开放原子开源基金会 OPENATOM FOUNDATION | OpenEuler | TongTu 计算机网络与协议实验室

灵衢通信关键技术和应用

演讲人　　演讲人职位

李力军　　openEuler社区UMDK maintainer

目录

智算和通算业务的技术趋势演进和通信诉求

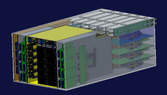
核心业务场景

推理	强化学习	Agent&多模态	通算生态
模型规模：超万亿稀疏 模型负载：长短混合序列	训推转换：分钟级->秒级 多模型混布：细粒度调度	长序列：K级->M级 多轮推理：100ms->ms	典型应用：数据库、大数据、虚拟化、Redis等

关键技术趋势

	智算	通算
超节点性能	• 大EP推理的极致低时延和高吞吐 • NPU/GPU/CPU/SSU异构算力高性能互联	
通信性能		• us级的极致低时延通信 • 充分使能超节点网络多路径互联带宽优势
生态接入	• DeepEP/类SHMEM库等AI通信生态接入 • 支持Sglang/vLLM等昇腾开箱性能加速	• 传统Socket、RoCE verbs接口接入，应用零修改迁移 • 支持bRPC/gRPC/客户自研等主流RPC框架
Agentic AI趋势	• 将成为新的算力增长领域，需构建高效安全弹性，多模海量记忆，高并发低时延计算，三大Infra能力	
典型应用		• 软硬协同性能，构建系统级的性价比优势 • 数据库/大数据/内存池化性价比提升


AI SuperPoD


AI Server


通算 SuperPoD


通算 Server

灵衢通信诉求

高性能	大规模	生态&易用
• **超节点训推时延：**All2All等MoE通信开销占比较高，需要优化 • **超节点亲和通信：**计算与通信深度融合，提升并行度	• **超节点互联：**规模翻倍、1D/2D Mesh组网 • **集群互联：**基础通信性能，支持不同代际和异构网络通信	• **开箱性能：**DeepEP、Socket和xRPC等主流社区通信加速 • **开箱可用度：** 超节点网络可用性、可靠性

灵衢通信技术：UMDK使能openEuler打造灵衢异构融合高性能通信底座

智算通信加速（高效训推）

- 突破MoE大EP通信低时延，TPS提升10%
- 跨层全协同技术提升算力利用率

通算通信加速（大带宽）

- URMA使能异构设备互联直访，提供T级超大带宽
- 灵衢URPC，us级极致低时延，RPC类中间件性能提升20%

生态友好兼容（零修改迁移）

- 兼容DeepEP等主流互联网通信接口，对接vLLM/SGLang等框架
- Socket/ Verbs/gRPC/bRPC等已有生态编程接口零修改无缝接入
- URMA南向支持兼容传统以太网卡硬件

关键技术示例

1.零修改兼容DeepEP等主流互联网通信接口

客户调用DeepEP接口不变，在昇腾平台上自动切换底层实现，使用灵衢通信加速训推。

2.Socket兼容方式使用灵衢通信

支持应用“零”代码修改接入灵衢网络加速通信性能

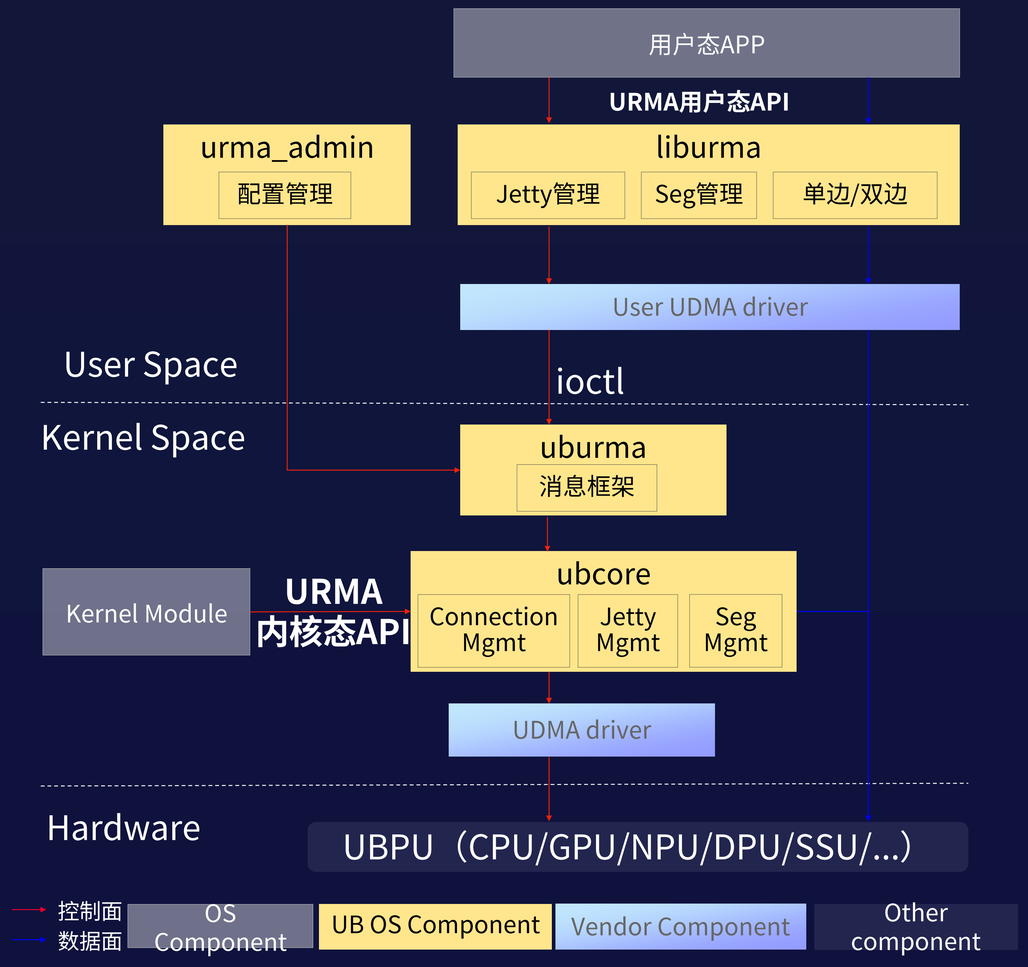
3. URMA异步编程接口使用灵衢通信：

客户使用URMA异步编程接口，充分释放灵衢总线性能。

灵衢基础通信URMA/URPC在UB协议栈架构中的位置

- URMA位于功能层，提供异步访问的编程模型，是灵衢通信能力的编程入口
- 通过基于Jetty的URMA API接口使能UB事务层的内存语义和M2N消息语义
- 为应用提供乱序、多路径等高性能、高可靠和大规模的UB通信能力
- URPC提供异构算力间的统一远程过程调用能力

灵衢基础通信URMA：提供高性能异步内存访问和M2N消息收发，支持生态兼容

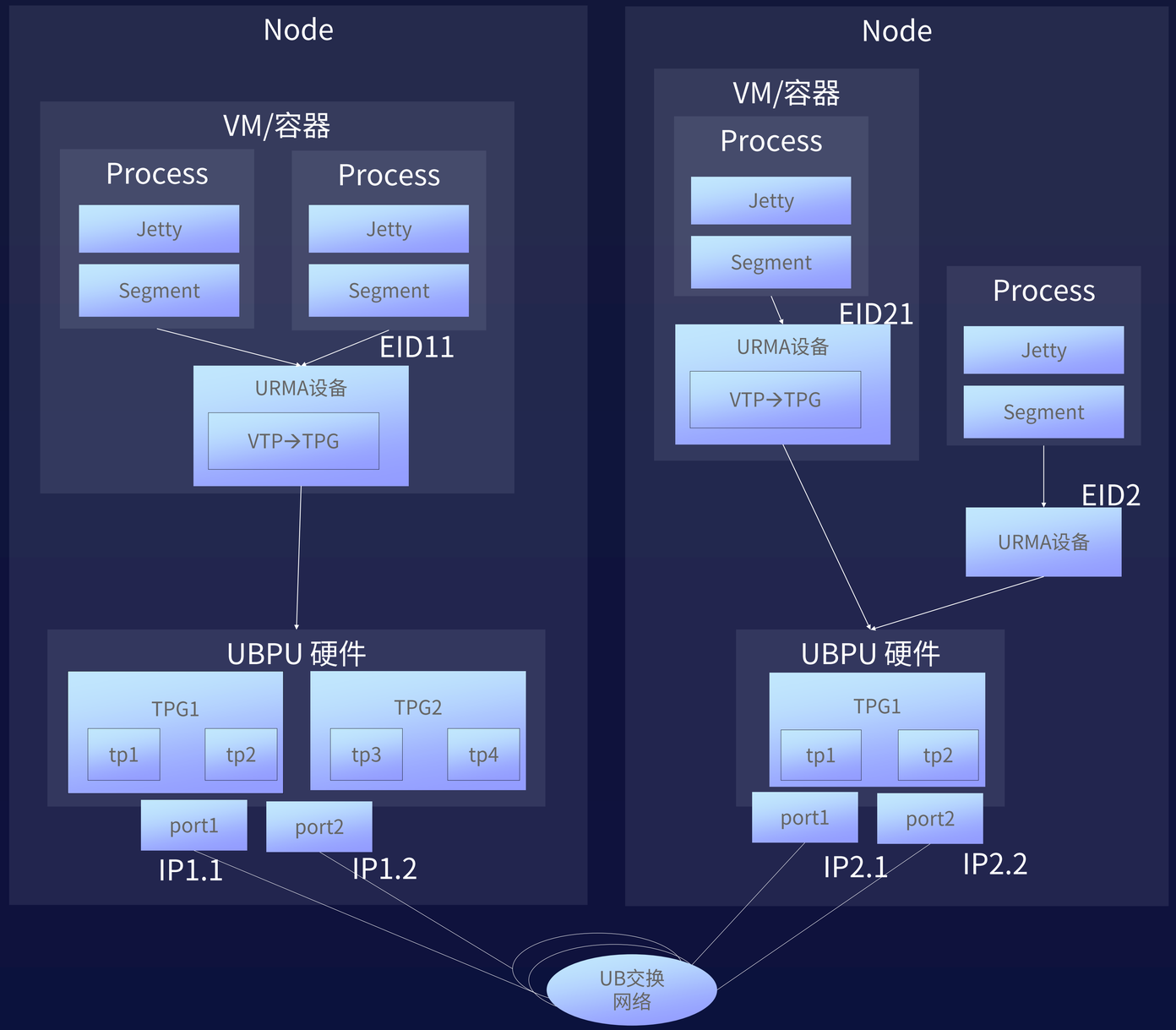


核心设计

- URMA设备对应UBPU的某个UB Entity，由EID唯一通信的标识
- URMA软件架构支持各类xPU算力间的统一接口通信，北向支持兼容Socket/Verbs接口，南向可对接三方Vendor的UB硬件或RoCE等传统以太网卡
- Jetty是URMA的基本通信单元，支持接收、发送和完成事件，可看成应用通信请求的“通信码头”，提供异步执行和多对多通信能力，支持RM/RC/UM三种模式

	1对1通信	多对多通信
可靠传输	Reliable Connection (单路径)	Reliable Messaging (乱序、多路径、大规模)
不可靠传输	No support in UB	Unreliable Messaging (管理和启动)

灵衢基础通信URMA：Jetty编程模型介绍



编程模型关键概念

- UB交换网络：基于IP地址路由，超节点网络中压缩CNA提升效率
- TP/CTP：TP提供可靠性和拥塞控制能力，多路径时使用TPG。轻量CTP提供更大带宽传输
- URMA Device：逻辑设备对应UBPU硬件的某个UB Entity，使用EID标识，基于该对象创建Jetty或Segment资源，
- Segment：由一段连续的VA地址空间和Token信息组成，用于单边内存访问

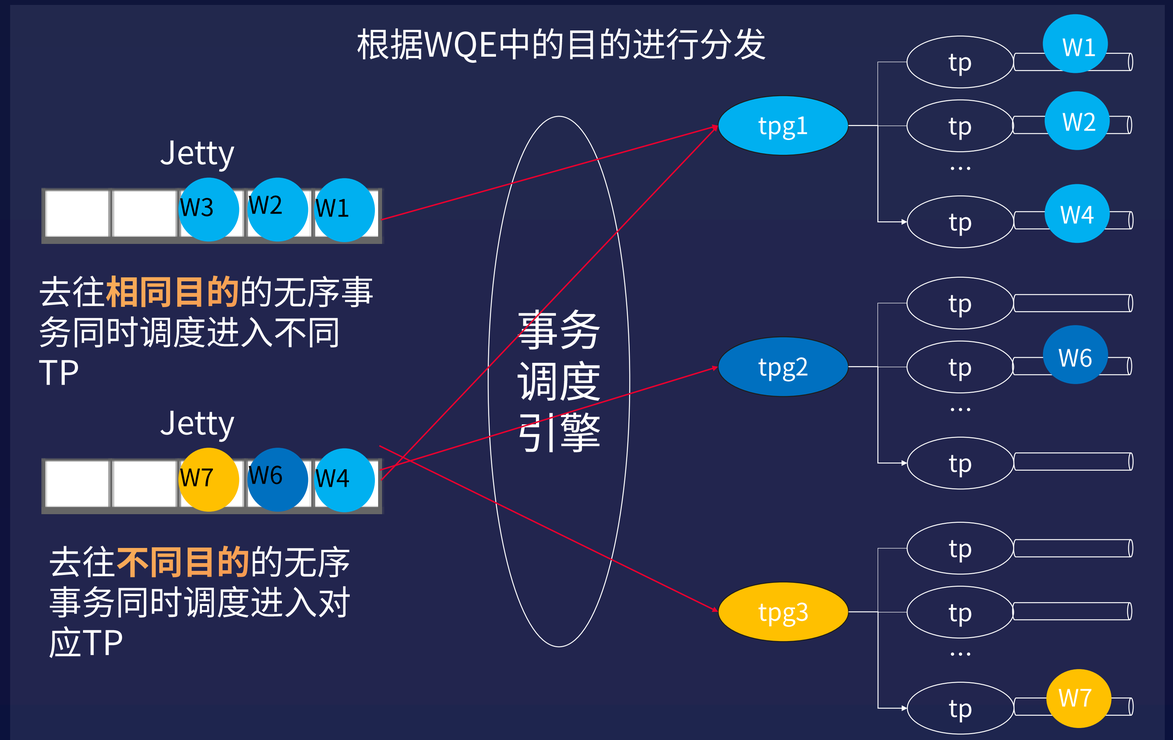
代码示例

```
jfc = urma_create_jfc(urma_ctx, &jfc_cfg);
//设置jetty_cfg，指定jfc
jetty = urma_create_jetty(urma_ctx, &jetty_cfg);
seg = urma_register_seg(urma_ctx, &seg_cfg);
//利用socket或其它jetty交换 远端segment信息
tseg = urma_import_seg(urma_ctx, &rem_seg, &token, 0, flag);
//将本地seg、远端tseg构造一个read的wr，下发给jetty多路径传输
urma_post_jetty_send_wr(jetty, &wr, &bad_wr);
urma_poll_jfc(jfc, 1, cr);
```

灵衢基础通信URMA：关键技术介绍

支持多路径大带宽Tb级传输

- 每个WQE就是一个事务，发生在一个Jetty上的所有事务序列称为一个事务流
- RM模式下，事务会被调度进入TP Group，最终选定TP发送
- 事务流按照下发顺序调度进入TP Group

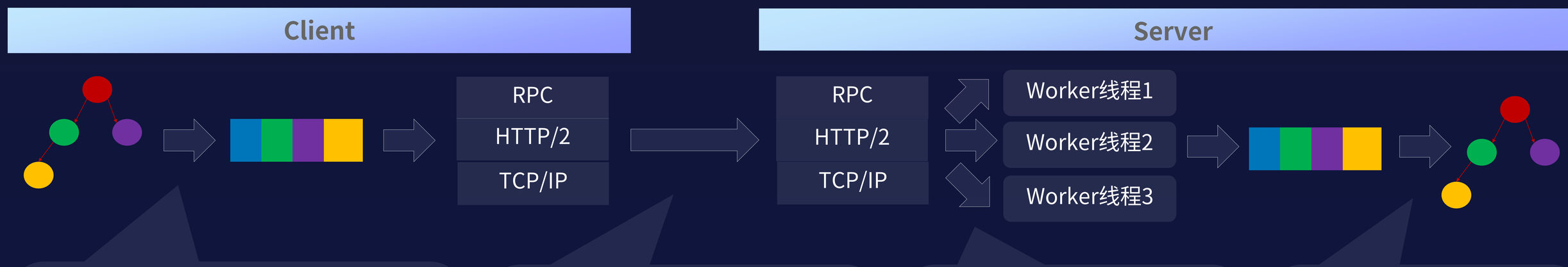


支持灵活的乱序执行

- NO/RO：如果事务WQE标记为NO（无需保序），或标记为RO（弱事务序），则执行顺序是任意的
- SO：如果事务WQE标记为SO（保序标记），该事务**必须**在其前面所有RO/SO事务执行完毕之后执行

- w3必须要在w1、w2执行完成后，才能执行
- w5必须要在w1、w2、w3、w4执行完成后，才能执行
- w1、w2、w4没有执行顺序的要求
- 一种可能的执行结果：W5 W3 W4 W1 W2

现有RPC性能挑战：序列化、网络传输和分发调度三方面



参数序列化

- 为WAN场景设计，网络带宽是瓶颈，变长编码效率低，为节约带宽进行**编码压缩**
- 需要一次**拷贝**完成离散内存线性化

传输

- **HTTP/2**为WAN场景设计，请求头格式复杂，处理效率低
- **TCP/IP**传统内核协议栈时延高

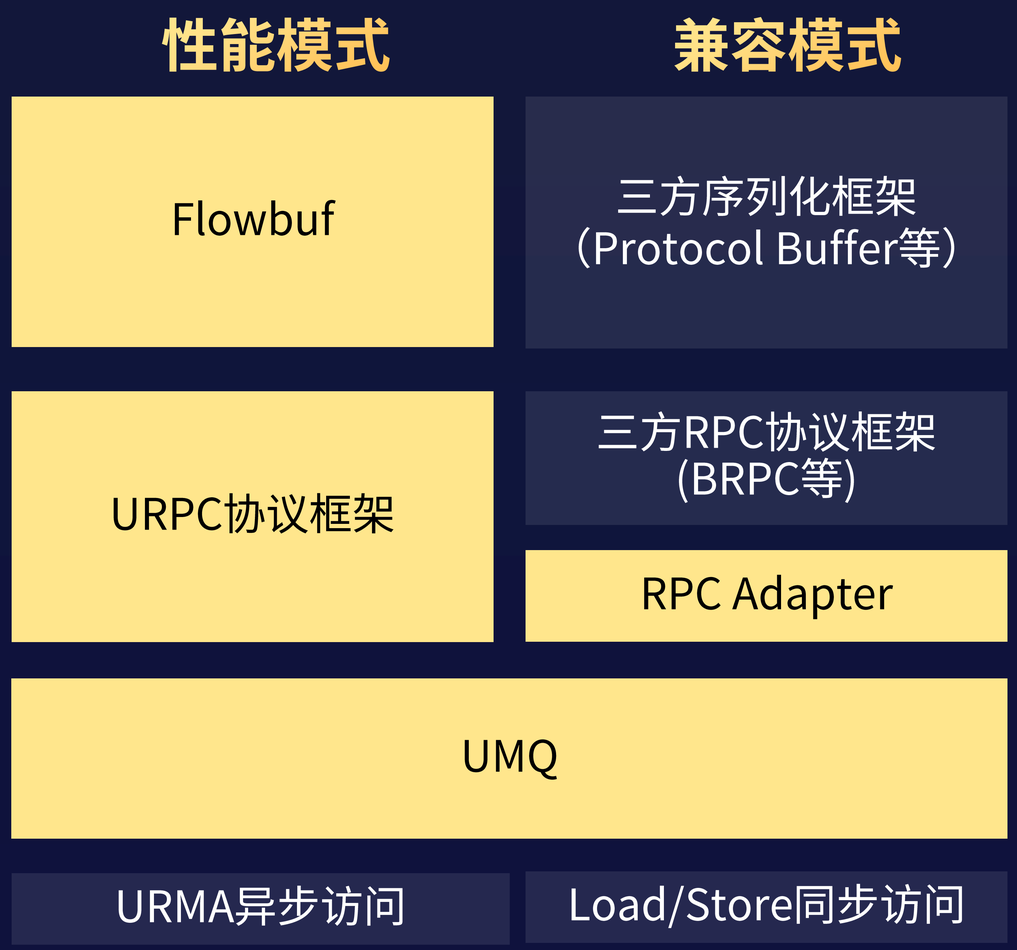
调度分发

- 通过内核事件唤醒Worker线程，时延高
- 软件集中式分发，性能受限

反序列化

- **变长编码**效率低
- 需临时分配大量零散内存对象，并进行一次拷贝

灵衢URPC：提供us级极致低时延RPC调用和高性能序列化能力

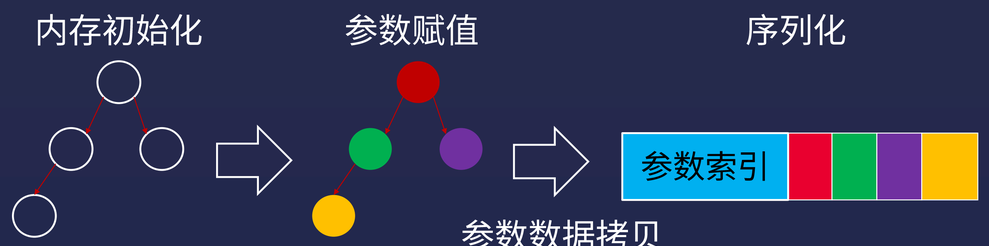


核心设计

- Flowbuf：提供高性能序列化能力，支持免拷贝极简序列化和引用传参提升RPC传输效率
- URPC协议框架：支持CPU/NPU作为Client以RPC函数方式调用SSU等Server侧异构算力的能力，支持使用Jetty Group硬件分发到多核处理
- UMQ：使用URMA异步访问和Load/Store同步访问两类编程模型，充分使能UB事务层语义
- RPC Adapter：三方RPC（BRPC等）适配模块，支持RPC应用“零”修改提升传输性能

Flowbuf序列化：支持免拷贝、数据直达，大幅提升序列化性能

现有序列化方案



拷贝损耗大：参数离散内存需拷贝成连续内存完成序列化

优化方案

FlowBuf值传递和引用传递的序列化方案

关键技术

- **值传递序列化**：连续内存中基于索引进行赋值，同时完成序列化
- **引用传参序列化**：在序列化内存中只存放数据内存地址，由server侧使用UB事务语义按需拉取

使用示例

IDL定义：

```
message request{
      int32 id;
      string name;
}
service greeter {
     rpc sayhello (request) returns (response);
}
```

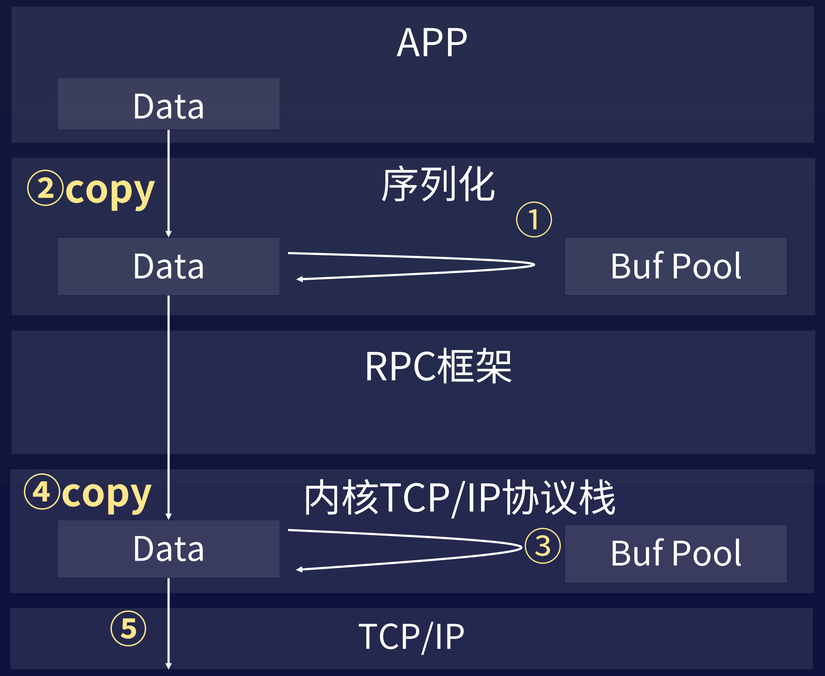
代码示例：

```
// 申请并初始化FlowBuf内存
Requeut *req = new (std::nothrow) Requeue();
// 赋值时内部自动做序列化
Req->set_name(“Client”);
status = stub->sayhello(req, &rsp);
```

UMQ和RPC Adapter：充分发挥UB通信的高性能优势，兼容业界RPC框架提升性能

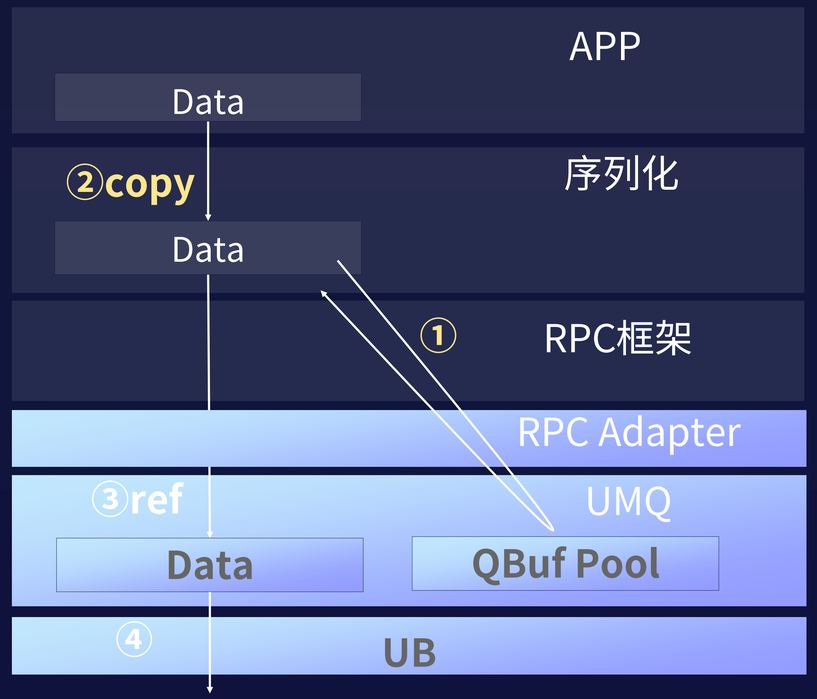
方案和挑战

- RPC需要二次数据拷贝
- 内核TCP/IP协议栈开销大



① **申请**序列化Buf
② APP Data**拷贝**到序列化Buf完成序列化
③ **申请**内核IO Buf
④ 序列化Buf**拷贝**到内核IO Buf
⑤ IO Buf基于**TCP/IP发送**

关键技术



① **申请**UMQ QBuf
② APP Data**拷贝**到QBuf完成序列化
③ QBuf**地址通过Socket API下发**到UMQ
④ QBuf基于**UB发送**

- 高性能通信内存池：批量申请/释放、thread local缓存、热点Qbuf复用，数据拷贝优化2次->1次
- 高效UMQ传输：大小IO自适应机制，基于共享内存支持小IO传输；基于URMA弱事务序多路径支持大IO传输
- 自动回退到TCP：识别对端非UB，自适应切换为TCP/IP通信

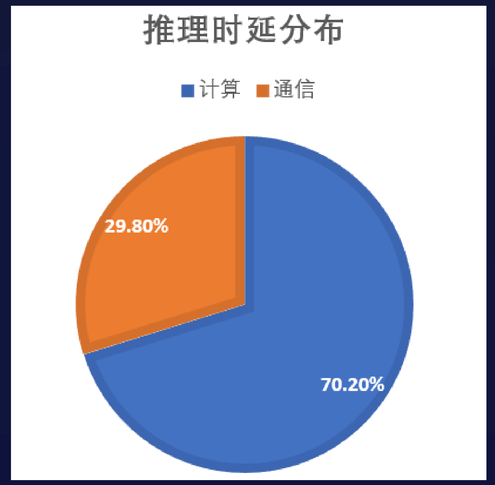
昇腾亲和通信加速库CAM：提供低时延大EP通信和MoE通算融合加速

关键技术

- 大EP通信加速：提供昇腾平台上的高性能MoE通信，兼容DeepEP接口并接入Sglang/vLLM
- FusedDeepMoE：通过Token级细粒度流水实现大EP通信+专家计算的深度融合加速
- M2N通信加速：针对AFD/EPD等计算分离场景探索异步M2N通信加速，缓解计算等待
- MIXL：构建PD分离场景推理传输加速技术，支持自动聚合和异构PD间传输

CAM：大EP通信实现内存语义与计算并行，通信性能提升30%+ ，北向兼容DeepEP

$$\text{通信时延} = \{\frac{\text{数据量}\downarrow}{\text{有效带宽}\uparrow} + \text{算子静态时延(初始化时延、同步开销、本地拷贝)}\downarrow\}$$



Dispatch代码流程	关键步骤	关键步骤描述
通信数据准备	1.1 数据格式初始化	算子内部数据结构初始化 ①
	1.2 计算对端地址与计数	计算通信数据在对端sharememory的地址偏移并计数
发送数据	2.1 GM2UB	将token从GM搬运到UB
	2.2 UB内数据量化	将token从FP16量化到INT8，降低通信数据量 ②
	2.3 UB2GM	将token从UB搬运到对端的sharememory
通信同步	3.1 核间同步	NPU内多AIV核间同步，确保所有核的数据都已完成发送
	3.2 写flag+count到远端	通知所有对端token传输已完成，并告知token的计数 ③
	4.1 读本地的flag+count	轮询对端写过来的传输完成标志，并读取token的计数
	4,2 核间同步（卡间同步）	NPU内多AIV核间同步，确保所有对端的数据都已收齐
本地拷贝	5.1 计算输出地址	计算通信数据在输出buffer的地址偏移
	5.2 GM2GM	将token从sharememory搬运到输出buffer

问题1 通信前准备耗时长：计算各个token在对端memory地址，**scalar核不适合复杂逻辑，地址计算耗时长**；

优化技术：Scalar标量计算→Vector向量计算，加速数据准备过程

问题2 传输数据时间长：因网络冲突和端口负载不均，平均耗时占比较高；

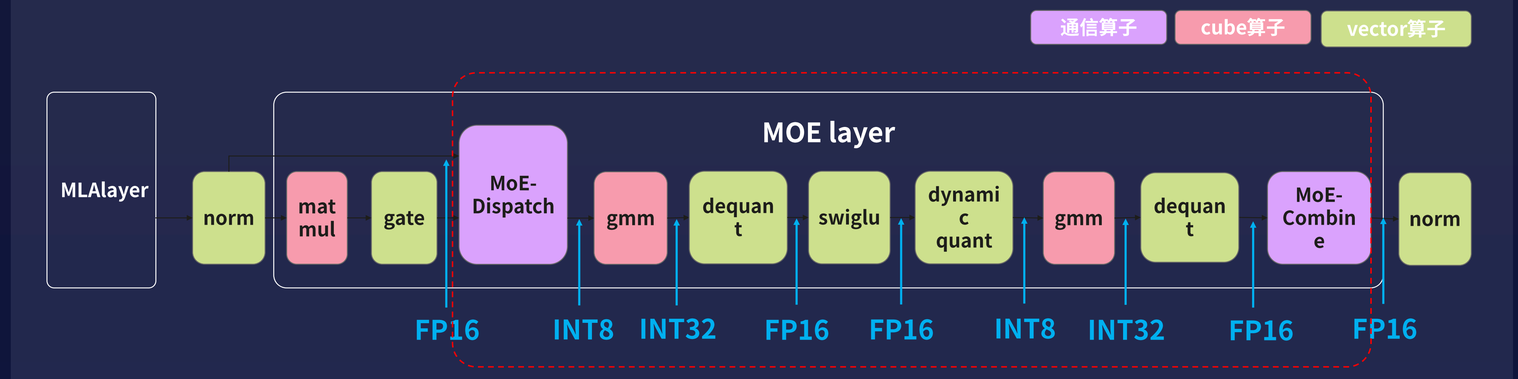
优化技术：1）共享专家token通信256->1变为8->1；
2）通过地址偏移实现NPU各端口负载均衡；

问题3 同步耗时长：Dispatch算子内涉及两次核间同步，**快慢核和快慢卡导致同步耗时长**；

优化技术：
1）采用EPLB负载均衡
2）构建FusedDeepMoE算子，进一步融合通信和计算，降低同步影响，见下页

CAM：实现大EP通算融合加速FusedDeepMoE，推理TPOT下降5%~10%

场景：DeepSeekMoE层涉及多个计算和通信算子串行执行



挑战和技术思路

挑战：

- 计算与通信串行执行，通信耗时（传输+同步）未被掩盖；
- 昇腾cube算子和vector算子串行执行任务，算力浪费；
- 算子下发次数多，kernel launch耗时长；

技术思路：

- 构建MoE融合大算子，实现计算&通信、C核与V核细粒度流水，掩盖部分算子执行耗时

实现方案

融合一：dispatch+gmm1+dequant1+swiglu 深度融合（先通信后计算）

- 基于gmm/dequant/swiglu可block-wise计算的特性，将dispatch与三者进行深度融合流水，利用cube计算耗时掩盖通信和vector计算耗时

融合二：gmm2+dequant2+combine 深度融合 （先计算后通信）：

- 将gmm与dequant、combine的深度融合流水，利用cube计算耗时掩盖vector计算和通信耗时

融合三：将上述两部分进一步融合，形成FusedDeepMoE，已接入Sglang，vLLM合入中

Thank you & QA

更多信息欢迎访问openEuler社区

https://gitee.com/openeuler/umdk

更多技术交流欢迎加入微信群

“灵衢通信库UMDK交流群”