Heterogeneous Communication API

https://github.com/michaelboth/Takyon

“Unification of RDMA, Sockets, and More”

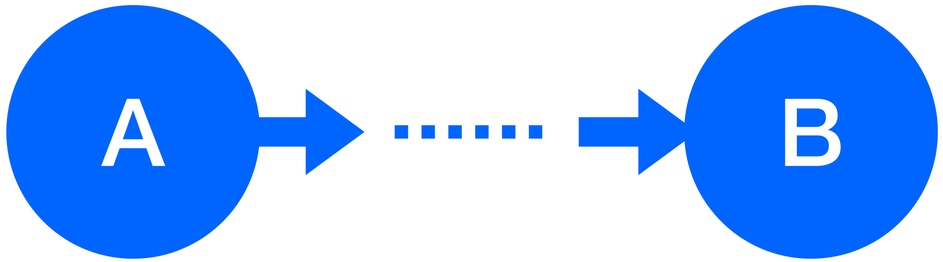
Michael Both: Sep 16 2022

Takyon is a Message Passing API

Connected / Reliable

- Every byte matters
- E.g. downloading, distributed computation

Unconnected / Unreliable



- Unicast & multicast packet streaming, with reconstruction
- Packets may drop, come out of order, or be duplicated
- E.g. audio, video, sensors

Why Create Another API?

Communication is Generally NOT a Primary Focus

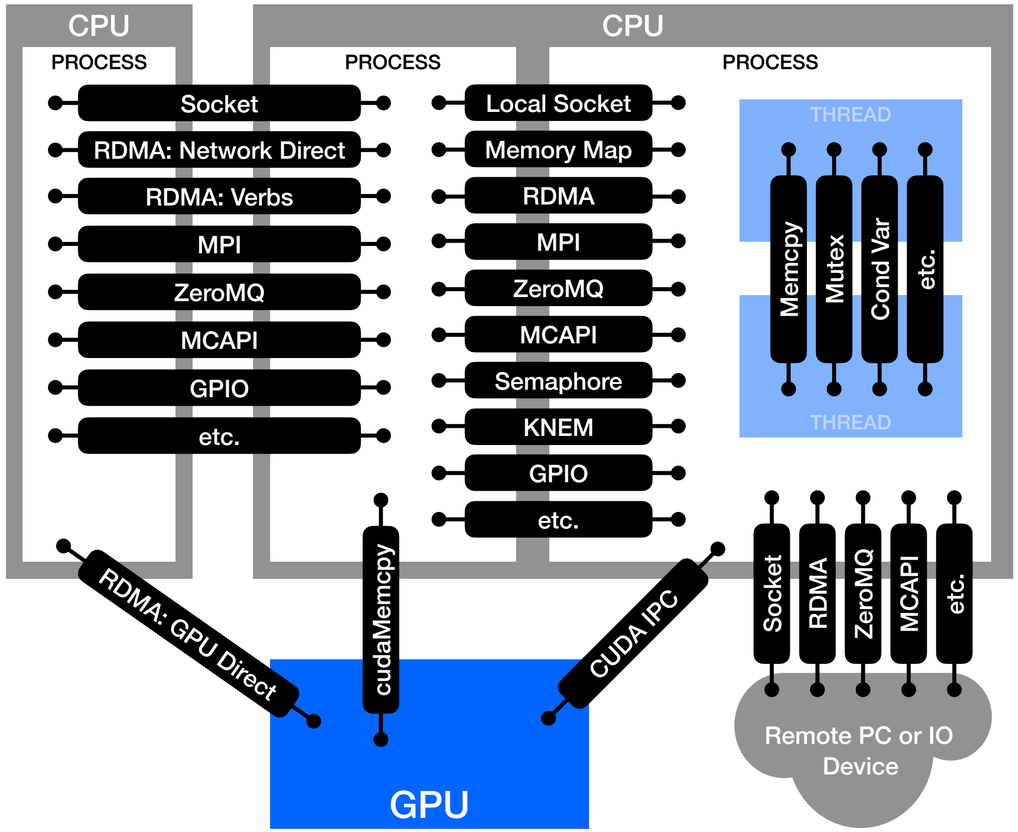
- Primary expertise and development is typically focused on a domain specific technology:
  - Gaming, Signal Processing, AI & Deep Neural Networks, etc.

- A lack of a good heterogeneous communication API commonly leads to various issues:
  - Insufficient time to become an expert with a complex standard
  - Choosing a less capable but easier API
  - Poor performing software that is difficult to maintain
  - Increased hardware and cost to compensate for low performance

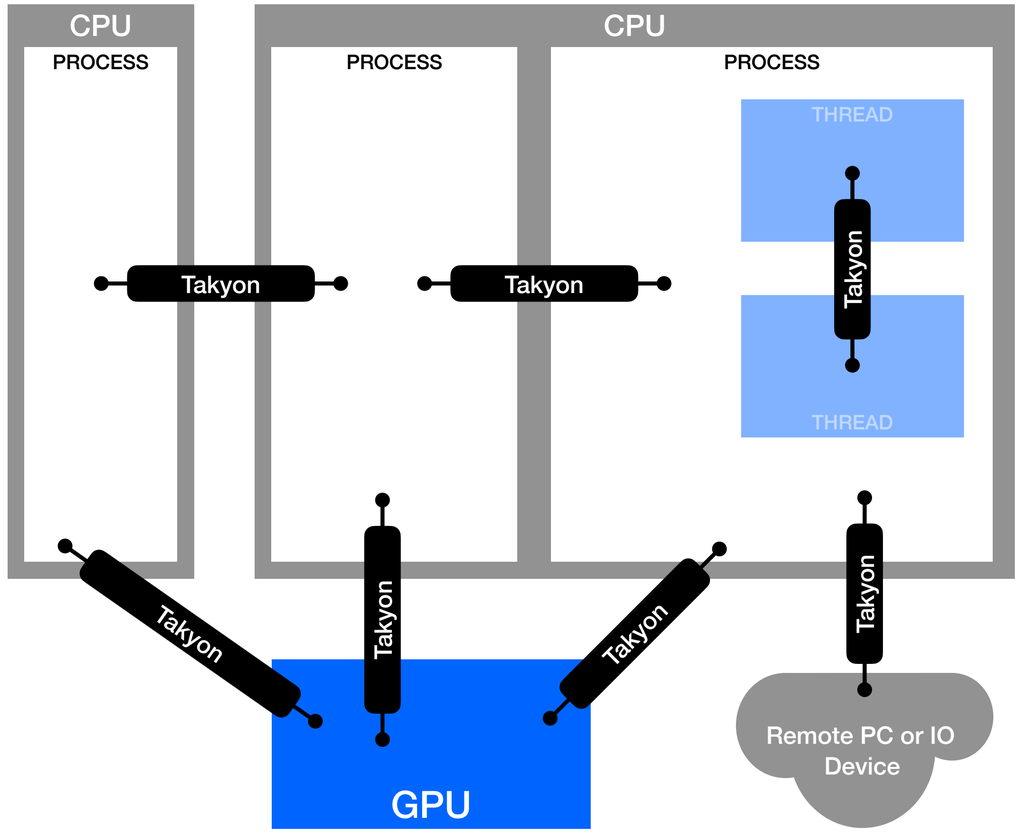
Takyon is based on almost 30 years of experience to solve the critical issues

Problem #1 Solved: A One-Stop Shop

Before Takyon



With Takyon



No single solution to fit all localities

Problem #2 Solved: No Compromised Features

Comparing with common open standard APIs

Feature	Sockets and similar: MCAPI, ZeroMQ	RDMA (OFA Verbs, Network Direct)	MPI	Takyon
Reliable and Unreliable	Yes	Yes		Yes
Communication to external apps, sensors, and other IO devices	Yes	Yes		Yes
Explicit fault tolerant hooks (timeouts, disconnect detection, create/destroy paths on the fly)	Yes	Yes		Yes
Deterministic: avoids implicit communication and allocations	Yes	Yes		Yes
Includes Inter-thread communication				Yes
Non blocking transfers		Yes	Yes	Yes
One way read/write: no involvement from remote endpoint		Yes	Yes	Yes
GPU support		Yes	Yes	Yes
Multiple memory blocks per message		Yes	Yes	Yes
Memory pre-registered before transfer		Yes	Partial	Yes
Zero copy and one-way (i.e. no implicit round trip)		Yes		Yes
32bit piggy back message with main message		Yes		Yes

All features are common with eHPC (embedded High Performance Computing)

Problem #3 Solved: Keep it Simple and Intuitive

Comparing with common APIs

API	Function Count	Typical Drawbacks
Sockets	~20	• Confusing naming and concepts • Need expertise or development may be slow/buggy • May not have required performance due to OS involvement
RDMA (OFA Verbs, Network Direct)	~100	• Very confusing naming and concepts • Experts are very rare • Development will be painfully slow or not time feasible
MPI	~300	• Very feature limited (see previous slide) and may not be feasible • Difficult to understand appropriate transfer model • 'mpirun' is not portable and very confusing to tune
Takyon	8	

SHOUT-OUT: MPI is a great API, but just not well suited for real-time or eHPC

Problem #4 Solved: Enable Best Performance

- Pre-register transport memory when the path is created
  - This is expensive and should not be done at transfer time
  - If the path is connected, then endpoints share memory registrations to allow for one-sided
- Pre-post receive requests
  - This makes sure there is no delay or implicit buffer needed when sending
- The above bullets result in: zero copy, and one-way
  - No round trip needed to coordinate transfers (one-sided or two-sided)
- Non-Blocking
  - Offload transfer from the CPU and OS to allow for efficient concurrent processing and IO

NOTE: Not all interconnects support the above, but Takyon does not inhibit the interconnects that do support the above. Bad example: Trying to wrap RDMA in a socket is limiting

Takyon API

Only 8 Functions!!!

Function	Description
takyonCreate()	Create one endpoint of a communication path
takyonDestroy()	Destroy the endpoint
takyonSend()	Start sending a message If the communication does not support non-blocking then this will block
takyonIsSent()	Complete a non-blocking send
takyonPostRecvs()	If supported, pre-post a list of recv requests before the sender starts sending
takyonIsRecved()	Block until a message arrives
takyonOneSided()	Start a one sided message transfer (read or write)
takyonIsOneSidedDone()	Complete a non-blocking one-sided transfer

All interconnects (RDMA, sockets, etc.) use the same functions

Defining the Interconnect

All interconnects are defined in a text string passed to takyonCreate()

Locality	Example Takyon Interconnects
Inter-Thread	"InterThread -pathID=<non_negative_integer>"
Inter-Process	"InterProcess -pathID=<non_negative_integer>” “SocketTcp -local -pathID=<non_negative_integer>"
Inter-Processor	"SocketTcp -client -remoteIP=<ip_addr> -port=<number>” "SocketTcp -server -localIP=<ip_addr>|Any -port=<number> [-reuse]” "SocketUdpSend -multicast -localIP=<ip_addr> -groupIP=<multicast_ip> -port=<number> [-noLoopback] [-TTL=<time_to_live>]" "SocketUdpRecv -multicast -localIP=<ip_addr> -groupIP=<multicast_ip> -port=<number> [-reuse] [-rcvbuf=<bytes>]” “RdmaRC -client -remoteIP=<ip_addr> -port=<number>” “RdmaRC -server -localIP=<ip_addr>|Any -port=<number> [-reuse]” “RdmaUC -client -remoteIP=<ip_addr> -port=<number> -IBport=<number>” “RdmaUC -server -localIP=<ip_addr>|Any -port=<number> [-reuse] -IBport=<number>” “RdmaUDSend -multicast -localIP=<ip_addr> -groupIP=<multicast_ip>” “RdmaUDRecv -multicast -localIP=<ip_addr> -groupIP=<multicast_ip>”

No limit to the possibilities: GPIO, sensors, FPGAs, etc.

Blocking versus Non-Blocking Transfers

Blocking

Starting a transfer and waiting for the transfer to complete is a single function call

Examples:

send() - blocks until complete

recv() - blocks until complete

Note: recv() can be called before send()

Non-Blocking

Starting a transfer and waiting for the transfer to complete are separate function calls

Examples:

sendStart() - starts transfer

isSent() - block until complete

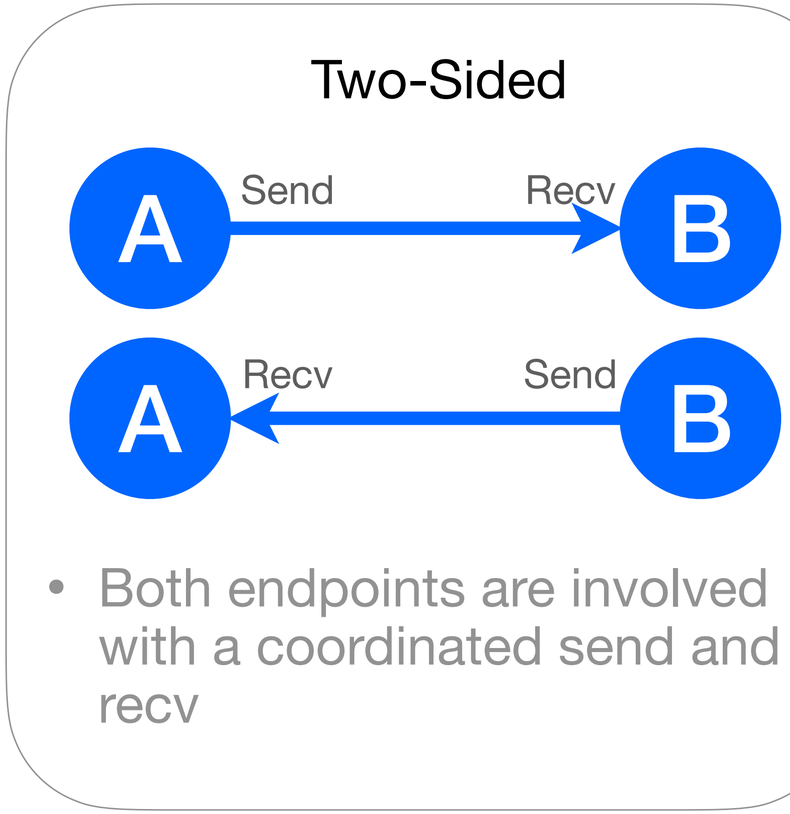
postRecv() - provides a place to recv data ahead of time

isRecved() - blocks until complete

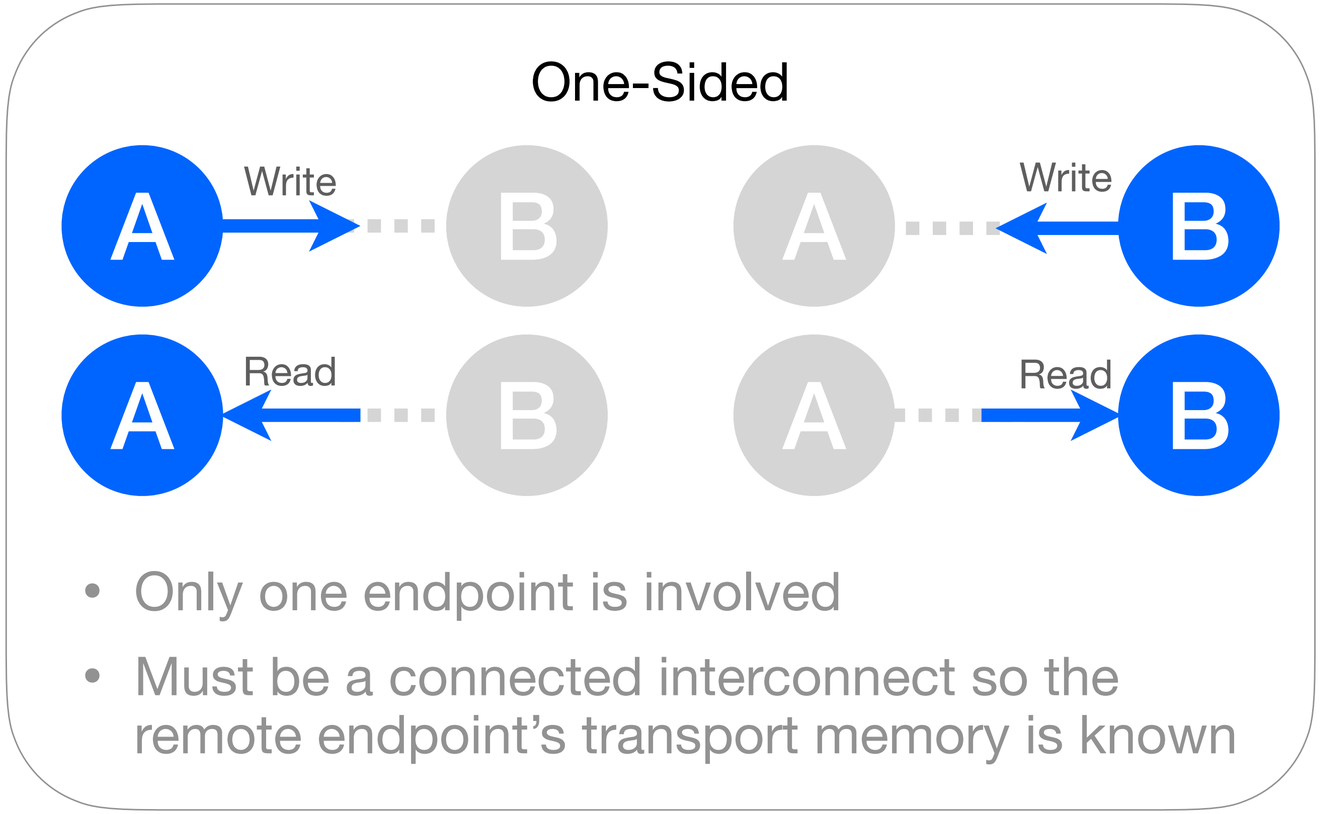
Critical: recv must be posted before send starts

Some interconnects only allow blocking

Two-Sided versus One-Sided Transfers



- Both endpoints are involved with a coordinated send and recv



- Only one endpoint is involved
- Must be a connected interconnect so the remote endpoint's transport memory is known

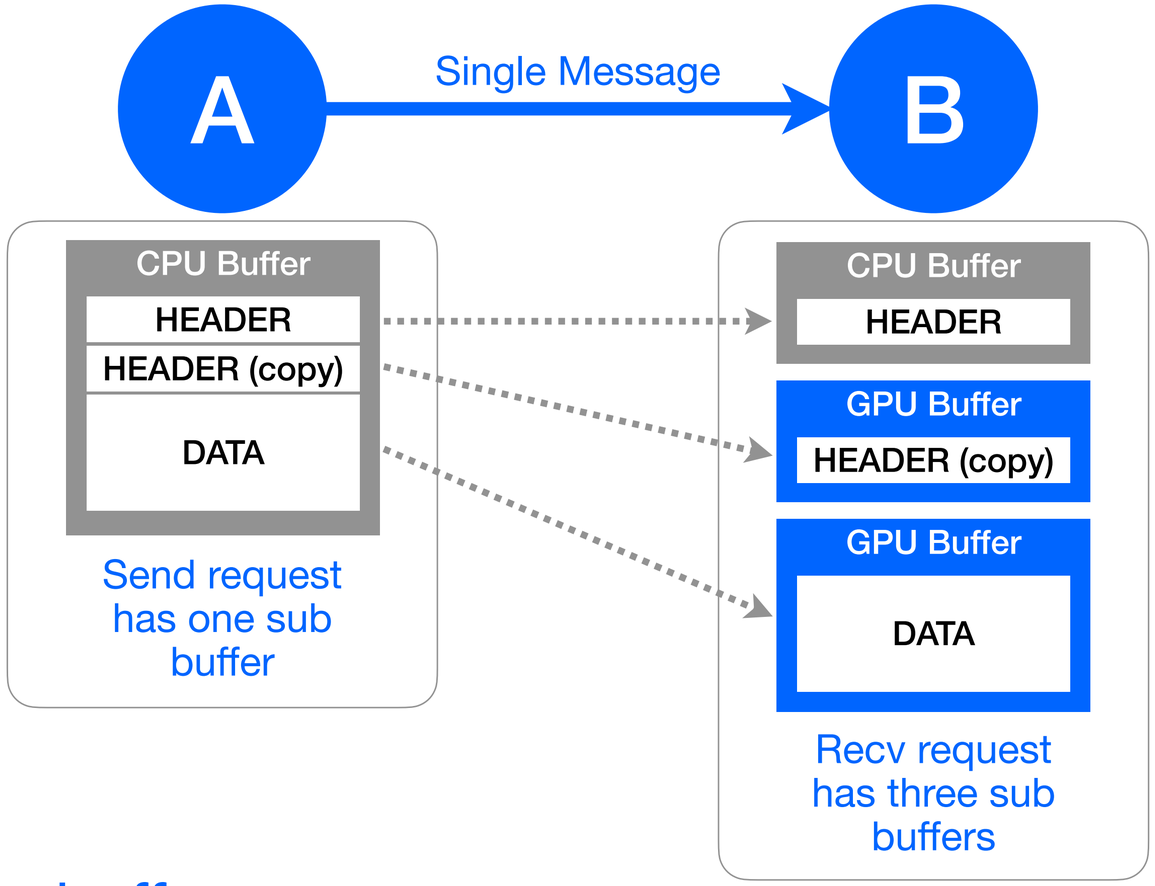
Some interconnects only allow one or the other

Single Message, Multiple Buffers

Multiple buffers may allow for highly optimized processing

Hypothetical Example:

- It's common for GPUs to do heavy processing and CPU does light book keeping, but both need to know the attributes of the data



Some interconnects only allow one buffer per message

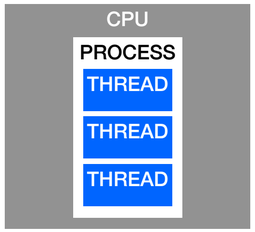
Fault Tolerant Communication

Detecting and handling degraded communication paths

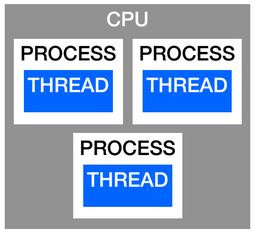
- Detecting degraded communication
  - Disconnect detection
  - Timeouts
- Handling degraded communication
  - Destroy and re-create path using the same interconnect
  - Use an alternative interconnect and path
- Communication API Requirements
  - Communication API should provided the hooks for fault tolerance
  - Only the app can know what to do when communication degrades
  - Communication paths should be independent of each other (“One light goes out they all go out”)

DISTINCTION: Takyon is not fault tolerant (and it shouldn’t be), but does provide fault tolerant hooks

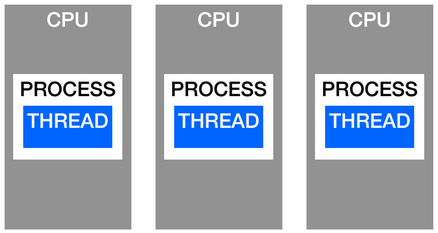
Takyon Enables Accelerating Development by Staging



1. Start with one process and multiple threads
   - While dataflow is being developed, only need to run a single executable
   - Easier to debug crashes or validate memory leaks/overwrites (e.g. valgrind)



2. Move to multiple processes on one CPU
   - Simple way to validate the migration of dataflow, without jumping to multi-processors



3. Move to multiple processors
   - Migration should be simple
   - Can now test for performance

Could also have a blend of inter-thread, inter-process, and inter-processor

Looking to the Future

Posible Enhancements

- Strided Transfers
  - Currently avoiding this since common interconnects don’t support this
- Publish/Subscribe
  - A potential replacement for the overly complex DDS
  - Could have simplified participants, publishers, subscribers, and QoS
  - Make messages opaque and private (removes need for DDS’s intermediate language)
- Collectives: barrier, scatter, gather, all-to-all, reduce, etc.
  - Already done as a separate API with Takyon 1.x, and I will convert it to Takyon 2.x
  - Create a complimenting GUI to build and maintain the collective groups visually

CHALLENGE: Is Takyon missing a key feature?

Ultimate Goal: It's Almost an Open Standard

KHRONOS® GROUP

CONNECTING SOFTWARE TO SILICON

The creation of Takyon inspired a Khronos exploratory group that determined the industry is in need of a better Heterogeneous Communication API:

https://www.khronos.org/exploratory/heterogeneous-communication/

Takyon is currently the only proposal and is waiting for an industry quorum (4 or 5 unique companies) to move forward with the working group that will formalize the specification. It's very difficult finding the expertise who can contribute.

NEED HELP: Join the Khronos Working Group to help get the specification moving forward