 GB/s，本文研究通过修改 CXL_link 时延与算力节点规模，从而观察不同计算系统性能的变化. 仿真结果如图 11 所示，经过分析，可以得到以下结论：

①在时延较小时，时延略微增加并不会影响计算系统的性能. 这应该是由于在时延较小时，时延对通信影响的比重较低，不会明显影响系统性能.

②随着一致性互连时延的增大，特别是在时延大于 10 μs 后，时延逐渐成为影响系统性能的关键因素. 这是因为参数同步过程中有大量的通信过程，这些过程受时延影响较大，而时延增加时该影响也会更加显著.

③在时延变化时，计算节点数量更多的计算任务会受到更大的影响. 这是由于计算节点较多时，参数同步过程中通信的次数也会随之增加，造成时延的影响更加显著.

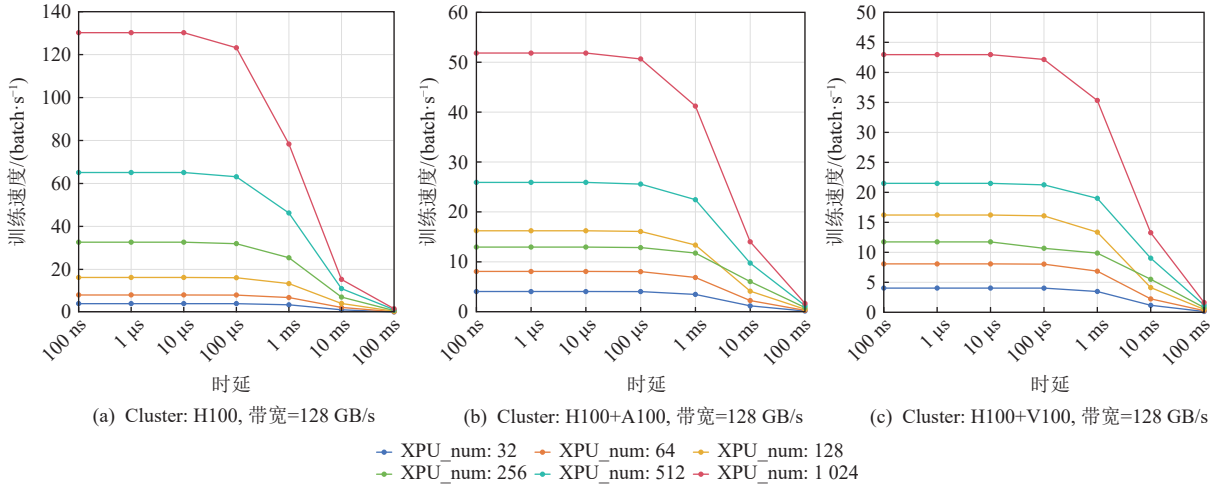


Fig. 11 Impact of latency on the performance of heterogeneous consistency-integrated computing systems

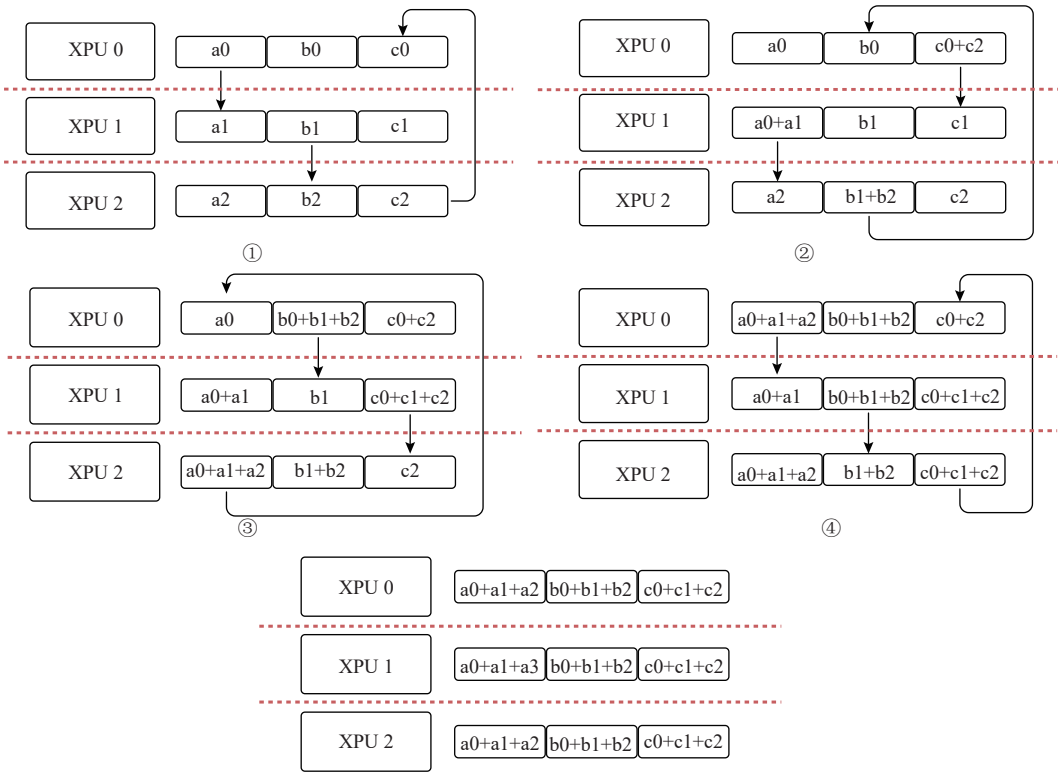
图 11 时延对异构一致性融合计算系统性能的影响

3.3 面向异构一致性融合计算系统的通信优化与仿真分析

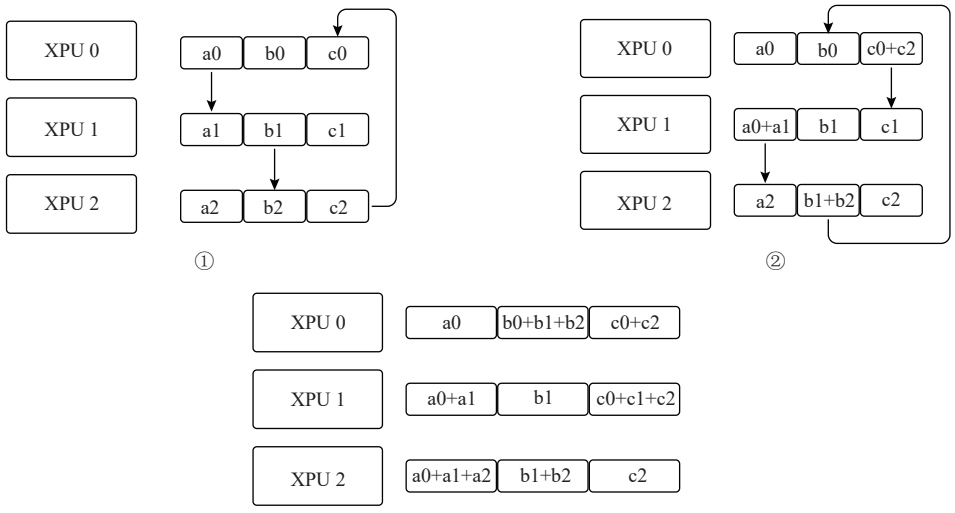
进一步地，针对异构一致性融合计算系统的通信优化问题，本文研究提出了基于一致性 ring-allreduce 的通信优化方法，并利用 HCSim 的仿真能力，验证了所提方法的有效性.

具体而言，在参与分布式训练任务的节点规模较大或模型参数量较多时，由于节点之间需要通过通信同步模型参数，节点之间的通信将影响任务执行效率. 本文研究提出了一致性 ring-allreduce，借助一致性系统的特性，降低了参数同步的通信次数，缓解了通信对 AI 计算任务的影响. 图 12 展示了传统 ring-allreduce 和本文研究所提出的一致性 ring-allreduce 之间参数同步运作流程的区别. 如图 12(a)所示，在传统的 ring-allreduce 中，受制于异构算力节点内存之间访问的隔离(即图 12(a)的虚线)，每个异构算力节点都需要将同步后的梯度放置到本地内存，才能实现参数同步更新. 对于 N 个需要同步参数的异构算力节点，总共需要 $2\times(N-1)$ 次通信. 相比之下，如图 12(b)所示，一致性 ring-allreduce 只需要进行 $(N-1)$ 次通信便可完成梯度同步，减少了一半的梯度同步的通信步骤. 这是因为在一致性系统中，由于 XPU 之间的内存可以互相访问，因此只需要在一致性系统中保证平均后的梯度存在(即图 12(b)的 a0+a1+a2、b0+b1+b2、c0+c1+c2)，而不再需要使用额外的 all-gather 集合通信将同步后的梯度发送到所有的 XPU 内存上.

基于这一思路，本文研究在 HCSim 中调整了负载图中对 ring-allreduce 的执行定义，在给定 CXL_link 的带宽时延分别为 128 GB/s 和 200 μs 的情况下，获得的仿真结果如表 6 所示. 仿真结果表明，虽然一致性 ring-allreduce 可以提升系统性能，但提升并不显著. 对于整体算力性能较强，或系统规模较大的计算系统，应用一致性 ring-allreduce 可以带来相对更大的收益. 总结来看，虽然提升的幅度较小，一致性 ring-allreduce 可以在没有额外性能损失的情况下合理利用一致性互连的特性，实现在异构一致性融合计算系统中提升 AI 训练任务的执行效率.



(a) 传统ring-allreduce



(b) 一致性ring-allreduce

Fig. 12 Comparison between traditional ring-allreduce and consistent ring-allreduce

图 12 传统 ring-allreduce 和一致性 ring-allreduce 的对比

Table 6 Comparison Results Before and After Optimization

表 6 优化前后对比结果

计算系统架构	传统 ring-allreduce	一致性 ring-allreduce
64 个 H100 同构	7.98 batch/s	8.12 batch/s
256 个 H100 同构	31.97 batch/s	32.53 batch/s
128 个 H100+128 个 A100 异构	10.65 batch/s	10.71 batch/s
128 个 H100+128 个 V100 异构	12.86 batch/s	12.93 batch/s

3.4 仿真小节

针对异构一致性融合计算系统，本文研究利用 HCSim 的仿真能力，不仅建模分析了引入一致性技术的效果，而且建模探究了一致性互连网络中带宽、时延对系统性能的影响，最终进一步模拟仿真了所提出的一致性 ring-allreduce 对该系统性能的提升. 这充分说明了 HCSim 不仅可以用于异构一致性融合计

算系统的性能建模，而且可以通过少量修改完成对异构一致性融合计算系统的优化方案验证.

4 结论与展望

针对新型异构一致性融合计算系统的性能建模和优化困难等问题，本文研究提出了一种全新的建模仿真工具 HCSim. 该工具解决了以往研究中异构一致性融合计算系统拓扑架构建模困难和计算负载建模偏差的问题，并集成了 SimGrid 仿真器，实现了可以对异构一致性融合计算系统进行灵活高效的性能建模与仿真. 本文研究利用 HCSim 模拟构建了一致性互连拓扑架构，并建模了 LLAMA2-13B 的训练负载. 基于 HCSim，仿真分析了异构算力分布、带宽、时延和计算规模等变量对异构一致性融合计算系统性能和 AI 计算任务执行效率的影响. 此外，还针对该系统的通信问题，提出了基于一致性 ring-allreduce 的通信优化方法，并使用 HCSim 进行了仿真验证. 通过仿真可以看出，HCSim 不仅可以低成本、高效地实现对异构一致性融合计算系统的性能建模，而且能够对异构一致性融合计算系统的优化方案进行仿真验证. 在未来，我们将继续扩展 HCSim 的仿真能力，包括加入对时间局部性、空间局部性等算子的建模以及加入对一致性维护开销的建模以及加入对更多典型互连拓扑建模的支持，并在仿真能力更加完整后进行开源. 希望本文研究提出的 HCSim 能为工业界和学术界提供低成本的异构一致性融合计算系统性能评估手段，并为未来新型计算系统的仿真建模提供一些新的思路.

作者贡献声明：李仁刚提出了性能建模的核心思路，撰写仿真器核心代码和论文的主要章节；唐轶男修改与校对论文；郭振华提出了一致性 ring-allreduce 的思路；王丽实现了一致性 ring-allreduce 的仿真代码；宗瓒进行了实验部分数据的汇总与论文相关部分的撰写；杨广文负责论文的整体指导.

参 考 文 献

[1] Guo Daya, Yang Dejian, Zhang Haowei, et al. Deepseek-r1: Incentivizing reasoning capability in LLMs via reinforcement learning[J]. arXiv preprint, arXiv: 2501.12948, 2025

[2] Huang Dawei, Yan Chuan, Li Qing, et al. From large language models to large multimodal models: A literature review[J]. Applied Sciences, 2024, 14(12): 5068

[3] Jiang Ziheng, Lin Haibin, Zhong Yinmin, et al. MegaScale: Scaling large language model training to more than 10, 000 GPUs[C]//Proc of 21st USENIX Symp on Networked Systems Design and Implementation (NSDI 24). Berkeley, CA: USENIX Association, 2024: 745−760

[4] Yang Zhuoping, Ji Shixin , Chen Xingzhen, et al. Challenges and opportunities to enable large-scale computing via heterogeneous chiplets[C]//Proc of the 29th Asia and South Pacific Design Automation Conf (ASP-DAC) . Piscataway, NJ: IEEE, 2024: 765−770

[5] Saghiri A M, Vahidipour S M, Jabbarpour M R, et al. A survey of artificial intelligence challenges: Analyzing the definitions, relationships, and evolutions[J]. Applied Sciences, 2022, 12(8): 4054

[6] Rajbhandari S, Ruwase O, Rasley J, et al. Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning[C]//Proc of the Int Conf for High Performance Computing, Networking, Storage And Analysis 2021. New York: ACM, 2021: 1−14

[7] Korthikanti V A, Casper J, Lym S, et al. Reducing activation recomputation in large transformer models[C]//Proc of Machine Learning and Systems 5, 2023: 341−353. https://proceedings.mlsys.org/paper_files/paper/2023/hash/80083951326cf5b35e5100260d64ed81-Abstract-mlsys2023.html

[8] Li Shen, Zhao Yanli, Varma R, et al. Pytorch distributed: Experiences on accelerating data parallel training[J]. arXiv preprint, arXiv: 2006.15704, 2020

[9] Ge Suyu, Zhang Yunan, Liu Liyuan, et al. Model tells you what to discard: Adaptive KV cache compression for LLMs[J]. arXiv preprint, arXiv: 2310.01801, 2023

[10] Chen Zixiang, Deng Yihe, Wu Yue, et al. Towards understanding mixture of experts in deep learning[J]. arXiv preprint, arXiv: 2208.02813, 2022

[11] Xu Yi, Mahar S, Liu Ziheng, et al. CXL shared memory programming: Barely distributed and almost persistent[J]. arXiv preprint, arXiv: 2405.19626, 2024

[12] Schieffer G, Wahlgren J, Ren Jie, et al. Harnessing integrated CPU-GPU system memory for HPC: A first look into grace hopper[C]//Proc of the 53rd Int Conf on Parallel Processing. New York: ACM, 2024: 199−209

[13] Xia Jing, Cheng Chuanning, Zhou Xiping, et al. Kunpeng 920: The first 7-nm chiplet-based 64-core arm soc for cloud services[J]. IEEE Micro, 2021, 41(5): 67−75

[14] Fusco L, Khalilov M, Chrapek M, et al. Understanding data movement in tightly coupled heterogeneous systems: A case study with the Grace Hopper superchip[J]. arXiv preprint, arXiv: 2408.11556, 2024

[15] CXL Organization. CXL® Specification[EB/OL]. 2020[2024-12-01]. https://computeexpresslink.org/cxl-specification/.

[16] Gholami A, Yao Zheiwei, Kim S, et al. AI and memory wall[J]. IEEE Micro, 2024, 44(3): 33−39

[17] Casanova H. SimGrid: A toolkit for the simulation of application scheduling[C]//Proc of the 1st IEEE/ACM Int Symp on Cluster

Computing and the Grid. Piscataway, NJ: IEEE, 2021: 430−437

[18] Casanova H, Giersch A, Legrand A, et al. Lowering entry barriers to developing custom simulators of distributed applications and plat forms with SimGrid[J]. Parallel Computing, 2025: 103125. https://www.sciencedirect.com/science/article/pii/S0167819125000018

[19] Saleh E, Shastry C. Simulation and modelling of task migration in distributed systems using SimGrid[C]//Proc of the Int Conf on Modeling, Simulation and Optimization. Singapore: Springer Nature Singapore, 2022: 475−486

[20] Guo Zhenhua, Tang Yinan, Zhai Jidong, et al. A survey on performance modeling and prediction for distributed DNN training[J]. IEEE Transactions on Parallel and Distributed Systems, 2024, 35(12): 2463−2478

[21] Lowe-Power J, Ahmad A. M, Akram A, et al. The Gem5 simulator: Version 20.0+[J]. arXiv preprint, arXiv: 2007.03152, 2020

[22] Bellard F. QEMU, a fast and portable dynamic translator[C]//Proc of USENIX Annual Technical Conf 2005. Berkeley, CA: USENIX Association, FREENIX Track, 2005: 10−5555

[23] Bakhoda A, Yuan G L, Fung W, et al. Analyzing CUDA workloads using a detailed GPU simulator[C]//Proc of 2009 IEEE Int Symp on Performance Analysis of Systems and Software. Piscataway, NJ: IEEE, 2009: 163−174

[24] Li Shang, Yang Zhiyuan, Reddy D, et al. DRAMSim3: A cycle-accurate, thermal-capable DRAM simulator[J]. IEEE Computer Architecture Letters, 2020, 19(2): 106−109

[25] Kim Y, Yang Weikun, Mutlu O. Ramulator: A fast and extensible DRAM simulator[J]. IEEE Computer Architecture Letters, 2015, 15(1): 45−49

[26] Henderson T R, Lacage M, Riley G F, et al. Network simulations with the NS−3 simulator[C]//Proc of SIGCOMM Demonstration 2008. New York: ACM, 2008. 527

[27] Varga A. OMNeT++[J]. Modeling and Tools for Network Simulation, 2010: 35−59

[28] Khairy M, Shen Zhesheng, Aamodt T M, et al. Accel-Sim: An extensible simulation framework for validated GPU modeling[C]//Proc of 2020 ACM/IEEE 47th Annual Int Symp on Computer Architecture (ISCA). Piscataway, NJ: IEEE, 2020: 473−486

[29] Won W, Heo T, Rashidi S, et al. Astra-sim2.0: Modeling hierarchical networks and disaggregated systems for large-model training at scale[C]//Proc of 2023 IEEE Int Symp on Performance Analysis of Systems and Software (ISPASS). Piscataway, NJ: IEEE, 2023: 283−294

[30] Moolchandani D, Kundu J, Ruelens F, et al. AMPeD: An analytical model for performance in distributed training of transformers[C]//Proc of 2023 IEEE Int Symp on Performance Analysis of Systems and Software (ISPASS). Piscataway, NJ: IEEE, 2023: 306−315

[31] Isaev M, McDonald N, Dennison L, et al. Calculon: A methodology and tool for high-level co-design of systems and large language models[C]//Proc of the Int Conf for High Performance Computing, Networking, Storage and Analysis. New York: ACM, 2023: 1−14

[32] Qi Hang, Sparks E R, Talwalkar A. Paleo: A performance model for deep neural networks[C]//Proc of Int Conf on Learning Representations. France, Toulon: MIT, 2017: 1−10

[33] Lu Wenyan, Yan Guihai, Li Jiajun, et al. FlexFlow: A flexible dataflow accelerator architecture for convolutional neural networks[C]//Proc of 2017 IEEE Int Symp on High Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2017: 553−564

[34] Zhu Hongyu, Phanishayee A, Pekhimenko G. Daydream: Accurately estimating the efficacy of optimizations for DNN training[C]//Proc of 2020 USENIX Annual Technical Conf (USENIX ATC 20) . Berkeley, CA: USENIX Association, 2020: 337−352

[35] Hu Hanpeng, Jiang Chenyu, Zhong Yuchen, et al. Dpro: A generic performance diagnosis and optimization toolkit for expediting distributed DNN training[C]//Proc of Machine Learning and Systems. New York: ACM, 2022: 623−637

[36] Lu Guandong, Chen Runzhe, Wang Yakai, et al. DistSim: A performance model of large-scale hybrid distributed DNN training[C]//Proc of the 20th ACM Int Conf on Computing Frontiers. New York: ACM, 2023: 112−122

[37] Santhanam K, Krishna S, Tomioka R, et al. DistIR: An intermediate representation for optimizing distributed neural networks[C]//Proc of the 1st Workshop on Machine Learning and Systems. New York: ACM, 2021: 15−23

[38] Lattner C, Amini M, Bondhugula U, et al. MLIR: Scaling compiler infrastructure for domain specific computation[C]//Proc of 2021 IEEE/ACM Int Symp on Code Generation and Optimization (CGO). Piscataway, NJ: IEEE, 2021: 2−14

[39] Duan Jiangfei, Li Xiuhong, Xu Ping, et al. Proteus: Simulating the performance of distributed DNN training[J]. IEEE Transactions on Parallel and Distributed Systems, 2024, 35(10): 1867−1878

[40] Zhang Shiwei, Yi Xiaodong, Diao Lansong, et al. Expediting distributed DNN training with device topology-aware graph deployment[J]. IEEE Transactions on Parallel and Distributed Systems, 2023, 34(4): 1281−1293

[41] Wang Haoran, Tachon T, Li Chong, et al. SMSG: Profiling-free parallelism modeling for distributed training of DNN[J]. International Journal of Parallel Programming, 2023, 51(2): 109−127

[42] Rashidi S, Sridharan S, Srinivasan S, et al. Astra-sim: Enabling sw/hw co-design exploration for distributed DL training platforms[C]//Proc of 2020 IEEE Int Symp on Performance Analysis of Systems and Software (ISPASS). Piscataway, NJ: IEEE, 2020: 81−92

[43] Samajdar A, Zhu Yuhao, Whatmough P, et al. Scale-sim: Systolic CNN accelerator simulator[J]. arXiv preprint, arXiv: 1811.02883, 2018

[44] Liu ZhiGang, Whatmough P N, Mattina M. Systolic tensor array: An efficient structured-sparse GEMM accelerator for mobile CNN inference[J]. IEEE Computer Architecture Letters, 2020, 19(1): 34−37

[45] Hagberg A, Swart P J, Schult D A. Exploring network structure,

dynamics, and function using NetworkX (No. LA-UR-08-05495; LA-UR-08-5495)[R]. Los Alamos, NM: Los Alamos National Laboratory (LANL), 2008

[46] The SimGrid Team. The SimGrid models[EB/OL]. 2002[2025-01-17]. https://simgrid.frama.io/simgrid/Models.html#cm02

[47] IEIT Systems. meta brain® Artificial Intelligence Servers > AI > Servers > NF5468A5[EB/OL]. [2025-03-01]. https://en.ieisystem.com/product/ai/9573.html

[48] Li Ang, Song S L, Chen Jieyang, et al. Evaluating modern GPU interconnect: PCIe, nvlink, nv-sli, NVSwitch and gpudirect[J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 31(1): 94−110

[49] Das Sharma D, Blankenship R, Berger D. An introduction to the compute express link (CXL) Interconnect[J]. ACM Computing Surveys, 2024, 56(11): 1−37

[50] Wang Yanwei, Li Rengang, Xu Ran, et al. Data center heterogeneous acceleration software-hardware system-level platform based on reconfigurable architecture[J]. Journal of Computer Research and Development, 2024, 61(6): 1388−1400 (in Chinese)
(王彦伟, 李仁刚, 徐冉, 等. 基于可重构架构的数据中心异构加速软硬件系统级平台[J]. 计算机研究与发展, 2024, 61(6): 1388−1400)

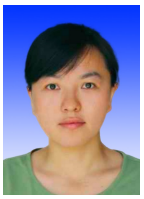
[51] Ge Xuran, Ou Yang, Wang Bo, et al A survey of storage optimization techniques in large language model inference[J]. Journal of Computer Research and Development, 2025, 62(3): 545−562 (in Chinese)
(葛旭冉, 欧洋, 王博, 等. 大模型推理中的存储优化技术研究综述[J]. 计算机研究与发展, 2025, 62(3): 545−562)

[52] Touvron H, Martin L, Stone K, et al. Llama 2: Open foundation and fine-tuned chat models[J]. arXiv preprint, arXiv: 2307.09288, 2023

Li Rengang, born in 1980. PhD candidate, senior engineer. His main research interest includes heterogeneous computing.
李仁刚, 1980 年生. 博士研究生, 正高级工程师. 主要研究方向为异构计算.

Tang Yinan, born in 1993. PhD. His main research interests include heterogeneous computing and simulation system.
唐轶男, 1993 年生. 博士. 主要研究方向为异构计算、仿真系统.

Guo Zhenhua, born in 1988. PhD. His main research interests include computer system architecture and heterogeneous computing.
郭振华, 1988 年生. 博士. 主要研究方向为计算机系统结构、异构计算.



Wang Li, born in 1989. Master. Her main research interests include heterogeneous computing and artificial intelligence.
王　丽, 1989 年生. 硕士. 主要研究方向异构计算、人工智能.

Zong Zan, born in 1994. PhD. His research interest includes performance acceleration of distributed deep learning systems and large-scale data processing systems.
宗　瓒, 1994 年生. 博士. 主要研究方向为分布式深度学习系统和大规模数据处理系统的性能加速.

Yang Guangwen, born in 1963. PhD, professor, PhD supervisor. His main research interest includes high performance computing.
杨广文, 1963 年生. 博士, 教授, 博士生导师. 主要研究方向为高性能计算.