

## DPU新范式: 网络大坝和可编程存内计算 (<https://lifeve.cn/Cr4ULdwoQkfpyq.html>)

🕒 2021-11-05 02:06:47 👁 210 👤 zartbot (/tags/zartbot.html)

这项工作主要是由思科中国研发中心两位同事Kevin Fang和David Peng一起完成，文章中我们对比了主机内各种通信总线(PCIe/CXL/CHI/AXI)和主机之间的通信协议(以太网、RDMA),得出结论需要在网络侧直接添加内存，并提供可编程的指令集实现SIMD访问和计算加速，这种做法对硬件和软件都非常友好，测试结果显示NetDAM网卡平均读延迟仅 618ns，抖动 39ns，远低于当前的RDMA实现。在分布式AI训练场景中，MPI-Allreduce任务也比HPC-X快数倍。

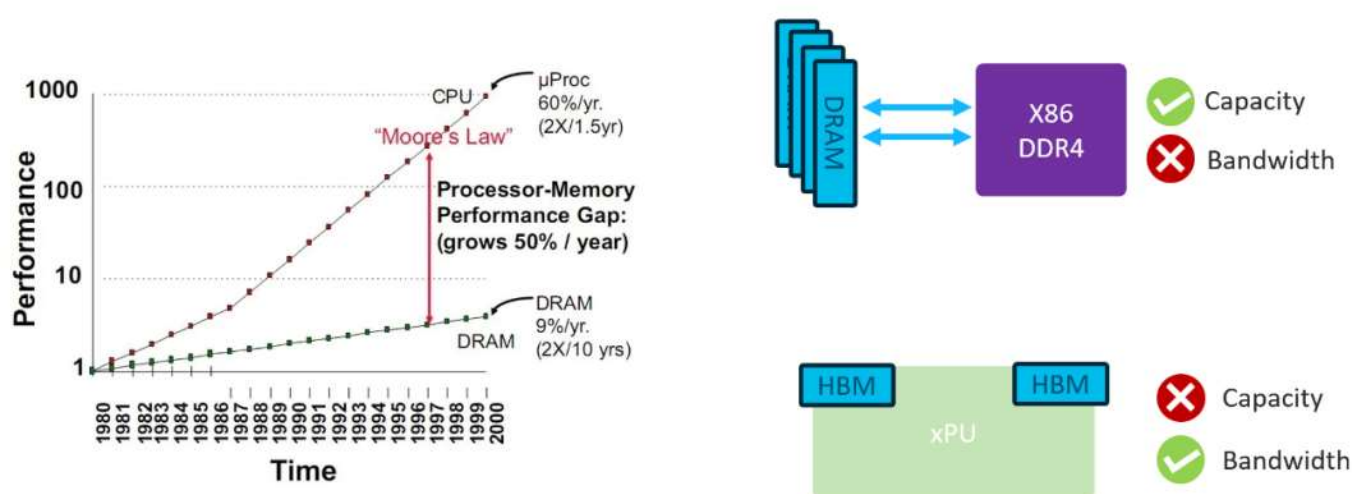
论文由于暂时没有arxiv的权限，暂时放在了Github上: <https://github.com/NetDAM/doc>

也麻烦能够在arxiv.org的endorse的读者帮忙点一下 <https://arxiv.org/auth/endorse?x=8FVCAJ>

### 概述

通常数据密集型应用(例如大型分布式数据库、分布式AI训练集群)需要数Tbps的带宽和数TBytes的容量

### Memory Wall



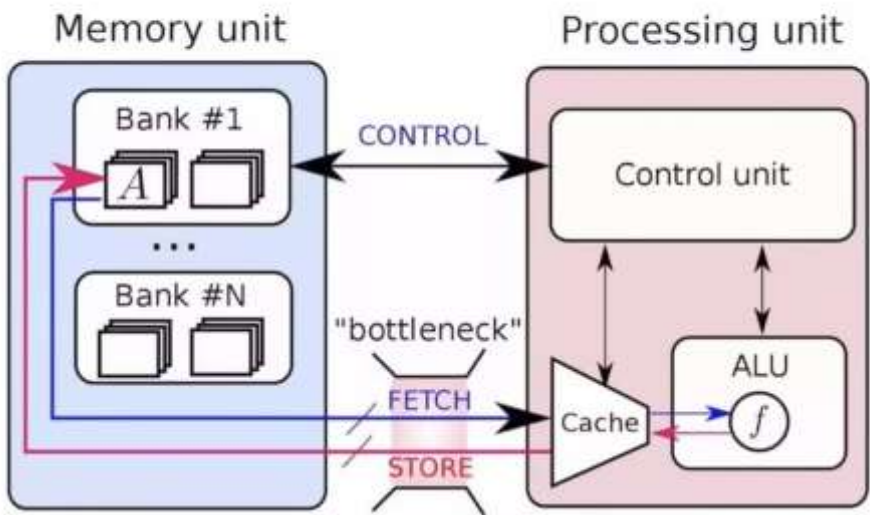
分布式内存池化和存内计算(in-memory Computing)及在网计算(in-network Computing)逐渐被工业界接受。我们实现了一种直接将内存直接挂载在以太网控制器并提供大量的ALU和可编程逻辑的方法，这种方法将成为一种高效的内存池化、存内计算及在网计算的新范式。

文章中我们利用NetDAM架构实现了一个基于FPGA的MPI Allreduce通信场景，证明了其架构上可编程指令集内存访问(Programmable ISA)针对特殊领域优化的优越性。

## NetDAM简介

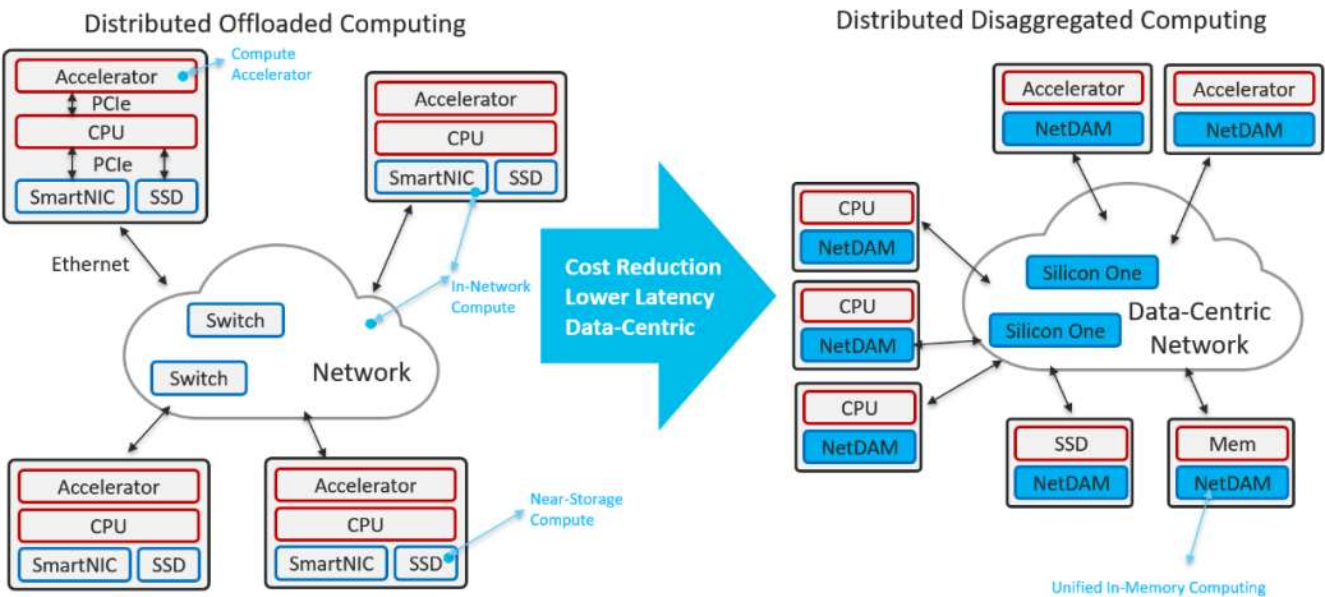
传统的冯诺依曼架构中，计算单元和存储单元是分离的，因此大量的数据流动产生了内存墙和冯诺依曼瓶颈

### The Conventional von Neumann Architecture



基于特殊领域的集成电路(DSA)被提出，大量的Offload卡被开发出来，但是这些卡通信同样需要超高带宽和超低延迟的通信总线，DPU开始关注并解决这个问题，但与此同时这样的Offload架构还增加了大量的成本，因此这是我们构建NetDAM的一个原因：

### New Trend towards Disaggregated Computing



## 主机内总线和主机间通信特征

PCIe作为主机内(Intra-Host)各扩展卡和CPU通信的标准已经存在了接近20年，基于PCIe的直接内存访问DMA也被广泛的用于芯片间的通信。RDMA over Ethernet简单的将DMA操作扩展到了主机间(Inter-Host)通信网络。但是go-back-N的策略对丢包非常敏感，因此DCQCN这一类基于PFC的可靠传输和拥塞控制机制被开发出来，但是随着网络规模增大及VPC等Overlay网络架构的出现，这样的架构将会带来巨大的延迟和抖动。因此像Fungible一样的厂商逐渐开始实现基于硬件TCP卸载的iWARP技术，与此同时阿里巴巴的HPCC和Intel的NDP也被开发出来用于消除PFC带来的影响。但与此同时，通过一些研究发现，PCIe本身由于RootComplex的设计和驱动的问题，也会带来巨大的延迟，因此GenZ、CCIX、CXL等总线被开发出来用于解决这些问题。

但是我们重新审视了主机内(Intra-Host)和主机间(Inter-Host)通信协议，主机内通信由于延迟可控丢包可控通常采用共享内存(share-memory)的模式，而主机间通信则通常采用消息传递(MPI)的方式，因此两者在设计原则上根本性的不同：

- 拓扑：主机内通信协议通常是有固定的树状拓扑的，并且设备编址和寻址相对固定(例如PCIe使用的DFS),消息路由相对简单。而主机间通信协议通常是非固定的并且有多路径支持和Overlay支持会使得报文调度更加复杂。当然有一些片上网络总线例如AMBA CHI可以实现多跳通信，这也是我们为什么在处理器架构中推荐基于ARM Neoverse2的Octeon CN10k的原因。但是CHI总线更多的用于片上网络设计，对于跨芯片传输和跨主机有丢包和延迟的以太网传输则不适合。
- 延迟：主机内通信协议通常只有小于200ns的固定传输延迟，而主机间以太网通常为数个微秒的延迟，并由于包调度和多路径及拥塞控制等原因会带来不确定性。
- 丢包：主机内通信通常由于仲裁器和Credit Token调度通常不会出现丢包，但是在主机间通信经常由于拥塞或者中间节点失效导致丢包，实现不丢包的以太网代价巨大并且成本过高而且网络利用率和复用率较低。
- 一致性：在主机内通信由于往返延迟非常低，因此通常采用基于MESI一类协议的缓存一致性协议实现共享内存的通信。而在主机间高延迟的情况下实现一致性会非常困难，也带来了编程模式的挑战。(注：可以参考OpenMP和OpenMPI在超算中的优劣。)
- 保序：通常主机内通信为了内存一致性是需要严格保序的，从物理实现上也相对容易，虽然PCIe也支持Relax Order但是用处并不是很大。而主机间通信由于多路径和一些网络安全设备调度的因素乱序时常发生。
- 传输报文大小：由于主机内通信实时性、低延迟和一致性的需求下，通常一个flit不会放的太大，大多数协议都最大维持到一个CacheLine(64B)的大小，再大会影响其它设备的实时通信，而且很多协议对于ACK、NACK有严格的时序约束，而以太网通常是1500B甚至9000B<sup>^</sup>的传输。

## RDMA:直接扩展主机内总线

RDMA实际上是一种直接将主机内DMA映射到以太网或者IB上的处理机制，同样继承了PCIe已有的go-back-N和QueuePair Credit机制，但是由于主机侧PCIe的原因会带来一些Cache操作的复杂性和总线争抢的不确定性，因此尾部延迟会非常高。同样在以太网上由于Lossless的需求使得交换机和各种控制协议设计会变得复杂并进一步加大了延迟。

## NanoPU:直接扩展主机间总线

于是NanoPU走了另一个极端，直接把主机间的总线怼到CPU上，然后通过寄存器级别的操作实现了整个网络协议栈的硬化处理。但这样的架构在过去数年中我们看到大量的网络处理器厂商采用，针对复杂的应用，特别是需要大量Branch和不确定性处理Cycles的应用(100~10K cycles)时，这样的架构性能会出现极大的问题。他们自己也意识到了这个问题，于是在最后的结论中提到：

### nanoPU Conclusions

#### Key Takeaway:

To truly minimize median and tail RPC latency and SW overheads:

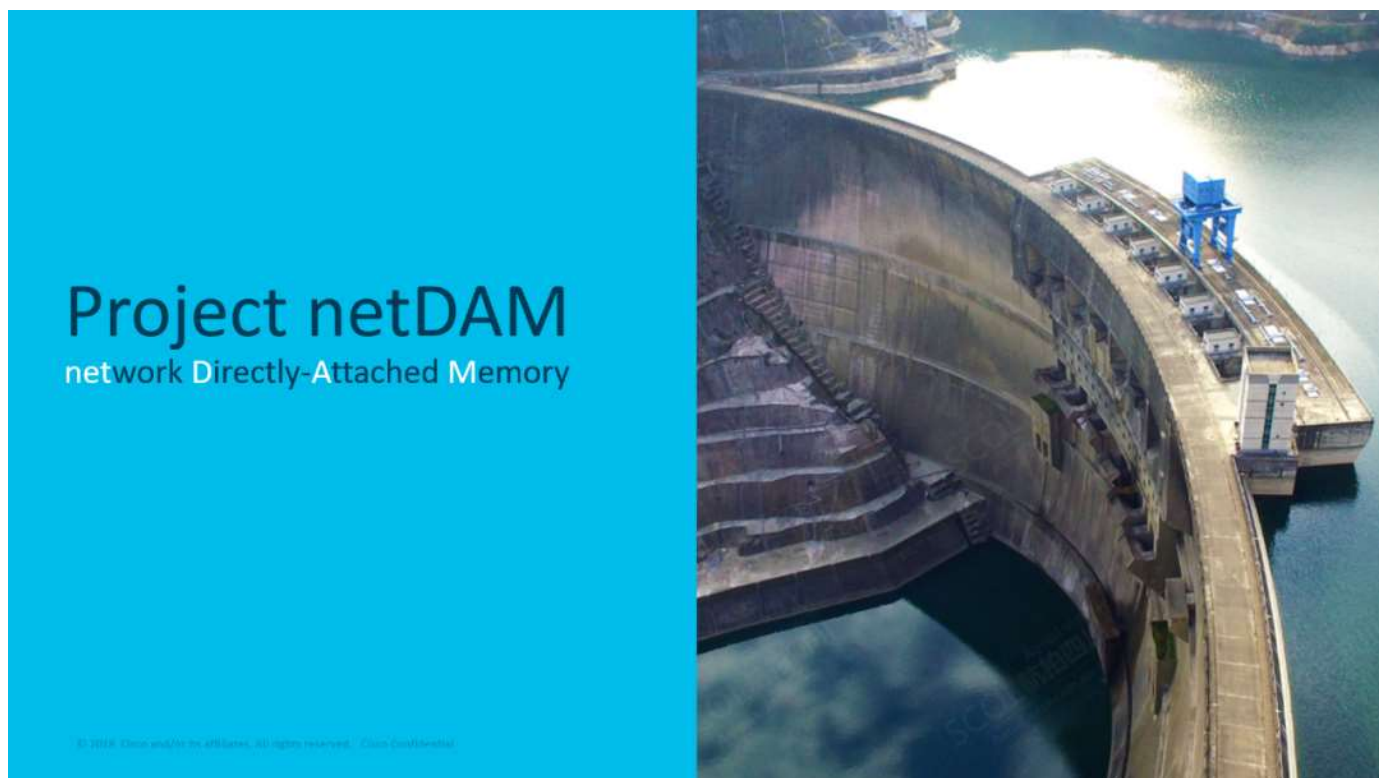
1. Fast path directly between network and CPU register file
2. Move entire network stack to HW: **transport, load balancing, thread scheduling**

#### Challenges:

- Need to rewrite applications
- Figure out how to use more sophisticated processors

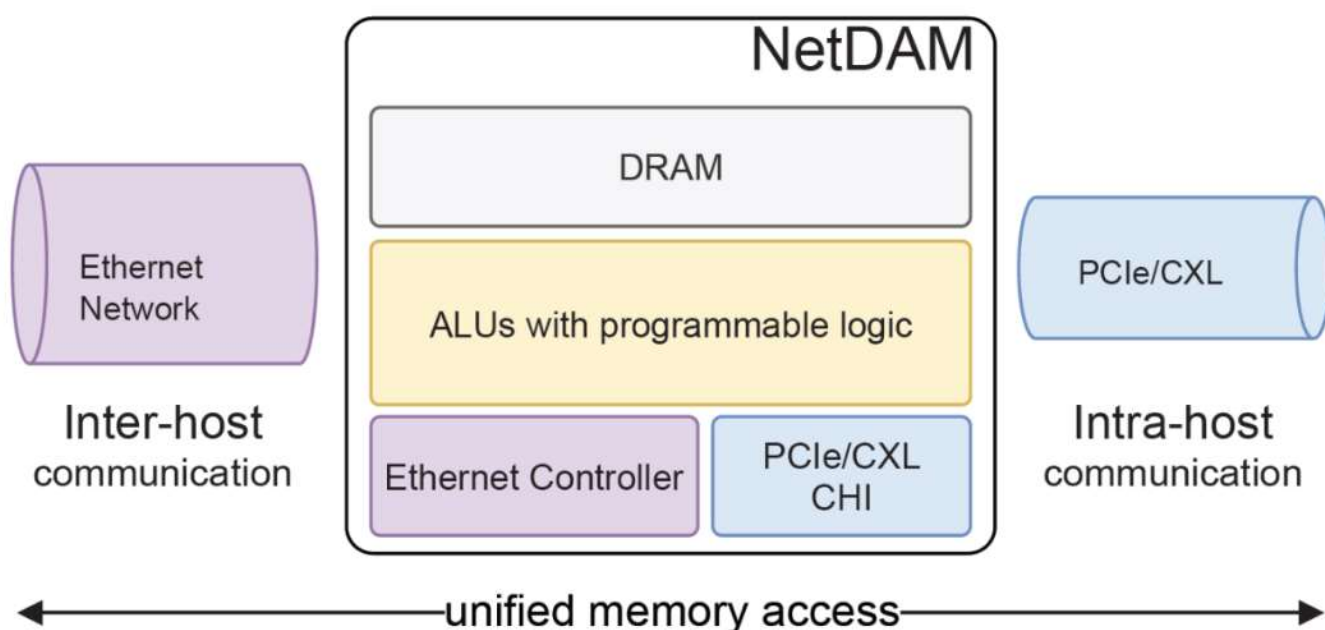
通过分析两种网络设计的方案，形象的说，我们需要在主机的边缘侧构建一个大坝，主机内部通常如同峡谷流速高，并且需要根据处理器的能力按需调度，而主机间通信则经常出现洪峰等难以预见的情况：





## NetDAM：筑坝桥接

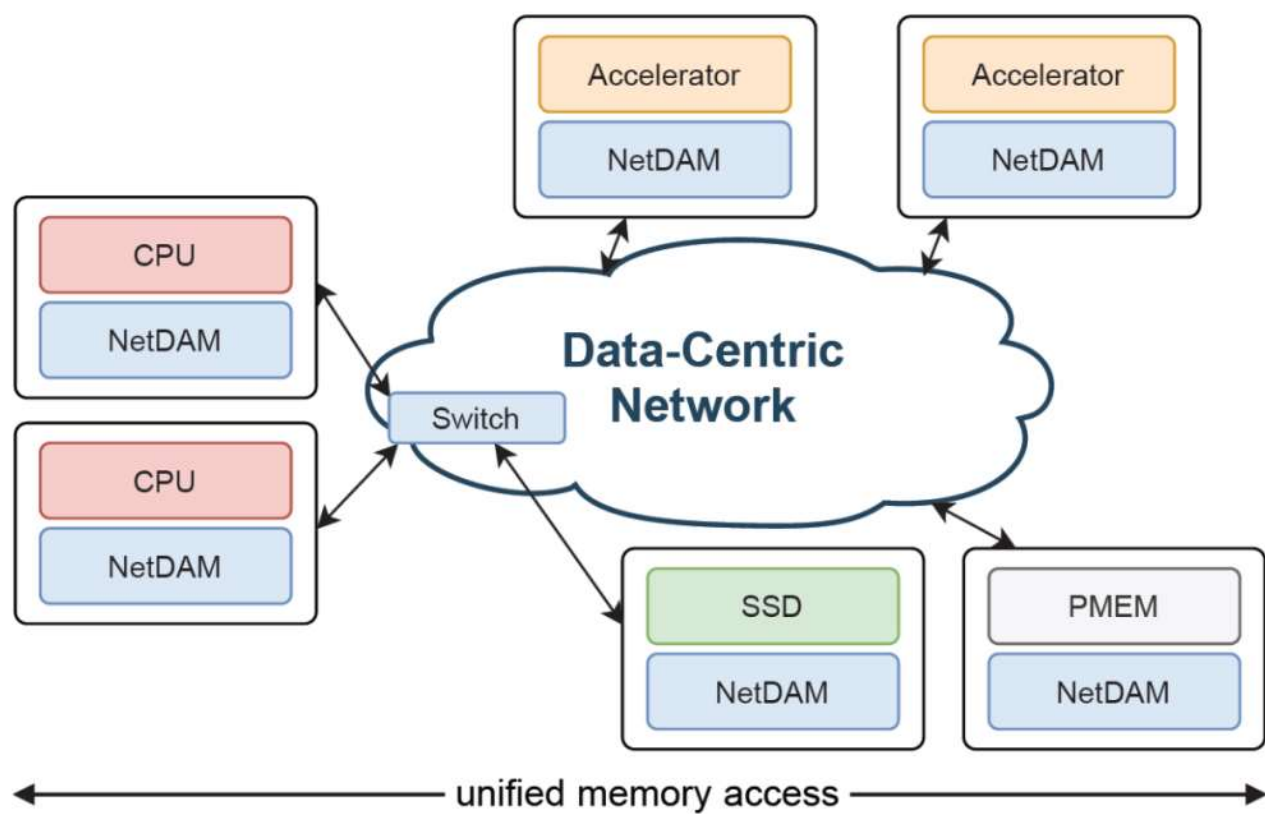
因此Directly-Attached Memory正好成了大坝来对不同特性的网络进行桥接和调度，类似于OpenMP和OpenMPI混合使用的方式，在主机间使用消息传递(MPI)的方式采用JumboFrame大批量的调度的调度流量来隐藏延迟，而在主机内通过PCIe或者CXL让处理器自行决定是否加载和进行一致性处理：



NetDAM架构如上图所示，我们将内存加在了网卡上，并定义了可扩展的指令集和大量的可编程逻辑用于近存计算和在网计算，当然这个架构以后也可以扩展给Samsung的Process-In-Memory HBM使用，我们可以直接将指令发送到HBM内让内存自己执行一些操作。因此主机内通信接口可

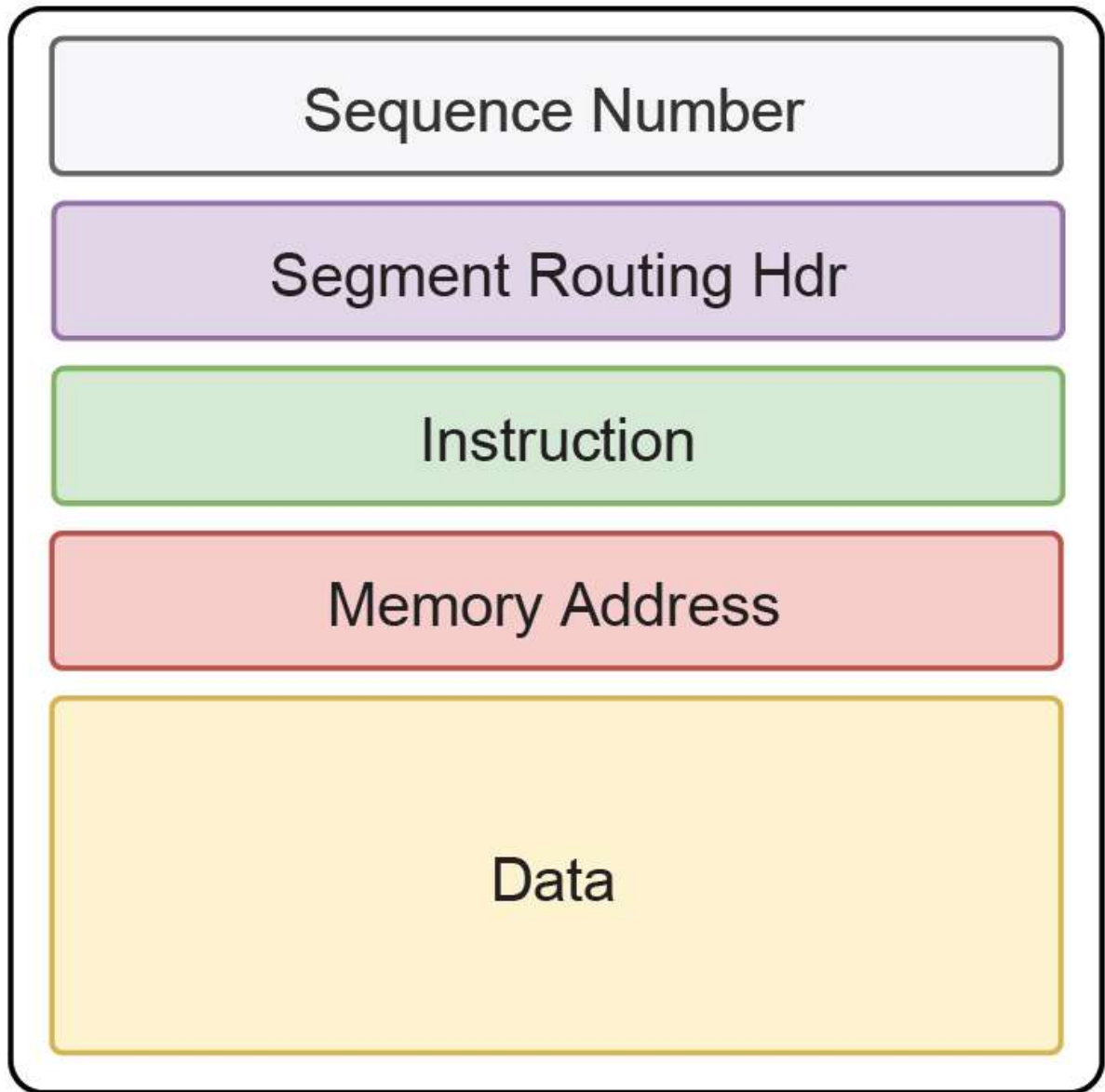
以使用PCIe、CXL，当然也可以使用您自己的硬件电路逻辑在FPGA上，并通过AXI总线和NetDAM连接。

而通过内存的映射，这样的一套架构构建了一个统一的内存访问系统，相较于以往的其它架构可以更加便捷和经济的实现超大规模计算集群



### NetDAM报文格式

NetDAM是一个基于包传输的协议，无论在主机内还是主机间通信，Payload具有相同的格式，其格式定义如下：



- Sequence: 字段主要用于包顺序和可靠传输使用.
- SegmentRouting Header: 主要是借鉴Ruta<sup>[1]</sup>的工作, 实现在数据中心内部的多路径拥塞避免, 与此同时也借助它实现了服务链(Service-Chain)的基于流的计算业务(Streaming Computation),通过Directed acyclic graph(DAG)很容易将计算业务分担到各个节点, 并使用SR技术串联在一起, 具体我们在Ring AllReduce中会有一个详细的案例
- Instruction: 这是整个NetDAM的核心, 直接将指令编码放置在数据包中并和数据紧耦合在一起实现了以数据为中心的指令集架构(Data-Centric ISA),NetDAM原始框架中提供了基本的 READ、WRITE、MEMCPY 及部分 Atomic 原子操作.但是我们预留了8bits用于用户可选择的扩展自己的指令集来支持不同的DSA或者不同的Serverless Function Call, 例如将后续的地址段扩展为Serverless的函数入口地址, 这样的架构对于Serverless平台具有极高的价值。

- Address: 这是指令操作的地址空间，根据不同的指令可以定义不同的地址，并且可以设计相应的IOMMU来实现地址映射将远端的地址映射到本地或者在多个虚机和容器中映射地址，或者给Serverless实现虚拟的地址空间。
- Data: Data字段域通常主要用于主机间通信，主机内通信本来很多东西都在内存上，直接就能共享地址访问，除非是需要Memory Copy的场景。通过这样的方式实现9000B传输时，SIMD可以同时操作多大约2048个32位浮点数，配合NetDAM自带的ALU可以实现超大规模的并行计算，比现有处理器的AVX512处理器执行效率高了数倍。

其实这就是我前面一直发的一个东西的意义：



## NetDAM传输层

在设计NetDAM传输层时，我们的原则是在提供超低延迟和超大带宽的同时尽量减少对硬件的依赖，其实简单来说就是避免使用IB或者特殊的串行结构和特殊的可编程交换机，当然可编程交换机可以辅助NetDAM实现分布式的MMU和内存安全访问控制，这个在后续的章节会详细介绍。结论：我们采用了标注的IP/UDP over Ethernet的方式来承载主机间的NetDAM通信。

确定性延迟：NetDAM具有固定的硬件流水线处理报文的读写，由于在主机间采用MPI的通信方式，因此可以避免像PCIe那样需要DMA和监听维持缓存一致性带来的长尾延迟缺陷。我们测试了线-线(wire-to-wire)使用NetDAM从远端SIMD读32个32位浮点数的操作，平均延迟 618纳秒，抖动 39纳秒，最大延迟也就 920ns（注意，我才不会用什么P99延迟麻痹自己）。这种确定性延迟对于拥塞控制非常有用，例如SWIFT就不用去测试主机处理延迟了，都固定的，underlay链路选择好了整个路径延迟就固定了，简单的流控加上去就够了。

可靠传输：在NetDAM设计中，可靠传输是可选的，因为一方面实现无丢包的以太网非常难，特别是在一些虚拟化环境中，而另一方面很多分布式系统应用层接口都具有幂等性，因此我们没有在NetDAM设计中把可靠传输作为必选，同时我们在后续的MPI Allreduce操作中也使用了幂等的处理方式简化故障恢复。但是正如前文所述，由于确定性延迟的存在，可靠传输实现非常容易。^



保序: 在很多并行计算中, 如果算子具有可交换性, 因此就可以乱序执行。由于我们在NetDAM中放置了地址和数据段长度的信息在地址空间上隔离了不同的内存, 因此在传输过程中的操作是完全可交换的。当然报文头部也提供了序列号的支持, 用户也可以根据自己的计算任务构建相应的排序和顺序执行器件。

多路径: 许多数据中心都采用Fat-Tree的架构, 当然也有很多超算也采用了2D-Torus和3D-Torus的架构, 相对于超算而言, Torus的架构扩展性和经济性都会好很多。因此NetDAM提供了基于Ruta的SegmentRouting over UDP方式, 并且支持Service-Chaining的方式来实现分布式计算。

## 可编程ISA

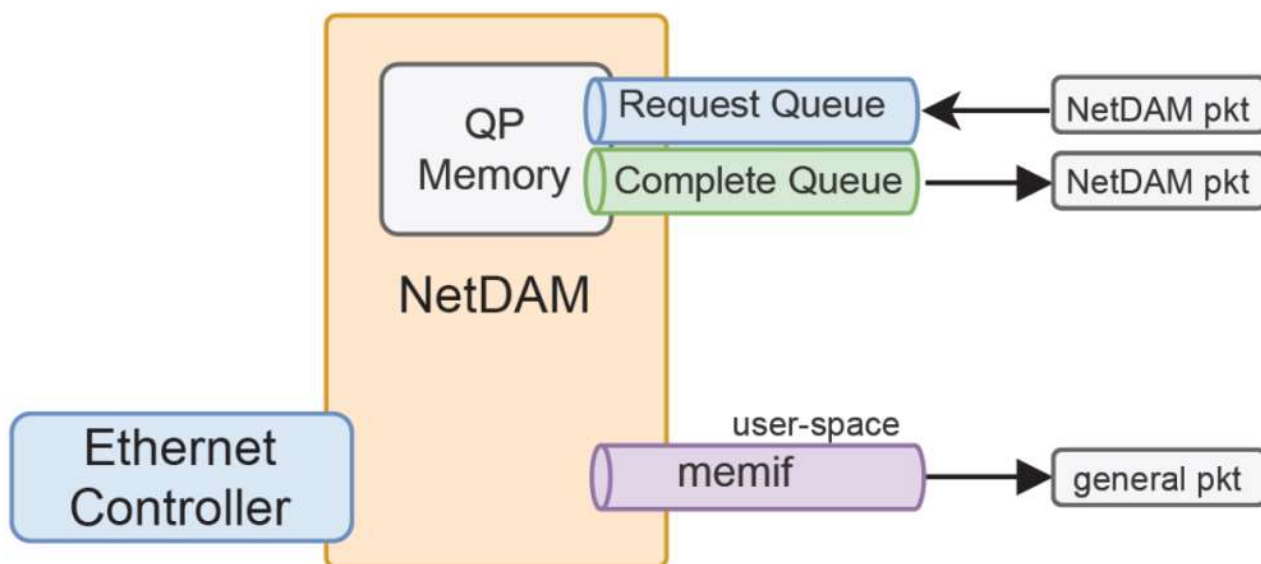
传统的处理器架构通常有固定的ISA, 例如X86、ARM、MIPS等。RISC-V给我们带来很多新的思路, 提供基本的ISA的同时增加可选的ISA来实现不同ISA芯片。而NetDAM实现上也借鉴了这样的做法, 但是我们更像一个RPC调用, 因为这样对软件更加友好并且完全与已有的CPU指令集无关。我们在内存中放置了一个特定的区域构建Queue Pair 用于DMA接收Request和放置Complete信息, 主机内由于指令携带部分操作数长度肯定小于一个CacheLine, 因此可以很容易的构建好一个结构体发送给ReqQ和从CompleteQueue中接收ACK, 而主机间, 我们可以根据内存访问地址自动封装成UDP报文发送。在NetDAM的基础模板框架中, 我们只提供了基本的READ、WRITE、MEMCPY及部分Atomic原子操作, 剩下的留了数个bits给用户自定义扩展, 因此这样的架构精简可选, 并且可以实现Xilinx提出的Adaptive Computing. 例如针对神经网络或者科学矩阵计算, 我们可以实现超大规模的SIMD(ADD, SUB, MUL, XOR, MIN, MAX)支持, 因为我们的架构中添加了大量的on-Chip ALU, 当然还有更快的做法是直接换用三星的PIM-HBM实现存内计算。

当然用户也可以定义自己的电路或者IP核并通过AXI总线或者以后通过CHI总线和一些封装技术实现更大规模的芯片。例如针对MPI-Allreduce场景, 我们实现了Reduce-Scatter和All-Gather指令集。以后针对DPU的场景我们还可以添加加解密和压缩器件实现压缩、加解密、Hash、LPM等指令集的扩展。

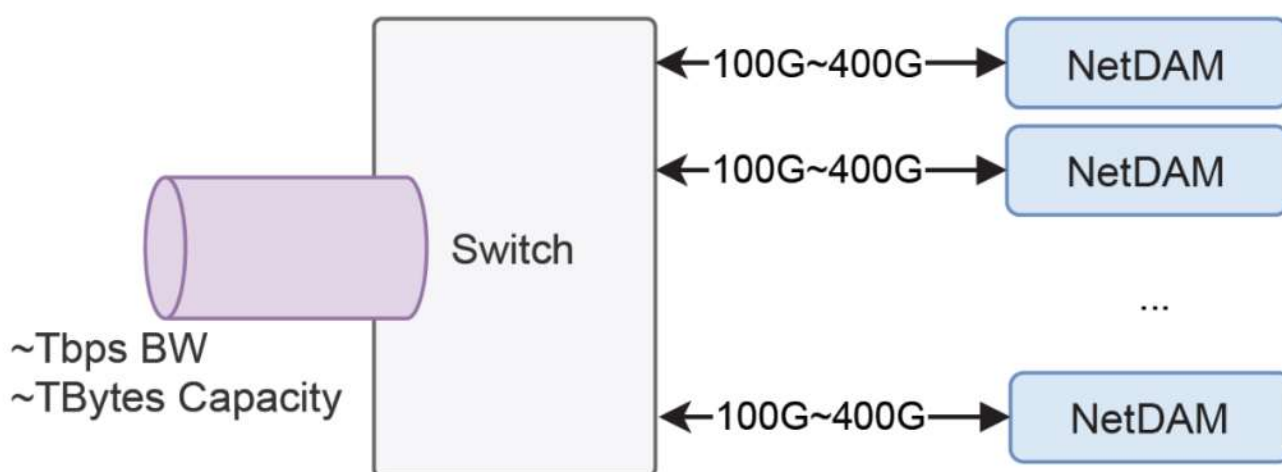
## 内存编址

每个NetDAM设备都有自己的独立地址空间, 而且这些空间和挂载的主机需要IOMMU进行映射, 正如前文所述, 在主机侧我们添加了一段特殊的地址空间来构建一个命令QueuePair接收和发送NetDAM指令包。与此同时, 我们也可以通过实现特殊的电路构建一个Memif并将它直接映射到userspace来实现容器和虚拟机的高速以太网访问, Memif是思科贡献给FD.io的技术, 我们已经通过它实现了Go语言原生28Mpps的收包能力, 这样简单的设计似乎连DPDK都可以bypass掉了, 嘻嘻~





当然谈到内存编址，还有一个特别的亮点是，NetDAM可以单独脱离主机工作，并直接连接到交换机上，实现整机Multi-Tbps和Multi-TeraBytes的内存池：



如果没有可编程交换机，每个NetDAM可以使用自己独立的IOMMU实现地址映射，而当使用了可编程交换机后，可以直接将MMU放置于交换机上，实现交换机根据NetDAM报文中的内存地址到底层NetDAM设备IP地址的自动翻译。

## Incast

Incast一直是整个数据中心网络中最头疼的问题，本质上它是一个多对一通信的问题，然而我们可以采用基于块交错的编址技术，使用NetDAM提供的内存池，将这样的操作变成多对多（实际上应用无感知)的处理，最后由真正的接收的那个主机按需取回即可，很简单的避免了拥塞。这样的处理方式在网络设备上可以追溯到上个世纪，思科和Juniper都采用类似的方法避免拥塞。

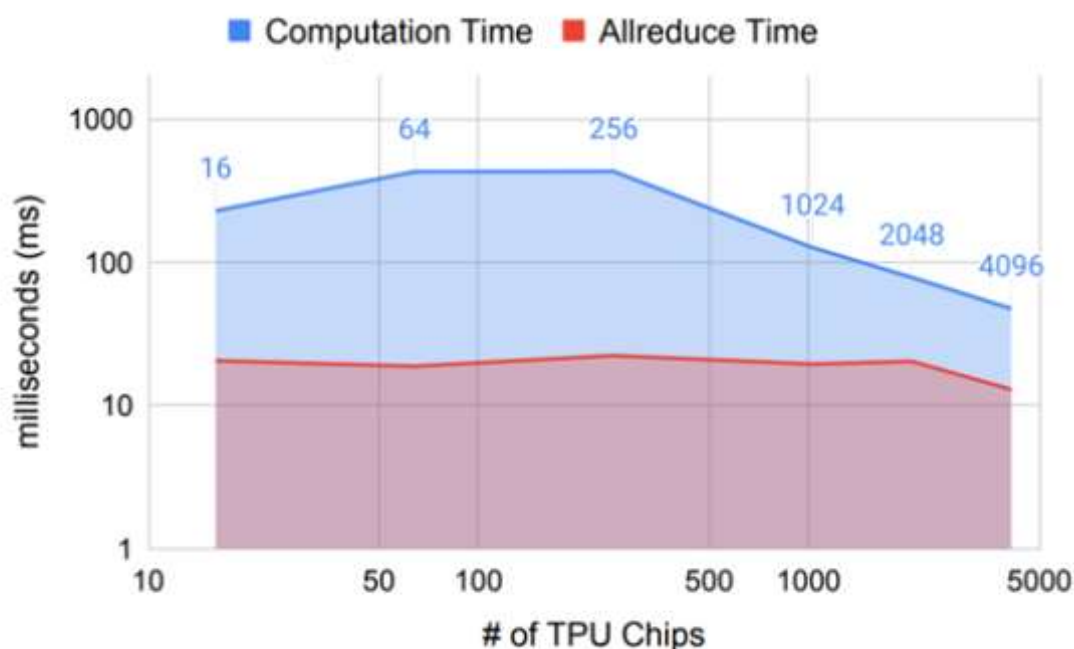
## 安全



当你跨机共享内存和允许远程执行时，安全就是一个大问题，我们可以通过SDN控制扮演一个操作系统内存管理进程的角色，通过简单的malloc、free RPC 实现内存分配，并且同时通过配置相应的ACL实现访问控制，当然我们还可以实现 加密写 和 解密读 等可扩展指令来进行更加安全的计算。

## MPI Allreduce应用

我们可以看到，现在AI模型规模越来越大，而参数同步梯度优化已经成为最大的瓶颈了。



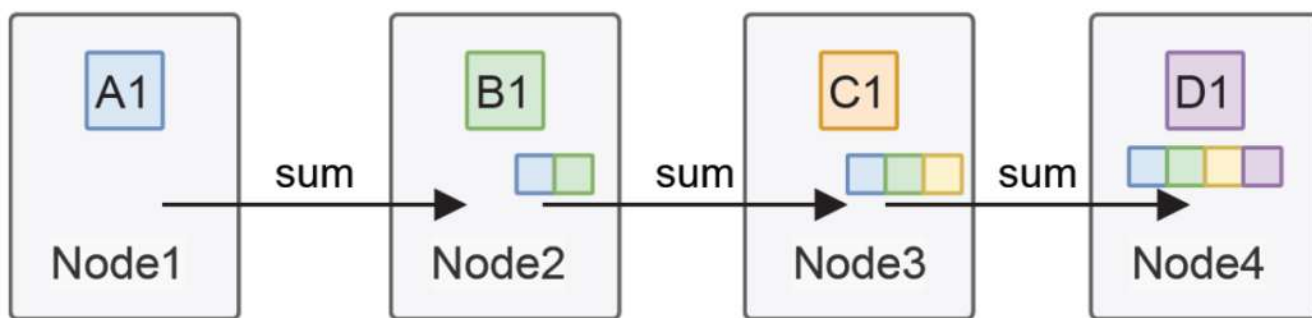
因此我们使用NetDAM实现了Allreduce的操作来显示这样的体系架构的优越性。算法上我们使用了百度于2017年发布的Ring Allreduce<sup>[2]</sup>算法，这个算法也被Horovod库使用，NCCL也长时间使用过。当然NCCL后来换到Tree的模式也因为RDMA实现的问题需要多次同步导致的，后面我们会详细解释。

具体算法和RoCEv2的测试结果可以参考我两个月前发的文：分布式AI训练：RoCEv2 AllReduce 实战<sup>[3]</sup>

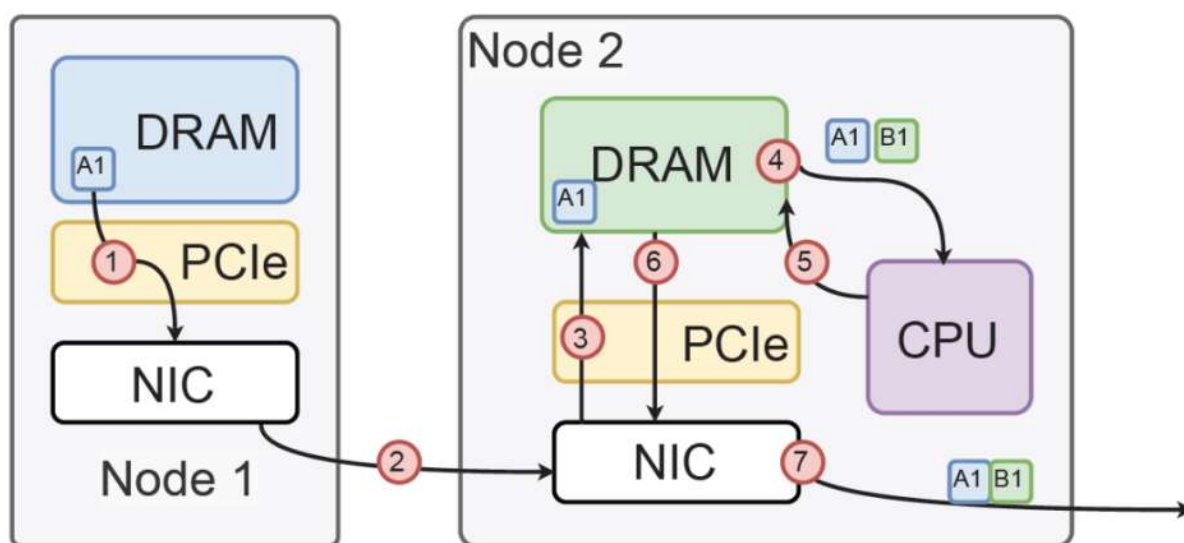
从通信量来看，Ring和Tree是相同的，计算也是相同的，我们也可以通过把Ring尺寸减小到2来构建一颗树。但是我们要选择Ring的目的是显示NetDAM分布式链式计算的优越性。我们通过对NetDAM扩展了两条指令 Ring Reduce-Scatter 和 Ring AllGather 来完成。测试环境使用了2块Xilinx Alveo U55N网卡，每个卡使用一个100G以太口和一半的HBM内存构建为一个NetDAM设备，总共四个节点连接到Cisco Nexus 93180FX交换机上。

## Ring Reduce-Scatter

它主要是一个链式的跨越多节点的计算，如下图所示，第一个节点把A1发送到第二个节点，第二个节点加上B1后发送到第三个节点，然后最终第四个节点获得A1+B1+C1+D1的结果



但是呢，这事跟DMA扯上关系就复杂了，在RDMA的实现上，数据需要多次DMA穿越PCIe总线，并且CPU需要大量的load、store的操作，在高并行的情况下，PCIe总线的拥塞时完全无法控制的，并且原始算法每轮迭代之间都还需要同步屏障。



**Figure 7: RoCE Reduce-Scatter**

而NetDAM就不同了，所有的操作都发生在PacketBuffer的SRAM里，而DRAM直接可以读取并封装好UDP包发送，整个链条伴随着SR头还可以自动路由到下一个节点产生链式反应，延迟完全可控：



当然会有一个小问题就是Error Handling，**丢包了拥塞了怎么办？**我们可以注意到对于中间节点，由于没有任何本地的内存读写，因此没有任何副作用，这些操作是完全幂等的，而只是最后一个节点需要写内存。因此我们实现了一个类似于区块链的做法，采用block hash的机制携带到NetDAM指令中，只有链的最后一个节点拥有相同的Hash才能写入，这样就解决了可靠性的问题，出了问题不用担心副作用简单的重传就能搞定。



这就是一个链条下去傻写咯，每个节点收到报文就拷贝写到本地内存，然后发送到下一个节点，也是幂等操作。当然我们都用了以太网了，也可以通过组播的方式进行复制呀，可靠传输参考很多证券行情传输，NAK指令单播复制就好。

## 对比测试

RoCEv2选择了思科的UCS-M5S服务器，Intel Xeon 6230R CPU，满配12根DDR4@2933Mhz 32GB内存，网卡Mellanox CX516A，交换机Cisco Nexus 91380FX，软件为HPC-X和OFED，测试结果536,870,912 x float32 allreduce，采用OpenMPI内带的Allreduce函数为2.8秒，而使用RingAllreduce 2.1s。初步使用NetDAM样机测试可以低至 400ms，并且我们后期还会进一步优化。

## 结论和展望

我们通过从根本上分析处理器间通信和主机间通信协议在拓扑、延迟、一致性、保序性及报文大小等各方面的区别，得出结论是直接使用任何一个协议去扩展都不是最有效的，RDMA和NanoPU正是从各自的领域去扩展通信协议但都带来了问题。

基于我们过去十多年在思科QuantumFlow处理器上的架构设计经验，从软硬件可编程友好性的经验来看，直接在以太网控制器上挂载内存并提供可编程的指令集的内存访问时最有效的方式，它可以有效隔离主机内核主机间的网络，并平滑的实现主机内共享内存、主机间消息传递的通信方式。

同时我们借助Ruta这样的主动选择路径和拥塞控制技术实现了数据中心网络的高复用，并且可以通过block interleaved的方式编址解决incast的问题。

最后我们通过FPGA实现了NetDAM，并证明了在读写延迟性上远远优于RDMA的方案，而可编程性上远优于NanoPU的方案。

未来当我们拿到支持CXL的FPGA和CPU样机后，我们还会进一步在主机内网络上进行优化处理，例如实现更加有效的容器网络边斗(SideCar)等，当然针对一些云服务提供商场景，我们还会进一步研究基于NetDAM的NVMeOF和持久性内存的技术。最后希望能够技术扶贫大家一起把它做起来:)

## Reference

[1]

Ruta云原生路由架构: <https://mp.weixin.qq.com/s/NcKyIJ-8dpfAtkKJ-kUsdA>

[2]


Ring AllReduce: <https://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/>


[3]

分布式AI训练：RoCEv2 AllReduce实战: <https://mp.weixin.qq.com/s/z3OvMdqiSOHY0jLFRyxJ5w>





 赞(0)

 分享

---

上一篇: 科技巨无霸来临: 最后一批鸿蒙升级开始, 华为向生态公司转向 (/CrwyHgcr2jqmgx.html)

下一篇: 华为新时代: 没有一片汪洋不可飞渡 (/Cra9RwckNqlfys.html)

---

免责声明: 本文仅代表文章作者的个人观点, 与本站无关。其原创性、真实性以及文中陈述文字和内容未经本站证实, 请读者仅作参考, 并自行核实相关内容。如发现有害或侵权内容, 请联系邮箱: 66553826(AT)qq.com, 我们将在第一时间进行核实处理。

Copyright© 2017-2021 Chrome插件网 (/) 版权所有 渝ICP备20009290号-1 (<http://www.miibeian.gov.cn>)

