LITE Kernel RDMA Support for Datacenter Applications

Shin-Yeh Tsai, Yiying Zhang
RDMA in Datacenters

1983: Berkeley Socket
- Userspace
- Kernel
- Hardware

1995: TCP Offload engine
- U-Net

2000s: Arrakis & mTCP
- RDMA in HPC

2014: IX

2017: RDMA in Datacenters

Unknown date: ?
RDMA (Remote Direct Memory Access)

- Directly read/write remote memory
- Bypassing kernel
- Memory zero copy

- Benefits
  - Low latency
  - High throughput
  - Low CPU utilization
Things have worked well in HPC

• Special hardware
• Few applications
• Cheaper developer
RDMA-Based Datacenter Applications
RDMA-Based Datacenter Applications

- Pilaf [ATC ’13]
- HERD-RPC [ATC ’16]
- Cell [ATC ’16]
- FaRM [NSDI ’14]
- Wukong [OSDI ’16]
- FaSST [OSDI ’16]
- HERD [SIGCOMM ’14]
- Hotpot [SoCC ’17]
- NAM-DB [VLDB ’17]
- RSI [VLDB ’16]
- DrTM [SOSP ’15]
- APUS [SoCC ’17]
- Octopus [ATC ’17]
- DrTM+R [EuroSys ’16]
- FaRM+Xact [SOSP ’15]
- Mojim [ASPLOS ’15]
Things have worked well in HPC

- Special hardware
- Few applications
- Cheaper developer

What about datacenters?

- Commodity, cheaper hardware
- Many (changing) applications
- Resource sharing and isolation
Native RDMA

**User-Level RDMA App**

- **Conn Mgmt**: Connections
- **send**
- **recv**
- **Keys**: node, lkey, rkey addr
- **Mem Mgmt**: Memory space

**Library**

- **OS**
- **RNIC**
  - Permission check
  - Address mapping

**Cached PTEs**

- lkey 1
- ... lkey n
- rkey 1
- ... rkey n

**User Space**

- **Connections**
- **Queues**
- **Keys**

**Kernel Space**

- **Network Interface**
- **Memory space**

**Hardware**

- **Permision check**
- **Address mapping**

- **lkey 1**
- ... **lkey n**
- **rkey 1**
- ... **rkey n**
Native RDMA

User-Level RDMA App

Conn Mgmt, send, recv, node, lkey, rkey addr, Mem Mgmt

Connections, Queues, Keys, Memory space

Library

Kernel Bypassing

OS

Permission check, Address mapping, Cached PTEs, lkey 1, rkey 1, lkey n, rkey n

Userspace

Hardware

User Space

Kernel Space

Hardware
Userspace

Hardware

Low-level

Difficult to use

High-level

Easy to use
Userspace

Hardware

Low-level
Difficult to use

High-level
Easy to use

Developers want
Userspace

Hardware

Low-level
Difficult to use

Developers want

High-level
Easy to use

Socket
Userspace

Hardware

Low-level

Difficult to use

RDMA

Developers want

High-level

Easy to use

Socket
Userspace

Hardware

Low-level
Difficult to use
Difficult to share

Developers want

High-level
Easy to use
Resource share
Isolation

RDMA
Socket
Userspace
Hardware

Abstraction Mismatch

Low-level
Difficult to use
Difficult to share

Developers want
High-level
Easy to use
Resource share
Isolation

Socket
RDMA
Fat applications
No resource sharing

Abstraction Mismatch

Userspace
Hardware

Low-level
Difficult to use
Difficult to share

RDMA

Developers want

High-level
Easy to use
Resource share
Isolation

Socket
Things have worked well in HPC

- Special hardware
- Few applications
- Cheaper developer

What about datacenters?

- Commodity, cheaper hardware
- Many (changing) applications
- Resource sharing and isolation
Things have worked well in HPC

- Special hardware
- Few applications
- Cheaper developer

What about datacenters?

- Commodity, cheaper hardware
- Many (changing) applications
- Resource sharing and isolation
Things have worked well in HPC

- Special hardware
- Few applications
- Cheaper developer

What about datacenters?

- Commodity, cheaper hardware
- Many (changing) applications
- Resource sharing and isolation
Native RDMA

User-Level RDMA App

Conn Mgmt -> send -> recv -> Mem Mgmt

Connections

Queues

Keys

Memory space

Library

User Space

Kernel Space

Hardware

OS

RNIC

Permission check
Address mapping

Cached PTEs

lkey 1

... lkey n

rkey

... rkey n

Userspace

Hardware
On-NIC SRAM
1. Fetches and caches page table entries
2. Stores secret keys for every consecutive memory region
On-NIC SRAM
1. Fetches and caches page table entries
2. Stores secret keys for every consecutive memory region
On-NIC SRAM

1. Fetches and caches page table entries
2. Stores secret keys for every consecutive memory region

Expensive, unscalable hardware

![Graph showing Requests/μs vs Total Size (MB)]
Things have been good in HPC

- Special hardware
- Few applications
- Cheaper developer

What about datacenters?

- Commodity, cheaper hardware
- Many (changing) applications
- Resource sharing and isolation
Things have been good in HPC

- Special hardware 😊
- Few applications 😊
- Cheaper developer 😊

What about datacenters?

- Commodity, cheaper hardware 😞
- Many (changing) applications 😞
- Resource sharing and isolation 😞
Things have been good in HPC

- Special hardware
- Few applications
- Cheaper developer

What about datacenters?

- Commodity, cheaper hardware
- Many (changing) applications
- Resource sharing and isolation
Fat applications
No resource sharing

Expensive, unscalable hardware
Are we removing too much from kernel?

Fat applications
No resource sharing

Expensive, unscalable hardware
Outline

• Introduction and motivation

• Overall design and abstraction

• LITE internals

• LITE applications

• Conclusion
Without Kernel

- High-level abstraction
- Resource sharing
- Protection
- Performance isolation
Without Kernel

- Protection
- Resource sharing
- Performance isolation
Without Kernel

- High-level abstraction
- Protection
- Resource sharing
- Performance isolation
Without Kernel
LITE - Local Indirection TiEr

- High-level abstraction
- Resource sharing
- Performance isolation
- Protection
All problems in computer science can be solved by another level of indirection

Butler Lampson
User-Level RDMA App

User Space
- Conn Mgmt
- send
- recv
- node, lkey, rkey, addr
- Mem Mgmt

LITE APIs
- Memory APIs
- RPC/Msg APIs
- Sync APIs

Kernel Space
- LITE APIs
  - Connections
  - Queues
  - Keys
  - Memory space

Hardware
- RNIC
  - Permission check
  - Address mapping
  - Cached PTEs
  - lkey 1
  - rkey 1
  - ... lkey n
  - ... rkey n
Simpler applications

User-Level RDMA App

- Conn Mgmt
- send
- recv
- node, lkey, rkey addr
- Mem Mgmt

User Space

LITE APIs
- Memory APIs
- RPC/Msg APIs
- Sync APIs

Kernel Space

LITE
- Connections
- Queues
- Keys
- Memory space

Hardware

RNIC
- Permission check
- Address mapping
- Cached PTEs
- lkey 1
- rkey 1
- lkey n
- rkey n
Simpler applications

User-Level RDMA App

User Space
- Conn Mgmt
- send
- recv
- node, lkey, rkey addr
- Mem Mgmt

Kernel Space
- Connections
- Queues
- Keys
- Memory space
- Permission check
- Address mapping

Hardware
- RNIC

LITE APIs
- Memory APIs
- RPC/Msg APIs
- Sync APIs

Cheaper hardware
Scalable performance
Simpler applications

User-Level RDMA App

User Space
- Conn Mgmt
- send
- recv
- node, lkey, rkey addr
- Mem Mgmt

Kernel Space
- LITE APIs
- Memory APIs
- RPC/Msg APIs
- Sync APIs
- Connections
- Queues
- Keys
- Memory space
- Permission check
- Address mapping

RDMA Verbs
- Global lkey
- Global rkey

Hardware
- RNIC
- Global lkey
- Global rkey

Cheaper hardware
Scalable performance
Simpler applications

User-Level RDMA App

User Space

Conn Mgmt, send, recv, node, lkey, rkey addr, Mem Mgmt

LITE APIs

Memory APIs

RPC/Msg APIs

Sync APIs

Kernel Space

LITE

Connections, Queues, Keys, Memory space

Permission check, Address mapping

Global lkey, Global rkey

Hardware

RDMA Verbs

RNIC

Global lkey, Global rkey

Cheaper hardware

Scalable performance
Implementing Remote *memset*

Native RDMA

```c
struct pingpong_context *ctx;

ctx = calloc(1, sizeof *ctx);
ctx->size = size;
ctx->rx_depth = rx_depth;
ctx->buf = malloc(roundup(size, page_size));
memset(ctx->buf, 0x7f, IB_SERVER, size);
ctx->context = ibv_open_device(ib_dev);
ctx->channel = NULL;
ctx->pd = ibv_alloc_pd(ctx->context);
ctx->mr = ibv_reg_mr(ctx->pd, ctx->buf, size, IBV_ACCESS_REMOTE_WRITE);
ctx->cq = ibv_create_cq(ctx->context, rx_depth + 1, NULL, ctx->channel, 0);

/* bunch of QP setup .... 50 LDCs */
ctx->qp = ibv_create_qp(ctx->pd, &attr);
ibv_modify_qp(ctx->qp, &attr, IBV_QP_STATE|IBV_QP_PKEY_INDEX|IBV_QP_PORT|IBV_QP_ACCESS_FLAGS);
/* exchange all required information, qpsns, psns, and keys */

/* start doing write request */
struct ibv_sge sg;
struct ibv_send_wr wr;
struct ibv_send_wr *bad_wr;
memset(&wr, 0, sizeof(wr));
memset(&sg, 0, sizeof(sg));

/* setup all required metadata for a write request*/
sq.addr = (uintptr_t)buf_addr;
sq.length = buf_size;
sq.key = ctx->mr->lkey;
wr.wr_id = 0;
wr.sg_list = &sg;
wr.num_sge = 1;
wr.opcode = IBV_WR_RDMA_READ;
wr.send_flags = IBV_SEND_SIGNED;
wr.wr.remote_addr = remote_address;
wr.wr.rdma.remote_addr = remote_address;
wr.wr.rdma.rkey = remote_key;
/* send out the request */
ibv_post_send(&qp, &wr, &bad_wr);
struct ibv_wc wc[2];
ibv_poll_cq(ctx->cq, 2, wc); /* busy poll until getting completion */
return ctx;
```
Implementing Remote `memset`

Native RDMA LITE

```c
struct pingpong_context *ctx;
ctx = calloc(1, sizeof *ctx);

ctx->size = size;
ctx->rx_depth = rx_depth;
ctx->buf = malloc(roundup(size, page_size));
memset(ctx->buf, 0,2b + ls_server, size);
ctx->context = ibv_open_device(ib_dev);
ctx->channel = NULL;
ctx->pd = ibv_alloc_pd(ctx->context);
ctx->mr = ibv_reg_mr(ctx->pd, ctx->buf, size, IBV_ACCESS_LOCAL_WRITE|IBV_ACCESS_REMOTE_WRITE);
ctx->cq = ibv_create_cq(ctx->context, rx_depth + 1, NULL, ctx->channel, 0);

/* bunch ofQP setup .... 58 L0C s */
ctx->qp = ibv_create_qp(ctx->pd, &attr);
ibv_modify_qp(ctx->qp, &attr, IBV_QP_STATE|IBV_QP_PKEY_INDEX|IBV_QP_PORT|IBV_QP_ACCESS_FLAGS);
/* build connections .... 100 L0C s */
/* exchange all required information, apns, psns, and keys */

/* start doing write request */
struct ibv_sge sg;
struct ibv_send_wr wr;
struct ibv_send_wr *bad_wr;
memset(&wr, 0, sizeof(wr));
memset(&sg, 0, sizeof(sg));

/* setup all required metadata for a write request */
sg.addr = (uintptr_t)buf_addr;
sg.length = buf_size;
sg.lkey = ctx->mr->lkey;
wr.wr_id = 0;
wr.sg_list = &sg;
wr.num_sge = 1;
w.r.opcode = IBV_WR_RDMA_READ;
w.r.send_flags = IBV_SEND_SIGNALED;
w.r.rdma.remote_addr = remote_address
w.r.rdma.rkey = remote_key;

/* send out the request */
ibv_post_send(qp, &wr, &bad_wr);
struct ibv_wc wc[2];
ibv_poll_cq(ctx->cq, 2, wc); /* busy poll until getting completion */
return ctx;
```
Implementing Remote `memset`

Native RDMA LITE

```
struct pingpong_context *ctx;
ctx = calloc(1, sizeof *ctx);
ctx->size = size;
ctx->rx_depth = rx_depth;
ctx->buf = malloc(roundup(size, page_size));
memset(ctx->buf, 0x7f, rx_depth);
ctx->context = ibv_open_device(ib_dev);
ctx->channel = NULL;

LITE_join(IP);
uint64_t lh = LITE_malloc(node, size);
LITE_memset(lh, 0, offset, size);

struct ibv_send_wr *bad_wr;
memset(&wr, 0, sizeof(wr));
memset(&sg, 0, sizeof(sg));

/* setup all required metadata for a write request */
sg.addr = ((uintptr_t)buf_addr;
sg.length = buf_size;
sg.lkey = ctx->mr->lkey;
wr.wr_id = 0;
wr.sg_list = &sg;
wr.num_sge = 1;
wr.opcode = IBV_WR_RDMA_READ;
wr.send_flags = IBV_SEND_SIGNED;
wr.wr.rdma.remote_addr = remote_address
wr.wr.rdma.rkey = remote_key;

/* send out the request */
ibv_post_send(gp, &wr, &bad_wr);
struct ibv_wc wc[2];
ibv_poll_cq(ctx->cq, 2, wc); /* busy poll until getting completion */
return ctx;
```
Implementing Remote `memset` Native RDMA

LITE

```c
struct pingpong_context *ctx;
ctx = calloc(1, sizeof *ctx);
ctx->size = size;
ctx->rx_depth = rx_depth;
ctx->buf = malloc(roundup(size, page_size));
memset(ctx->buf, 0x7b + ls_server->size);
ctx->context = ibv_open_device(ib_dev);
ctx->channel = NULL;

LITE_join(IP);
uint64_t lh = LITE_malloc(node, size);
LITE_memset(lh, 0, offset, size);

struct ibv_send_wr *bad_wr;
memset(&wr, 0, sizeof(wr));
memset(&sg, 0, sizeof(sg));

/* setup all required metadata for a write request */
sg.addr = (uintptr_t)buf_addr;
sg.length = buf_size;
sg.lkey = ctx->mr->lkey;
wr.wr_id = 0;
wr.sg_list = &sg;
wr.num_Sg = 1;
wr.opcode = IBV_WR_RDMA_READ;
wr.send_flags = IBV_SEND_SIGNALED;
wr.wr.udmi.remote_addr = remote_address
wr.wr.udmi.rkey = remote_key;

/* send out the request */
ibv_post_send(&p, &wr, &bad_wr);
struct ibw_wc wc2;
ibv_poll_cq(ctx->cq, 2, wc2); /* busy poll until getting completion */
return ctx;
```
All problems in computer science can be solved by another level of indirection

Butler Lampson
All problems in computer science can be solved by another level of indirection.
All problems in computer science can be solved by another level of indirection

except for the problem of too many layers of indirection

– David Wheeler
Main Challenge: How to preserve the performance benefit of RDMA?
Design Principles

1. Indirection only at local for one-sided RDMA
Design Principles

1. Indirection only at local for one-sided RDMA

Diagram:
- Berkeley Socket
- RDMA
- LITE

Layers:
1. Userspace
2. Kernel
3. Hardware
Design Principles

1. Indirection only at local for one-sided RDMA

2. Avoid hardware indirection

Kernel Space
- LITE

Hardware
- RNIC
  - Address mapping
  - Permission check
Design Principles

1. Indirection only at local for one-sided RDMA

2. Avoid hardware indirection

Kernel Space

LITE

Address mapping

Permission check

Hardware

RNIC

Address mapping

Permission check
Design Principles

1. Indirection only at local for one-sided RDMA

2. Avoid hardware indirection

No redundant indirection
Scalable performance
Design Principles

1. Indirection only at local for one-sided RDMA
2. Avoid hardware indirection
3. Hide kernel cost
Design Principles

1. Indirection only at local for one-sided RDMA
2. Avoid hardware indirection
3. Hide kernel cost

eexcept for the problem of too many layers of indirection – David Wheeler
Design Principles

1. Indirection only at local for one-sided RDMA
2. Avoid hardware indirection
3. Hide kernel cost

except for the problem of too many layers of indirection – David Wheeler

Great Performance and Scalability
Outline

• Introduction and motivation
• Overall design and abstraction
  • LITE internals
• LITE applications
• Conclusion
LITE - Architecture

- **OS**
- **LITE Abstraction**
- **Verbs Abstraction**
- **RNIC Driver**

- **Mgmt**
- **User-Level App**
- **Kernel App**
- **User-Level RPC Function**

- **global lkey**
- **global rkey**
LITE - Architecture

- RNIC Driver
  - global lkey
  - global rkey

- User-Level App
  - global rkey
  - addr1
  - addr2

- Kernel App
  - global lkey

- User-Level RPC Function

- LITE 1-Side RDMA
  - lh1
  - lh2
  - Permission check
  - Address mapping

- Mgmt

- OS
  - User-Level App
  - User-Level App

- LITE Abstraction

- Verbs Abstraction
LITE - Architecture

- LITE Abstraction
- Verbs Abstraction

OS

LITE 1-Side RDMA
- LITE 1-Side RDMA
- LH1
- LH2
- Permission check
- Address mapping
- Global lkey
- Global rkey
- Addr1
- Addr2

LITE RPC
- RPC Client
- RPC Server
- Connections
- Queues
- RDMA Buffer Mgmt
- Send
- Poll recv

User-Level RNIC Driver
- User-Level App
- Kernel App
- User-Level RPC Function
- Global lkey
- Global rkey
LITE - Architecture

LITE 1-Side RDMA

- global lkey
- global rkey

Permission check
Address mapping

LITE APIs
- synch
- mgmt
- mem
- RPC
- msging

LITE - Architecture

OS
Kernel App
User-Level App
User-Level RPC
Function

send
poll
recv

Connections
Queues
RPC
Client
RPC
Server
RDMA Buffer
Mgmt

RPC Client
RPC Server
Connections
Queues
send
poll recv

LITE - Architecture

LITE APIs
- synch
- mgmt
- mem
- RPC
- msging
Onload Costly Operations

LITE
- Connections
- Queues
- Keys
- Memory space

OS

RNIC
- Permission check
- Address mapping
Onload Costly Operations

Perform **address mapping** and **protection** in kernel
Avoid Hardware Indirection

Challenge: How to eliminate hardware indirection without changing hardware?
Avoid Hardware Indirection

Challenge: *How to eliminate hardware indirection without changing hardware?*

- Register with **physical address** → no need for any PTEs
Avoid Hardware Indirection

Challenge: *How to eliminate hardware indirection without changing hardware?*

- Register with **physical address** → no need for any PTEs
Avoid Hardware Indirection

Challenge: *How to eliminate hardware indirection without changing hardware?*

- Register with **physical address** → no need for any PTEs
- Register **whole memory** at once → one global key
Avoid Hardware Indirection

<table>
<thead>
<tr>
<th>LITE</th>
<th>OS</th>
<th>RNIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connections</td>
<td>Queues</td>
<td>Keys</td>
</tr>
<tr>
<td>Permission check</td>
<td>Address mapping</td>
<td></td>
</tr>
<tr>
<td>Global lkey</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Challenge: How to eliminate hardware indirection without changing hardware?

- Register with **physical address** → no need for any PTEs
- Register **whole memory** at once → one global key
LITE LMR and RDMA

Userspace application

LITE in Kernel

Remote nodes

Network
LITE LMR and RDMA
LITE LMR and RDMA

LITE in Kernel

Userspace application

Network

Remote nodes
LITE LMR and RDMA

LITE in Kernel

Userspace application

LMR

Node Phy Addr

1 0x45

4 0x27

Remote nodes

Node 1

0x45

Node 4

0x27
LITE LMR and RDMA

LMR

Node | Phy Addr
---|---
1 | 0x45
4 | 0x27

Remote nodes

Network

Userspace application

LITE in Kernel
LITE LMR and RDMA

LITE_read(lh, offset, size)

LMR

Node Phy Addr
1 0x45
4 0x27

Node 1
0x45

Node 4
0x27

Network

Remote nodes

Userspace application

LITE in Kernel
LITE LMR and RDMA

LITE_read(lh, offset, size)

Permission check

QoS

LMR

Node 1

Node 4

Node

Phy Addr

1 0x45

4 0x27

Network

Remote nodes

Userspace application

LITE in Kernel
LITE LMR and RDMA

LITE_read(lh, offset, size)

Permission check
QoS
Offset

LMR

Node | Phy Addr
---|---
1 | 0x45
4 | 0x27

Node 1
0x45

Node 4
0x27

Network
Remote nodes
LITE LMR and RDMA

`LITE_read(lh, offset, size)`

Permissions check

QoS

Userspace application

LITE in Kernel

LMR

Node 1: 0x45

Node 4: 0x27

Remote nodes

Network
LITE LMR and RDMA

LITE in Kernel

Userspace application

Permission check

QoS

LMR

Node | Phy Addr
---|---
1 | 0x45
4 | 0x27

Node 1

Node 4

Network

Remote nodes

LITE_read lh, offset, size
LITE RDMA: Size of MR Scalability

- Write-64B
- LITE_write-64B
- Write-1K
- LITE_write-1K

Requests /us vs Total Size (MB)
LITE RDMA: Size of MR Scalability

![Graph showing the performance of Write-64B, LITE_write-64B, Write-1K, and LITE_write-1K with varying Total Size (MB) and Requests/µs.](image)
LITE RDMA: Size of MR Scalability

LITE scales much better than native RDMA wrt MR size and numbers
LITE-RDMA Latency

Latency (us)

- TCP/IP
- LITE_write user space
- LITE_write kernel space
- Verbs write

Request Size (B)
LITE-RDMA Latency

Latency (us)

Request Size (B)

TCP/IP
LITE_write user space
LITE_write kernel space
Verbs write
LITE-RDMA Latency

Latency (us)

Request Size (B)

- TCP/IP
- LITE_write user space
- LITE_write kernel space
- Verbs write
LITE-RDMA Latency

- Latency (us)
- Request Size (B)
- LITE-RDMA Latency
- TCP/IP
- LITE_write user space
- LITE_write kernel space
- Verbs write

Graph showing latency (us) against request size (B) for different communication modes.
LITE only adds a very slight overhead even when native RDMA doesn’t have scalability issues.
LITE RPC

• RPC communication using two RDMA-write-imm
• One global busy poll thread
• Separate LMRs at server for different RPC clients
• Hide syscall cost behind performance critical path

• Benefits
  – Low latency
  – Low memory utilization
  – Low CPU utilization
Outline

- Introduction and motivation
- Overall design and abstraction
- LITE internals
- **LITE applications**
- Conclusion
LITE Application Effort

<table>
<thead>
<tr>
<th>Application</th>
<th>LOC</th>
<th>LOC using LITE</th>
<th>Student Days</th>
</tr>
</thead>
<tbody>
<tr>
<td>LITE-Log</td>
<td>330</td>
<td>36</td>
<td>1</td>
</tr>
<tr>
<td>LITE-MapReduce</td>
<td>600*</td>
<td>49</td>
<td>4</td>
</tr>
<tr>
<td>LITE-Graph</td>
<td>1400</td>
<td>20</td>
<td>7</td>
</tr>
<tr>
<td>LITE-Kernel-DSM</td>
<td>3000</td>
<td>45</td>
<td>26</td>
</tr>
<tr>
<td>LITE-Graph-DSM</td>
<td>1300</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>

- Simple to use
- Needs no expert knowledge
- Flexible, powerful abstraction
- Easy to achieve optimized performance

*LITE-MapReduce ports from the 3000-LOC Phoenix with 600 lines of change or addition*
MapReduce Results

• LITE-MapReduce adapted from Phoenix [1]

[1]: “Ranger et al., Evaluating MapReduce for Multi-core and Multiprocessor Systems. (HPCA 07)”
MapReduce Results

- LITE-MapReduce adapted from Phoenix [1]

LITE-MapReduce outperforms Hadoop by 4.3x to 5.3x

[1]: “Ranger et al., Evaluating MapReduce for Multi-core and Multiprocessor Systems. (HPCA 07)”
Graph Results

- LITE-Graph built directly on LITE using PowerGraph design
- Grappa and PowerGraph
Graph Results

- LITE-Graph built directly on LITE using PowerGraph design
- Grappa and PowerGraph

LITE-Graph outperforms PowerGraph by 3.5x to 5.6x
Conclusion

- LITE virtualizes RDMA into flexible abstraction
- LITE preserves RDMA’s performance benefits
- *Indirection* not always degrade performance!
Conclusion

- LITE virtualizes RDMA into flexible abstraction
- LITE preserves RDMA’s performance benefits
- *Indirection* not always degrade performance!

- Division across user space, kernel, and hardware
Thank you
Questions?

Get LITE at: https://github.com/Wuklab/LITE

WukLab
wuklab.io