

# 拓扑感知通信优化

# 目录

- 1 网络拓扑基础
- 2 FlagCX拓扑感知实现
- 3 拓扑感知通信优化实践

## 网络拓扑组成要素

节点：网络中的设备，如路由器，交换机，网卡等；这些设备通过传输介质相互连接

链路：两个节点间的传输介质

网络协议：规定网络中数据传输格式、传输控制规则等；决定了网络节点间如何通信

## 物理拓扑和逻辑拓扑区别

物理拓扑：描述网络设备和传输介质之间的实际物理连接方式，关注的是物理设备的实际布局

逻辑拓扑：描述的是网络中数据流动的路径，不直接反应物理连接细节和设备的实际布局

## 点对点拓扑



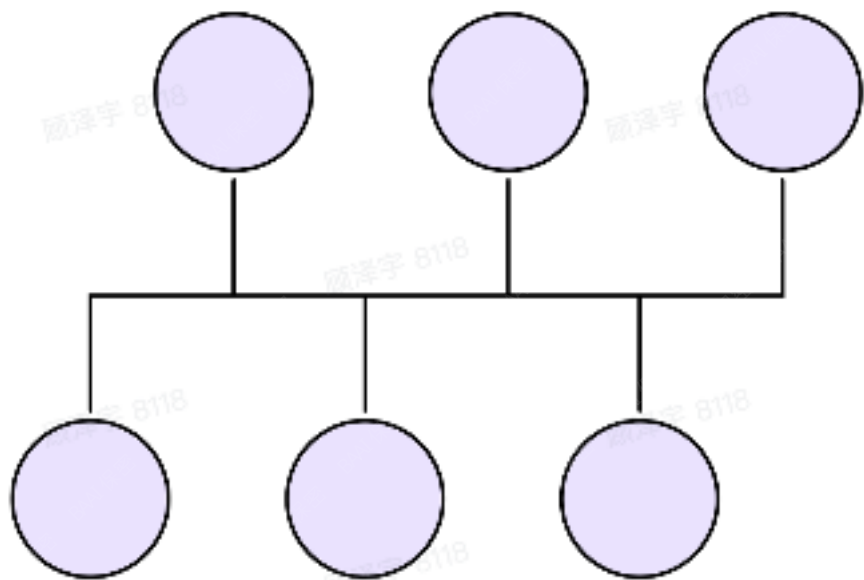
### 优点:

- 简单: 连接简单, 每两个设备间有独立链路
- 低延迟: 数据传输无需经过中转节点, 通信延迟较低
- 高带宽: 每对设备间有独立链路, 不会受到其他设备干扰

### 缺点:

- 扩展性差: 随着设备数量增加, 所需连接数呈指数级增长, 维护困难
- 容错性差: 如果设备或链路故障, 通信会立即中断

## 总线拓扑



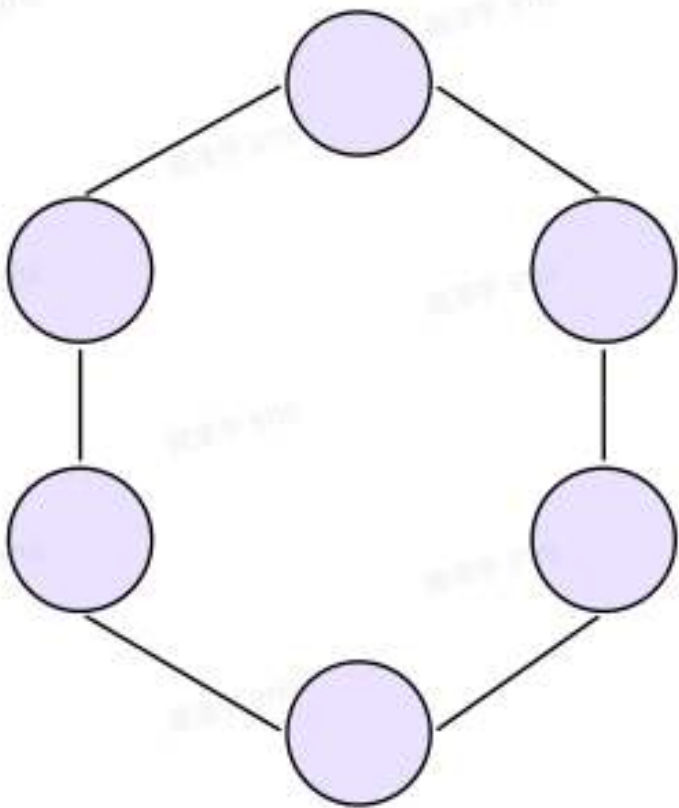
### 优点:

- 成本低: 实现简单, 布线成本低
- 布线简洁: 设备通过一条共享的总线连接, 结构简单

### 缺点:

- 带宽共享: 所有设备共享同一条通信线路, 带宽被分割, 性能下降
- 易受干扰: 信号传输距离越长, 传输质量下降, 易发生干扰
- 故障难定位: 总线出现故障时, 难以定位问题所在

## 环形拓扑



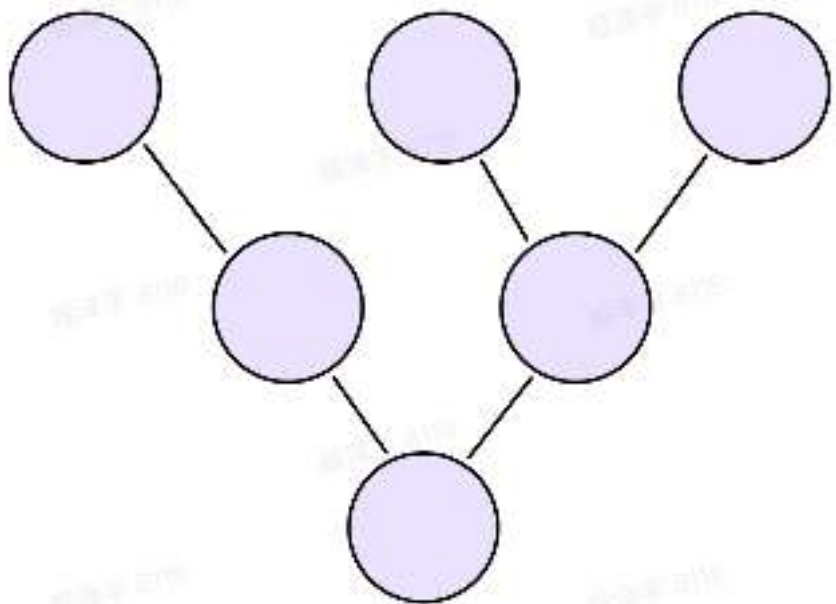
### 优点:

- 结构简单: 每个节点只连接两个邻节点, 拓扑结构简单
- 数据传输有序: 数据按固定顺序传递, 避免数据冲突

### 缺点:

- 故障影响大: 环路中任何一个节点或链路故障都会影响整个网络的通信
- 数据传输延迟: 数据需要沿环路传递, 延迟较高
- 扩展性差: 随着节点数增加, 环路长度增加, 导致延迟和带宽问题

## 树形拓扑



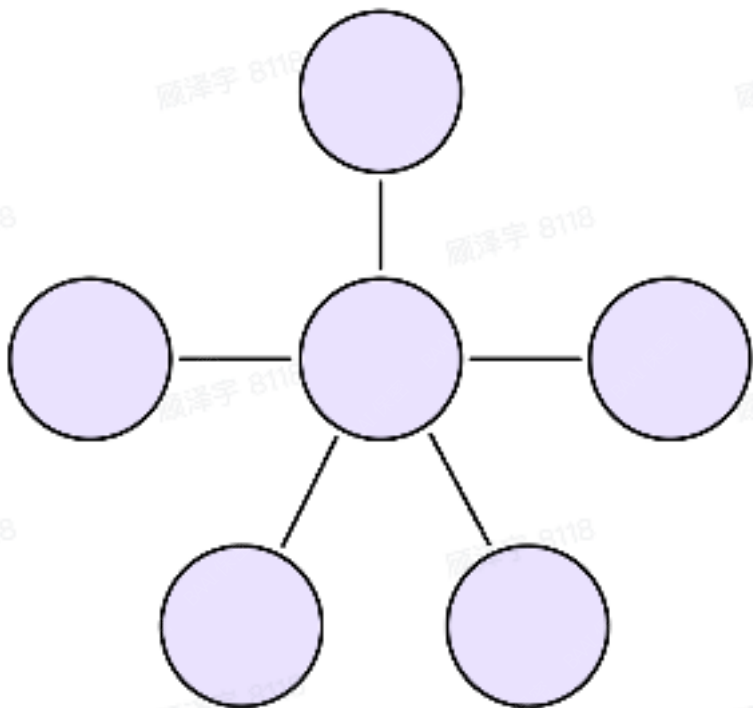
### 优点:

- 层次结构清晰: 有明显层级关系, 适用于大规模的组织结构
- 易于扩展: 新增节点只需连接到树的任意位置, 不影响整体结构
- 故障隔离: 故障只会影响到某一层级, 易于定位和隔离问题

### 缺点:

- 瓶颈问题: 根节点带宽和性能决定了整个网络吞吐量, 易成为瓶颈
- 单点故障: 根节点或某个重要节点故障可能影响整个网络

## 星型拓扑



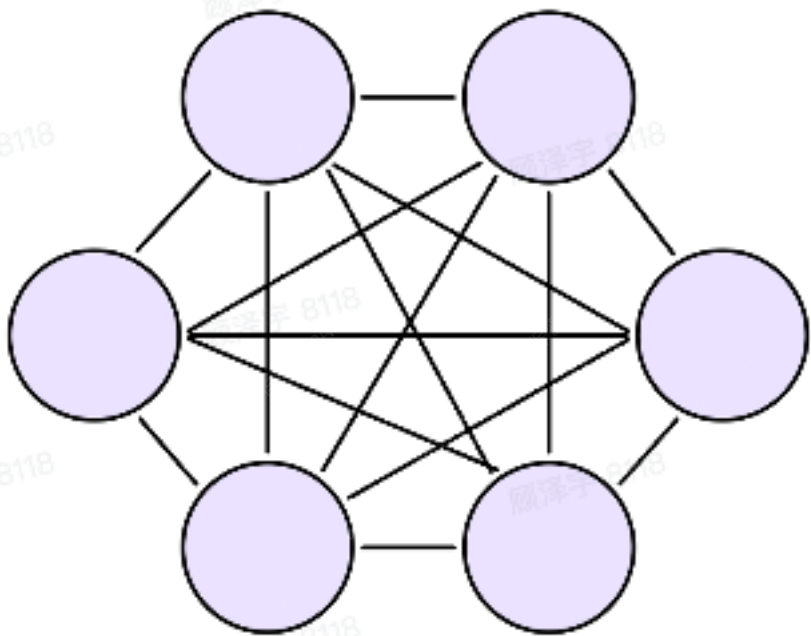
### 优点:

- 易于管理和扩展：所有设备通过中心节点连接，易于添加新设备
- 故障隔离：设备之间通信只经过中心节点，非中心节点故障不会影响其他设备通信
- 集中控制：中心节点集中管理网络流量，便于管理和监控

### 缺点:

- 中心节点瓶颈：所有流量经过中心节点，中心节点性能和带宽成为整个网络的瓶颈
- 单点故障：中心节点故障，整个网络无法工作

## 网状拓扑



### 优点:

- 可靠性高: 每个节点与其他节点都有连接, 某条链路故障, 也有其他路径可选
- 高带宽: 由于冗余路径存在, 能够动态选择最佳路径进行传输

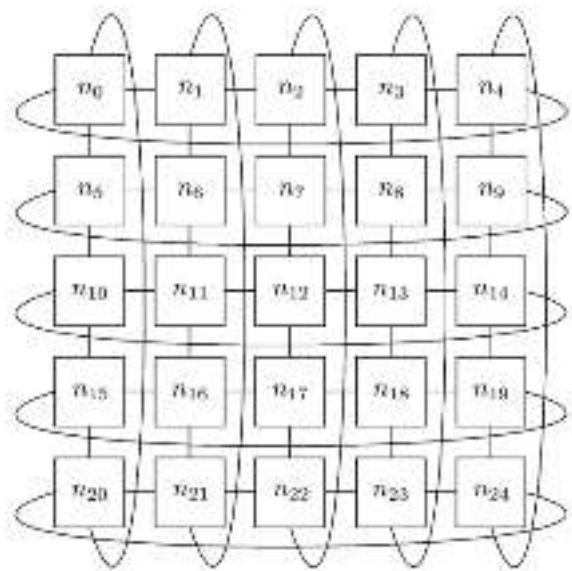
### 缺点:

- 高成本: 每个节点都需要和其他节点连接, 布线成本高
- 难管理: 随着节点数增加, 链路管理和维护变得复杂

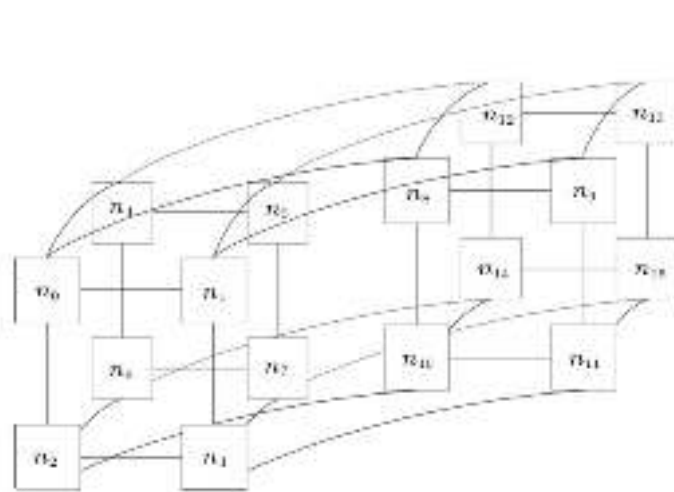
## 拓扑感知对通信优化的重要性

- 优化数据路径
- 提高带宽利用率
- 增强容错性

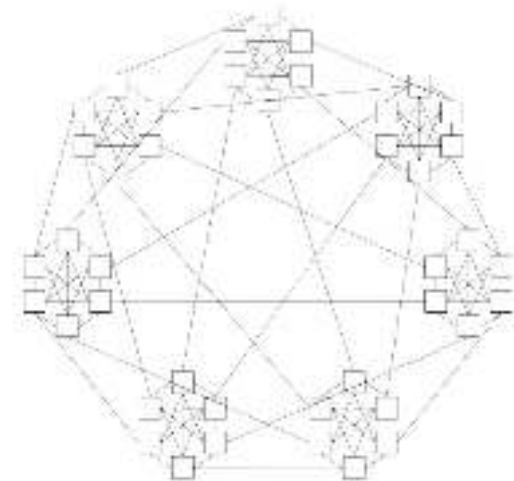
## 实际集群中的复杂拓扑结构



2D Torus



Hyper cube



dragonfly

# 目录

- 1 网络拓扑基础
- 2 FlagCX拓扑感知实现
- 3 拓扑感知通信优化实践

## 如何表达拓扑结构

- 适用于多种硬件平台
- 简单直接的表达设备间的连接关系

## FlagCX拓扑描述

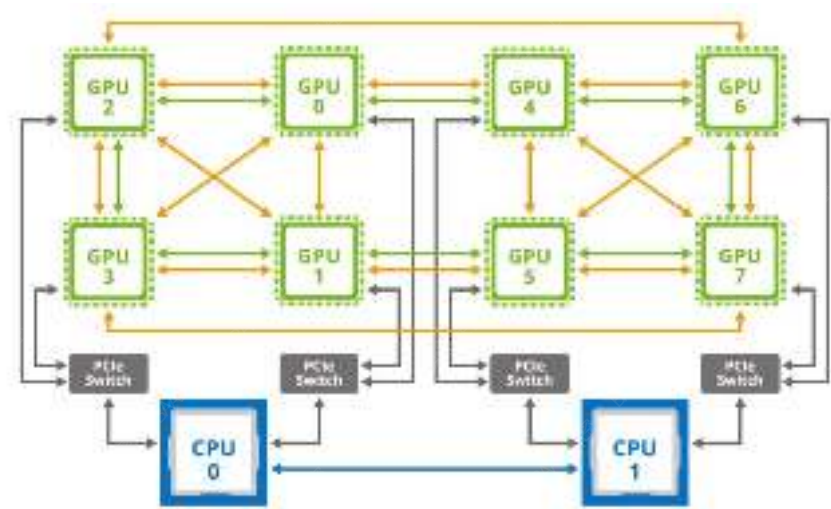
- 通过设备+链路双层抽象，对主流拓扑结构提供统一描述

### 物理设备抽象

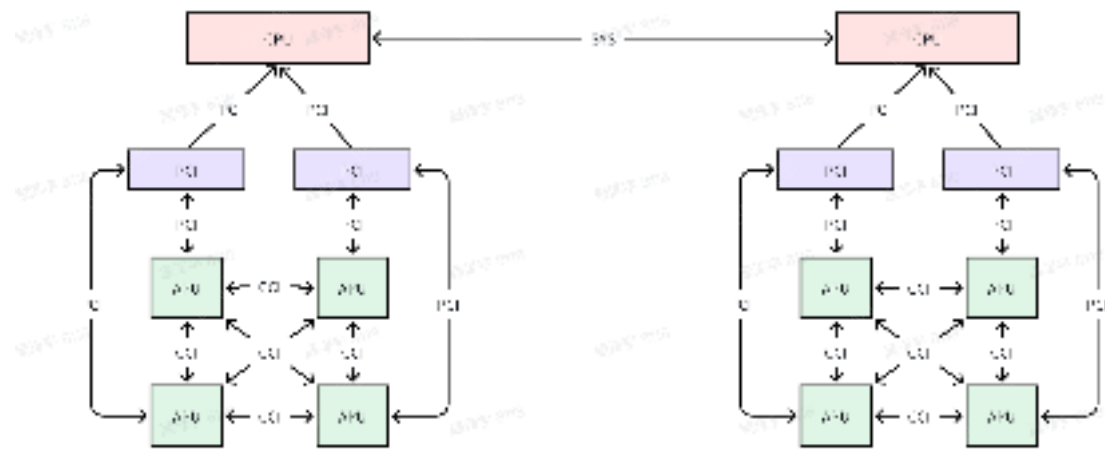
- APU：加速器设备
- PCI：PCI switch
- HBD：high-bandwidth domain (i.e. nvswitch)
- CPU：NUMA节点
- NET：网卡设备

### 互联链路抽象

- LINK\_LOC：本地链路(self)
- LINK\_CCI：cache-coherent interconnect (i.e. nvlink)
- LINK\_PCI：PCI链路
- LINK\_SYS：NUMA节点间链路
- LINK\_NET：网络链路 (i.e. RDMA)



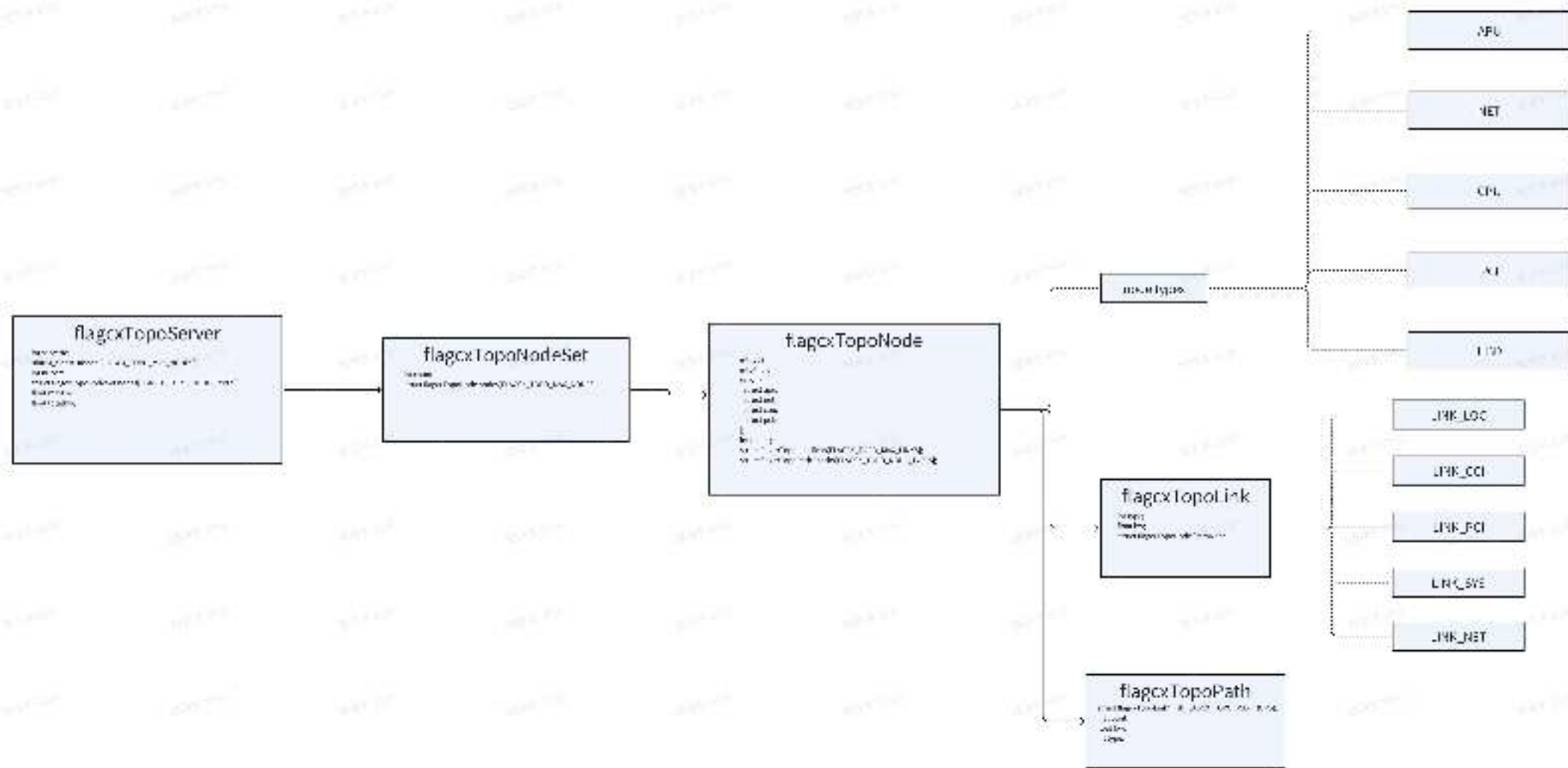
实际拓扑



抽象后拓扑

图片引用: <https://developer.nvidia.com/blog/dgx-1-fastest-deep-learning-system/>

## Server拓扑数据结构



## Server内拓扑探测

核心实现方法：

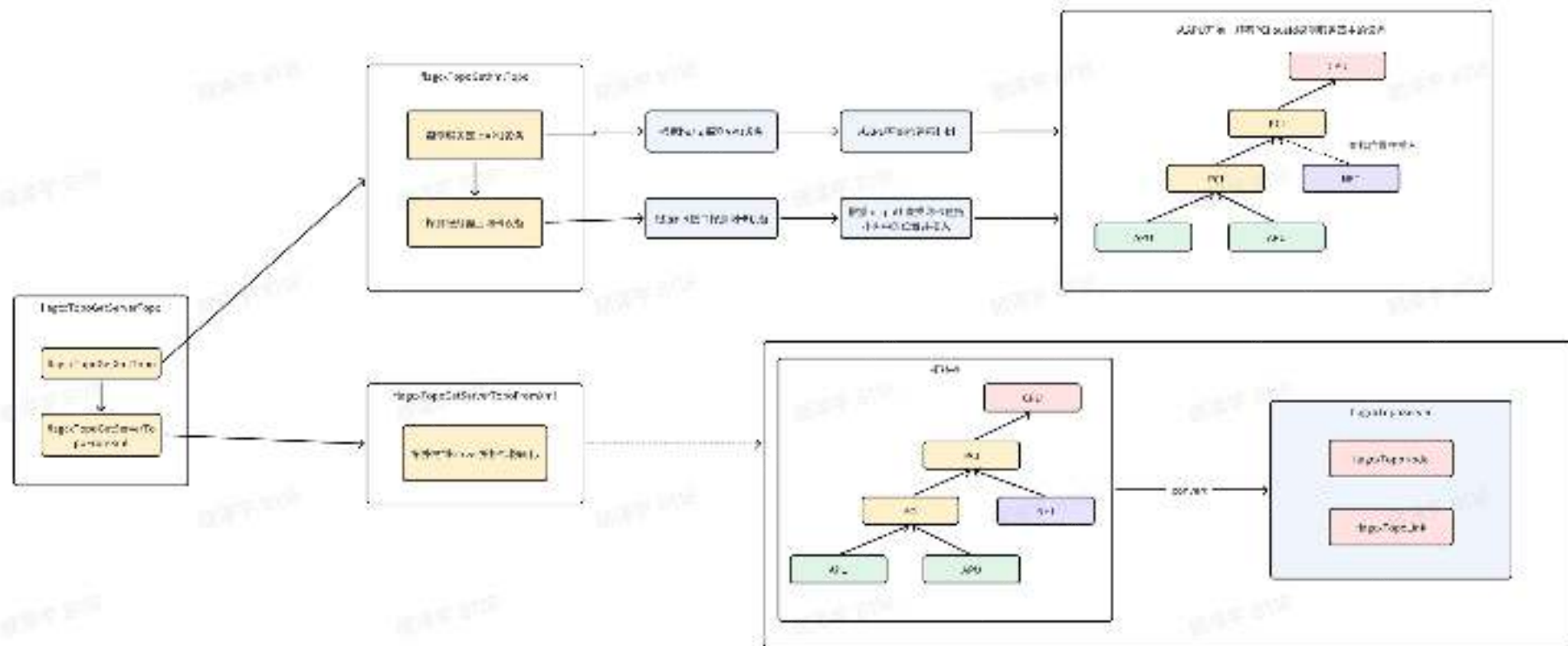
通过设备pci busid以及pci path获取硬件在系统中的位置及连接关系

Example:

Pci busid: e.g. 0000:03:00.0

Pci path: e.g.

/sys/devices/pci0000:00/0000:00:01.0/0000:03:00.0



- 遍历所有rank，找到属于同一host的rank
- 获取每个rank的busId
- 调用flagcxTopoFillApu，获取设备信息并把设备节点添加到xml结构中

```
// will remove this function when we finish the function that builds server topo
flagcxResult_t flagcxTopoGetXmlTopo(struct flagcxHeteroComm *comm,
                                   struct flagcxXml *xml) {
    // create root node if we didn't get topo from xml file
    if (xml->maxIndex == 0) {
        INFO(FLAGCX_INIT, "creating root XML node");
        // Create top tag
        struct flagcxXmlNode *top;
        // TODO: change root node name from "system" to "root"
        FLAGCXCHECK(xmlAddNode(xml, NULL, "system", &top));
        FLAGCXCHECK(xmlSetAttrInt(top, "version", FLAGCX_TOPO_XML_VERSION));
    }

    INFO(FLAGCX_INIT, "start detecting APUs");
    for (int r = 0; r < comm->nRanks; r++) {
        if (comm->peerInfo[r].hostHash == comm->peerInfo[comm->rank].hostHash) {
            INFO(FLAGCX_INIT, "preparing to detect APU for rank %d", r);
            char busId[FLAGCX_DEVICE_PCI_BUSID_BUFFER_SIZE];
            INFO(FLAGCX_INIT, "converting busId to string");
            FLAGCXCHECK(int64ToBusId(comm->peerInfo[r].busId, busId));
            struct flagcxXmlNode *node;
            FLAGCXCHECK(flagcxTopoFillApu(xml, busId, &node));
            if (node == NULL) {
                continue;
            }
            int devLogicalIdx = 0;
            deviceAdaptor->getDeviceByPciBusId(&devLogicalIdx, busId);
            FLAGCXCHECK(xmlSetAttrInt(node, "dev", devLogicalIdx));
            FLAGCXCHECK(xmlSetAttrInt(node, "rank", r));
        }
    }
}
```

- 创建一个PCI父节点
- 通过PCI父节点向上遍历PCI树，构建xml拓扑树
- 在xml拓扑树中添加一个APU节点

```
flagcxResult_t flagcxTopoFillApu(struct flagcxXml *xml, const char *busId,  
                                struct flagcxXmlNode **gpuNode) {  
    struct flagcxXmlNode *pciNode;  
    INFO(FLAGCX_INIT, "creating xml pci node for busId [%s]", busId);  
    FLAGCXCHECK(flagcxTopoGetPciNode(xml, busId, &pciNode));  
    FLAGCXCHECK(flagcxTopoGetXmlFromSys(pciNode, xml));  
    INFO(FLAGCX_INIT, "creating xml apu node for busId [%s]", busId);  
    FLAGCXCHECK(flagcxTopoGetXmlFromApu(pciNode, xml, gpuNode));  
    return flagcxSuccess;  
}
```

flagcxTopoGetXmlFromSys →

- 解析PCI path, 逐层构建拓扑树
- 遍历到CPU节点停止
- 如果为非CPU节点, 添加PCI节点, 并继续向上遍历

```
struct flagcxXmlNode *parent = pciNode->parent;
if (parent == NULL) {
    // try to find the parent along the pci path
    if (path) {
        // Save that for later in case next step is a CPU
        char nuidStr[MAX_STR_LEN];
        FLAGCHECK(flagcxTopoGetStrFromSys(path, "nuid", nuidStr));

        // Go up one level in the PCI tree. Beware for "/" and follow the upper
        // PCI switch, or stop if we reach a CPU root complex.
        int slashCount = 0;
        int parentOffset = 0;

        for (parentOffset = strlen(path) - 1; parentOffset > 0; parentOffset--) {
            if (path[parentOffset] == '/') {
                slashCount++;
                path[parentOffset] = '\0';
                int start = parentOffset - 1;
                while (start > 0 && path[start] != '/')
                    start--;

                // Check whether the parent path looks like "0000:00:00.0" unit.
                if (checkBDFFormat(path + start + 1) == 0) {
                    // This is a CPU root complex. Create a CPU tag and stop there.
                    struct flagcxXmlNode *topNode;
                    FLAGCHECK(xmlFindTop(xml, "system", &topNode));
                    FLAGCHECK(
                        xmlGetSubKv(topNode, "cpu", &parent, "nuid", nuidStr));
                    if (parent == NULL) {
                        FLAGCHECK(xmlAddNode(xml, topNode, "cpu", &parent));
                        FLAGCHECK(xmlSetAttr(parent, "host_bash", getHostHash()));
                        FLAGCHECK(xmlSetAttr(parent, "nuid", nuidStr));
                    }
                } else if (slashCount == 2) {
                    // Continue on the upper PCI switch
                    for (int i = strlen(path) - 1; i > 0; i--) {
                        if (path[i] == '/') {
                            FLAGCHECK(
                                xmlFindTagKv(xml, "pci", &parent, "busid", path + i + 1));
                            if (parent == NULL) {
                                FLAGCHECK(xmlAddNode(xml, NULL, "pci", &parent));
                                FLAGCHECK(xmlSetAttr(parent, "busid", path + i + 1));
                            }
                            break;
                        }
                    }
                }
            }
        }
    }
    if (parent)
        break;
}
```

flagcxTopoGetXmlTopo

- 遍历网卡设备
- 调用flagcxTopoFillNet添加网卡设备到拓扑树中

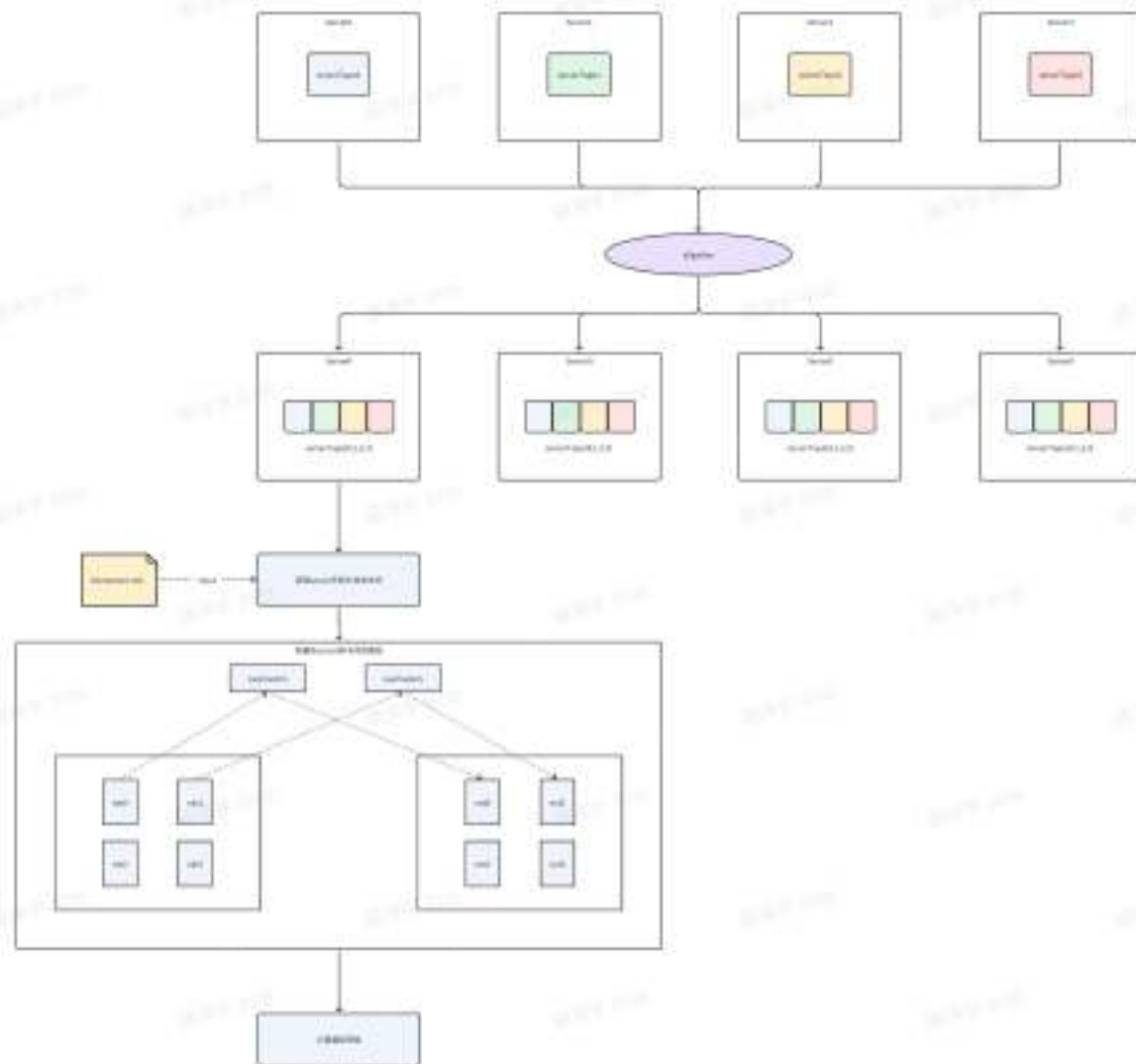
```
int netDevCount = 0;
FLAGCXCHECK(flagcxNetIb.devices(&netDevCount));
for (int n = 0; n < netDevCount; n++) {
    flagcxNetProperties_t props;
    FLAGCXCHECK(flagcxNetIb.getProperties(n, &props));
    struct flagcxXmlNode *netNode;
    FLAGCXCHECK(flagcxTopoFillNet(xml, props.pciPath, props.name, &netNode));
    FLAGCXCHECK(xmlSetAttrInt(netNode, "dev", n));
    FLAGCXCHECK(xmlSetAttrInt(netNode, "speed", props.speed));
    FLAGCXCHECK(xmlSetAttrFloat(netNode, "latency", props.latency));
    FLAGCXCHECK(xmlSetAttrInt(netNode, "port", props.port));
    FLAGCXCHECK(xmlInitAttrUInt64(netNode, "guid", props.guid));
    FLAGCXCHECK(xmlSetAttrInt(netNode, "maxConn", props.maxComms));
}
```

## 从server拓扑扩展到cluster拓扑

- 记录server间网卡路径及带宽
- 记录集群server数量

通过server间拓扑描述文件构建server间路径

```
<interserver_route>
  <nic_pairs>
    <pair>
      <nic1 guid="0xf6e65d0083d23fb8" />
      <nic2 guid="0xae85d0083d23fb8" />
      <interSwitch count="1">
        <switch downBw="25" upBw="25" upLink="1" downLink="1" isTop="1"/>
      </interSwitch>
    </pair>
  </nic_pairs>
</interserver_route>
```



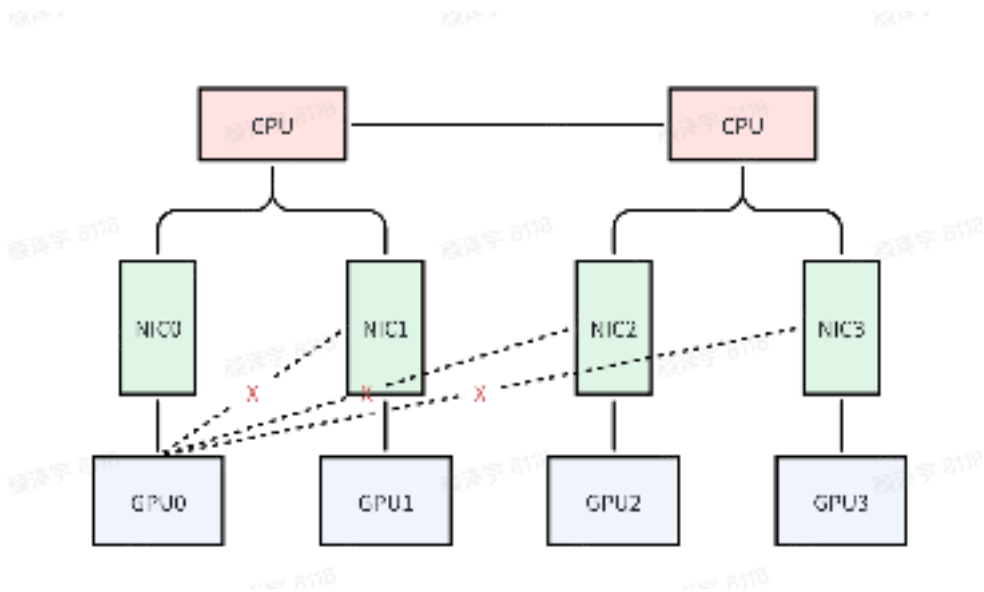
# 目录

- 1 网络拓扑基础
- 2 FlagCX拓扑感知实现
- 3 拓扑感知通信优化实践

## 拓扑感知在FlagCX中的应用

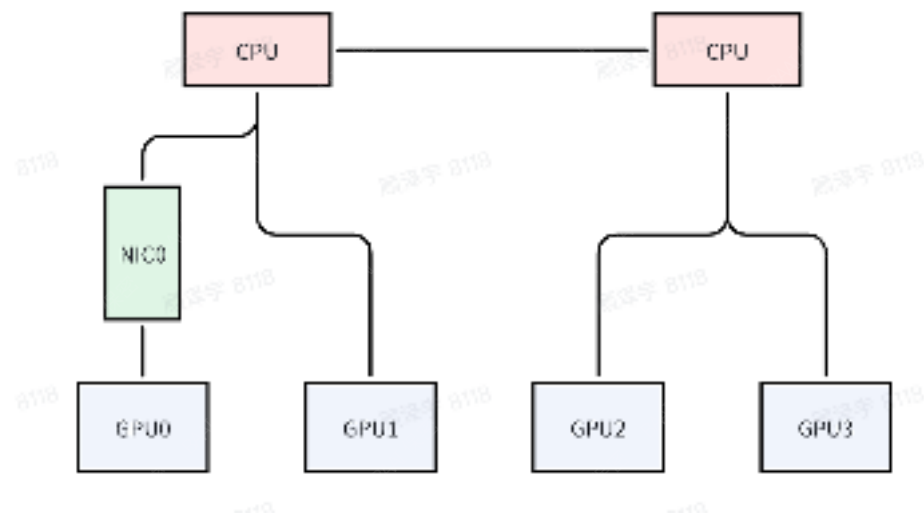
- 选择跨芯通信device以及最优通信网卡
- 保障C2C跨芯算法性能

Ex.1



GPU0选择使用NIC0，而非其他距离较远的NIC

Ex.2



只使用GPU0进行跨芯通信可能比使用所有GPU更高效

## FlagCX拓扑 + 算法协同优化

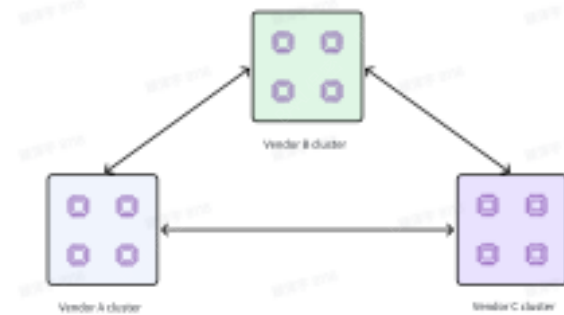
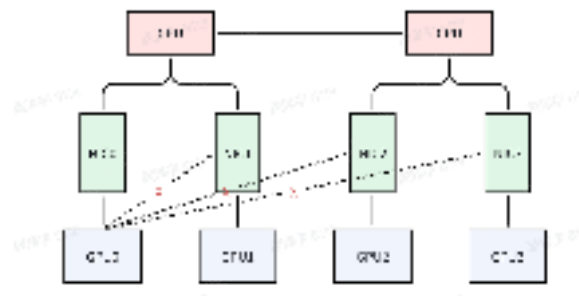
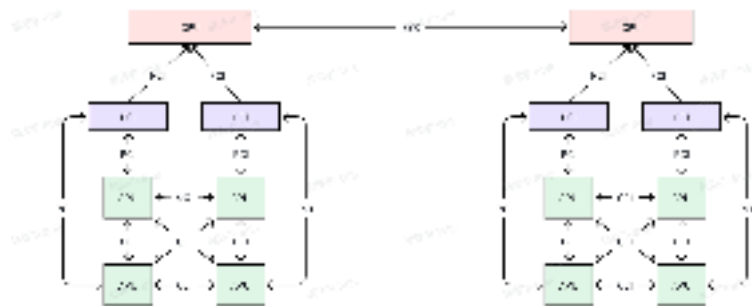
Topology Detection



NIC & inter-rank selection



C2C Algo



## 拓扑感知在NCCL中的应用 → NCCL channel搜索

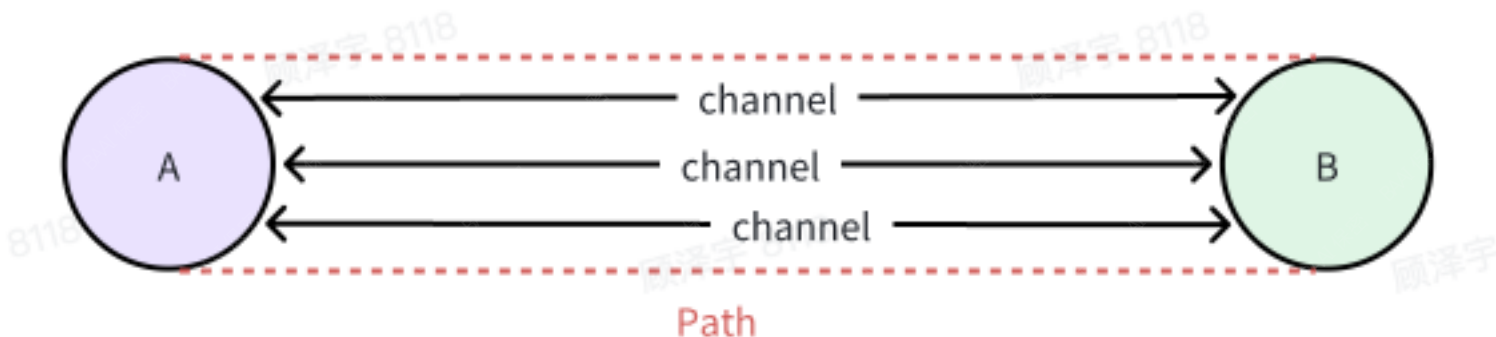
- 基于拓扑信息构建最优通信路径
- 构建逻辑拓扑

## 什么是channel?

- NCCL中，channel代表一个通信“信道”，用于在设备间传输数据

## channel和path的区别

- Path是物理路径，channel是通过这个路径进行通信的通信流



## NCCL channel搜索流程

- 步骤1: 初始化graph
- 步骤2: 初始化搜索条件
- 步骤3: 第一次搜索（逐步降低搜索条件）
- 步骤4: 第二次搜索（逐步提升搜索条件）
- 步骤5: 比较channel

## channel搜索核心目标

- 带宽优化：最大化数据吞吐量，最小化延迟
- 路径优化：找到最佳通信路径，根据拓扑动态调整通信策略

## channel搜索核心函数

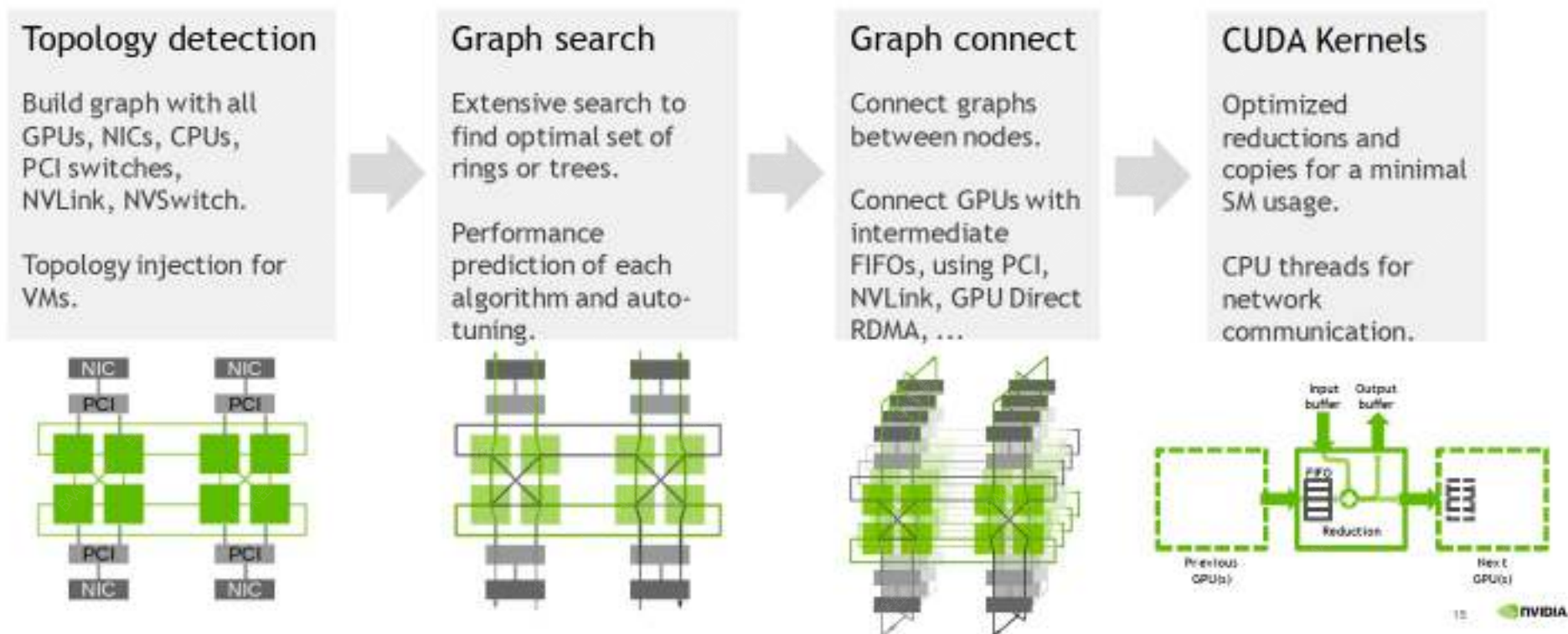
ncclTopoSearchRec:

- 递归函数，用于channel搜索，每次调用会进行一次新的搜索

ncclTopoCompareGraphs:

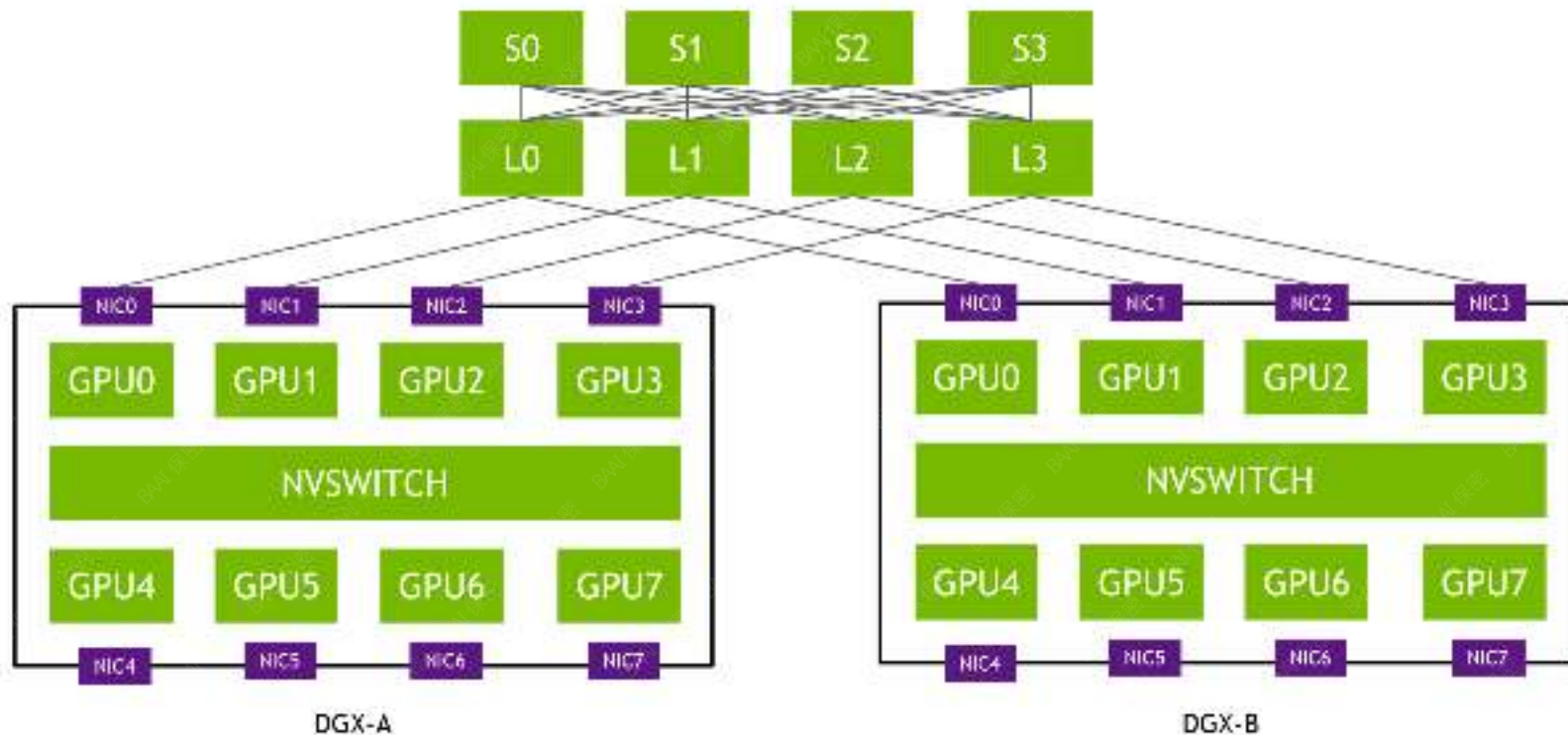
- 比较当前搜索结果，判断新找到的channel是否比已有的更优

## NCCL拓扑 + 算法协同优化



## NCCL拓扑 + 算法协同优化：多轨通信 + PXN

- PXN(PCI x NVLink) 利用节点内 GPU 之间的 NVIDIA NVSwitch 连接，首先将 GPU 上的数据移动到与目的地相同的轨道上，然后在不跨越轨道的情况下将其发送到目的地。这可以实现消息聚合和网络流量优化。

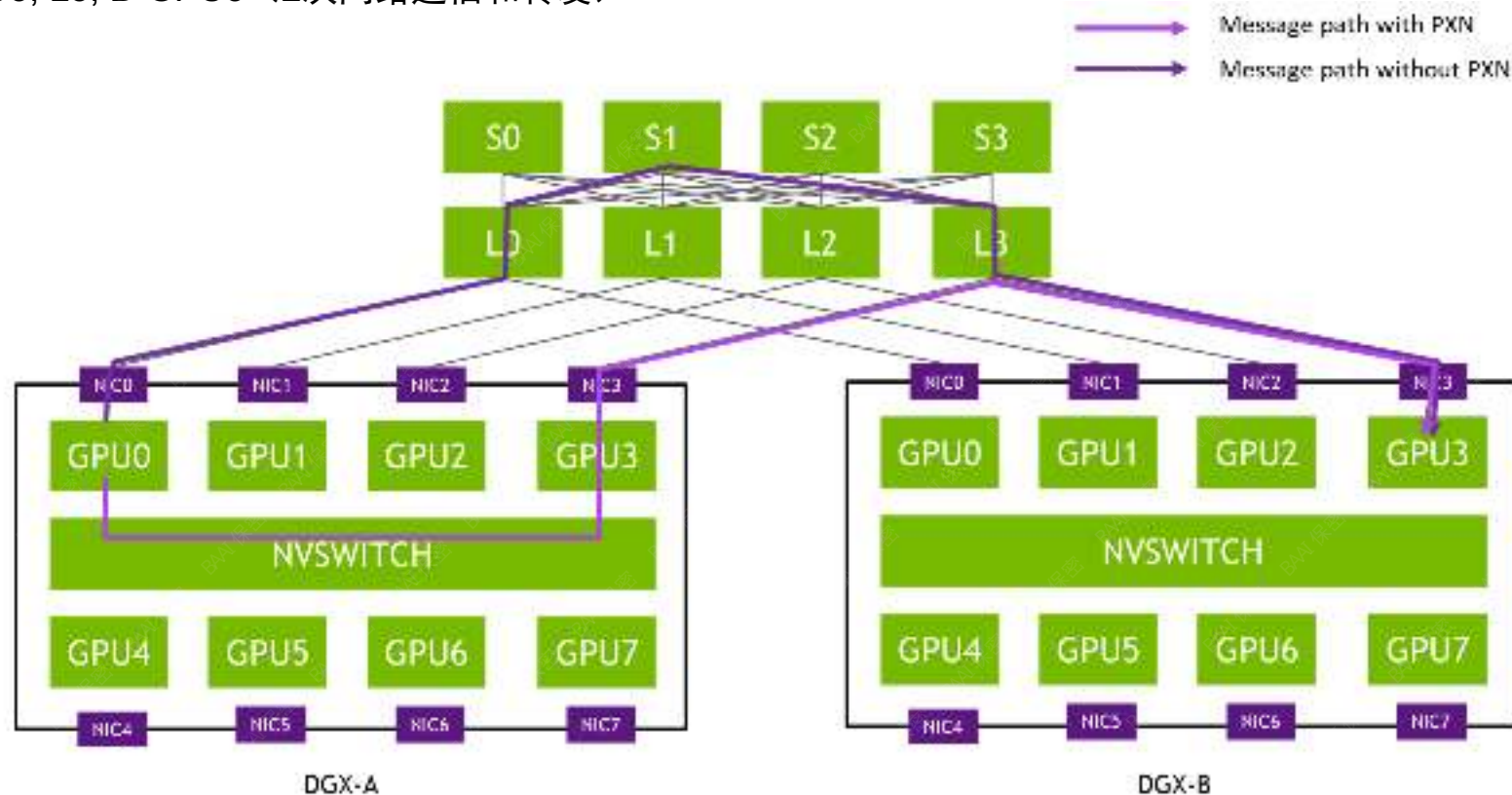


图片引用: <https://developer.nvidia.com/zh-cn/blog/doubling-all2all-performance-with-nvidia-collective-communication-library-2-12/>

## NCCL拓扑 + 算法协同优化：多轨通信 + PXN

正常：A-GPU0, L0, S1, L3, B-GPU3 (4次网络通信和转发)

PXN: A-GPU0, A-GPU3, L3, B-GPU3 (2次网络通信和转发)



图片引用: <https://developer.nvidia.com/zh-cn/blog/doubling-all2all-performance-with-nvidia-collective-communication-library-2-12/>

Message path with PXN  
Message path without PXN

## 拓扑感知优化方向

- 动态拓扑感知
  - 大型集群中实时更新拓扑信息并进行优化
- 多层次拓扑建模
  - 将机内，机间，数据中心间的多层拓扑结合起来进行全局优化
- 异构平台拓扑优化
  - 针对不同类型加速器的拓扑感知优化
- 基于拓扑感知的通信代价建模
  - 通过代价建模选择最优通信策略



## 联系我们

邮件: [job@baai.ac.cn](mailto:job@baai.ac.cn)

电话: 010 - 6893 3383

地址: 北京市海淀区成府路150号智源大厦



智源公众号