Generating High-Performance Communication Kernels

Madan Musuvathi

Peng Cheng, Changho Hwang, Abhinav Jangda, Binyang Li, Saeed Maleki, Youshan Miao, Jacob Nelson, Olli Saarikivi, Aashaka Shah, Yifan Xiong, Yongqiang Xiong, Fan Yang, Ziyue Yang

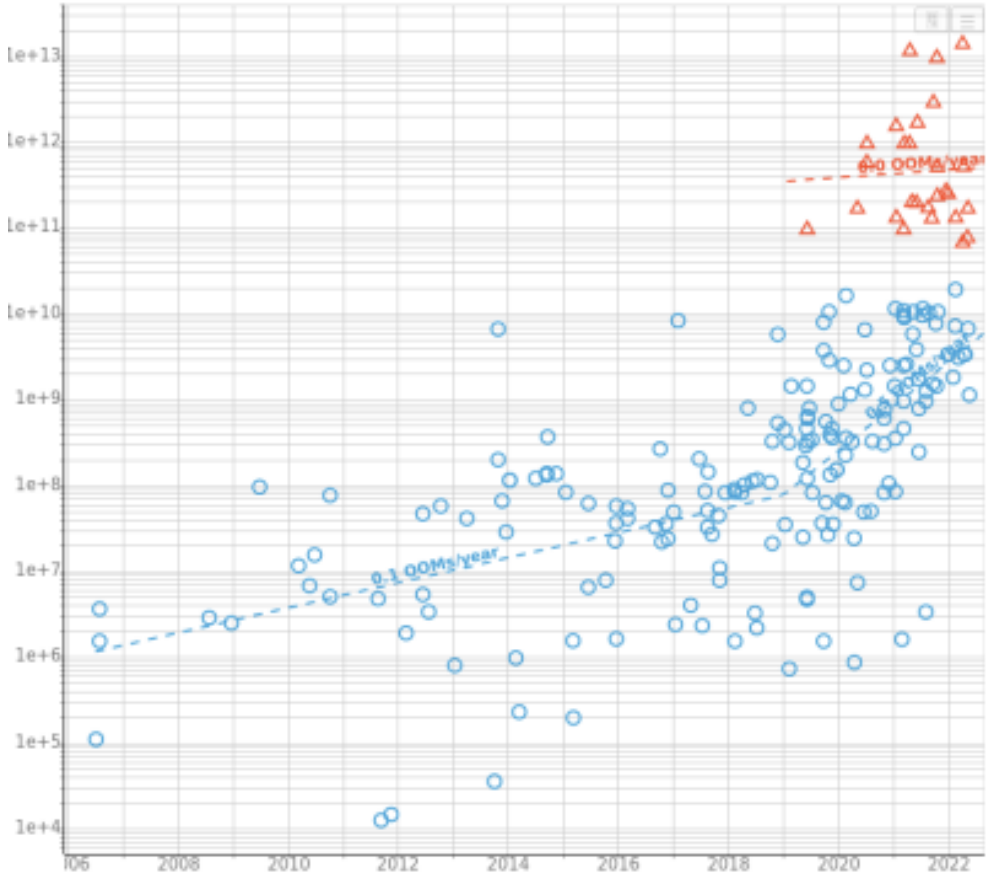
[Microsoft Research]

Meghan Cowan, Todd Mytkowicz [Google]

Vijay Chidambaram [UT Austin], Rachee Singh [Cornell], Zixian Cai [ANU], Zhengyang Liu [Utah]

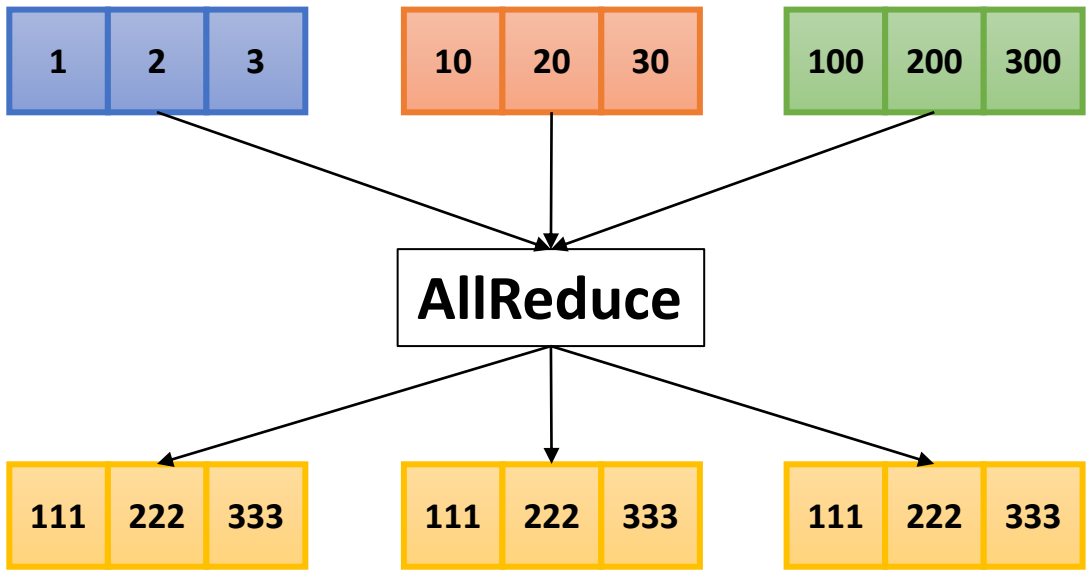
ML computations are increasingly communication bound

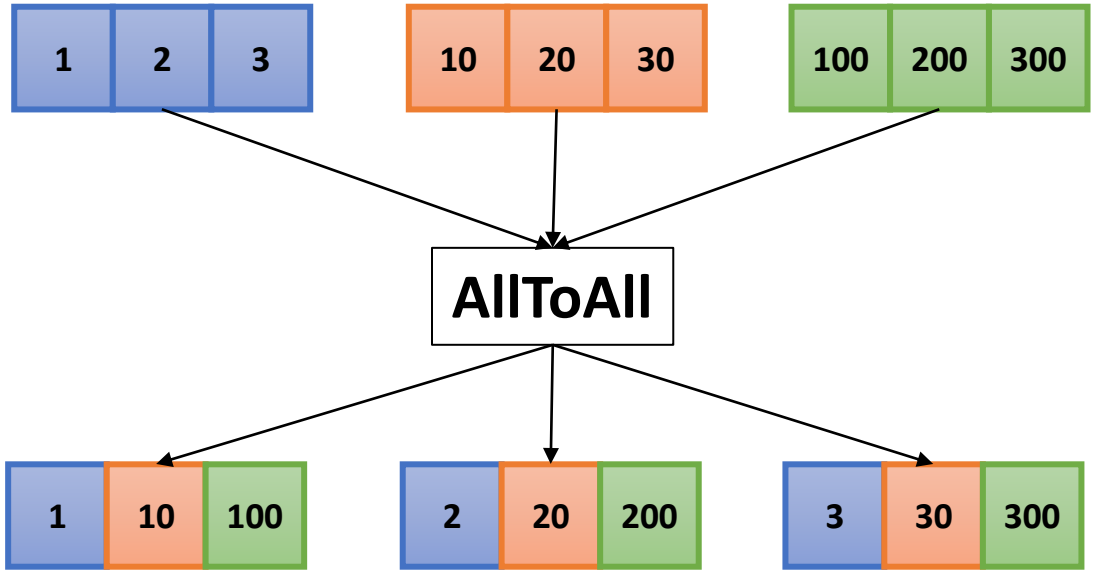
- Model sizes increasing ~10× / year
- Need to distribute computations for both inference and training

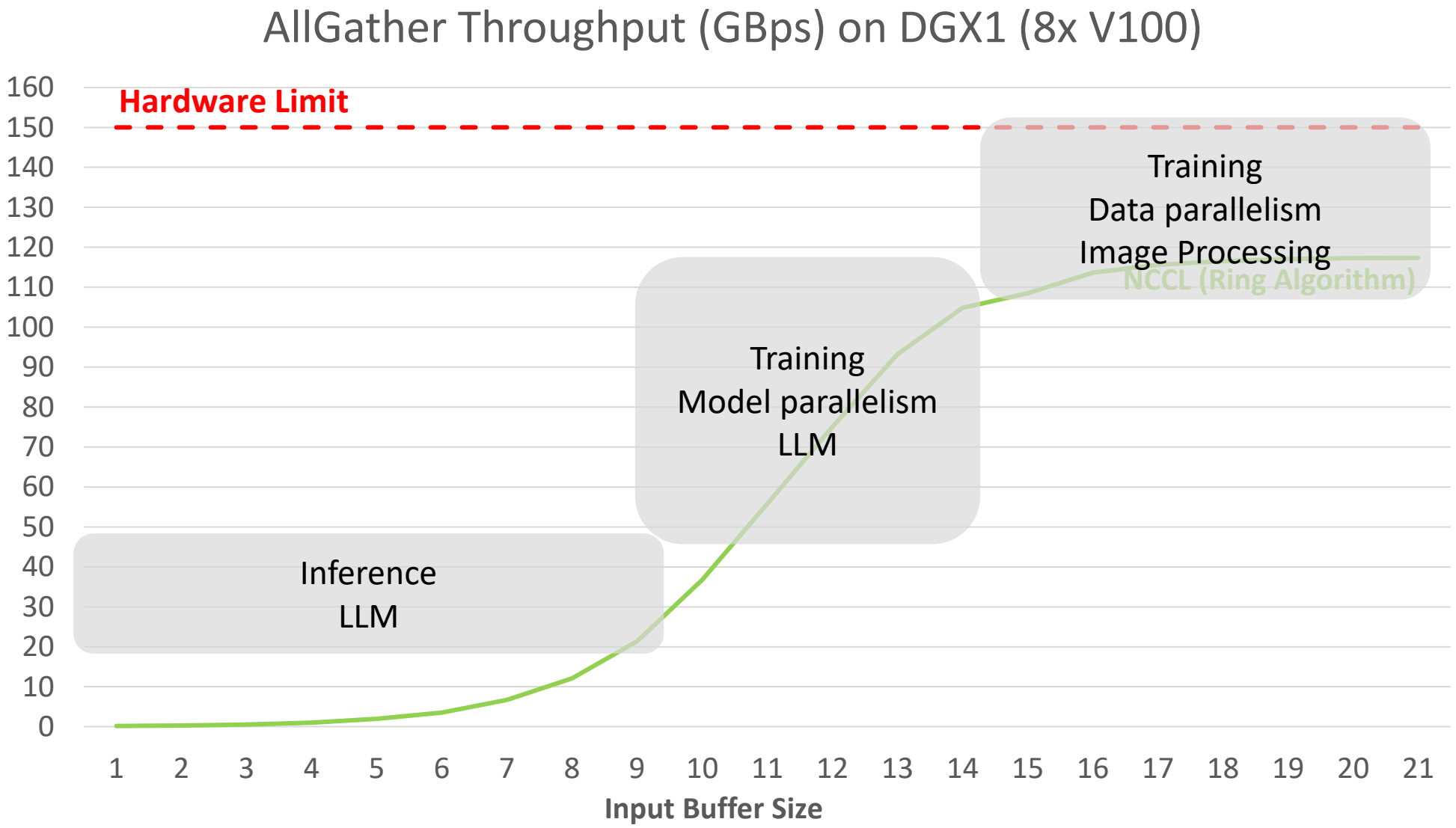


Villalobos et al. ‘22

Collective Communication

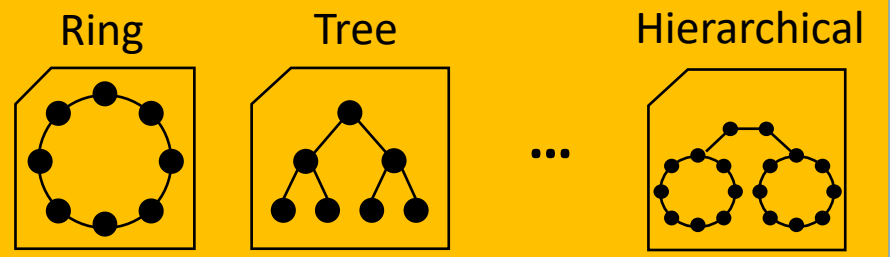





AllGather Throughput (GBps) on DGX1 (8x V100)
Hardware Limit
Training
Data parallelism
Image Processing
NCCL (Ring Algorithm)
Training
Model parallelism
LLM
Inference
LLM
160
150
140
130
120
110
100
90
80
70
60
50
40
30
20
10
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
Input Buffer Size

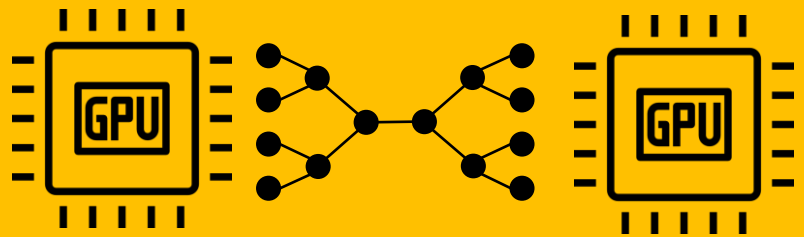
Implementing Communication Kernels

Algorithmic choices



Route data
Distribute compute
Latency vs bandwidth
Intra- vs inter- node

Implementation choices

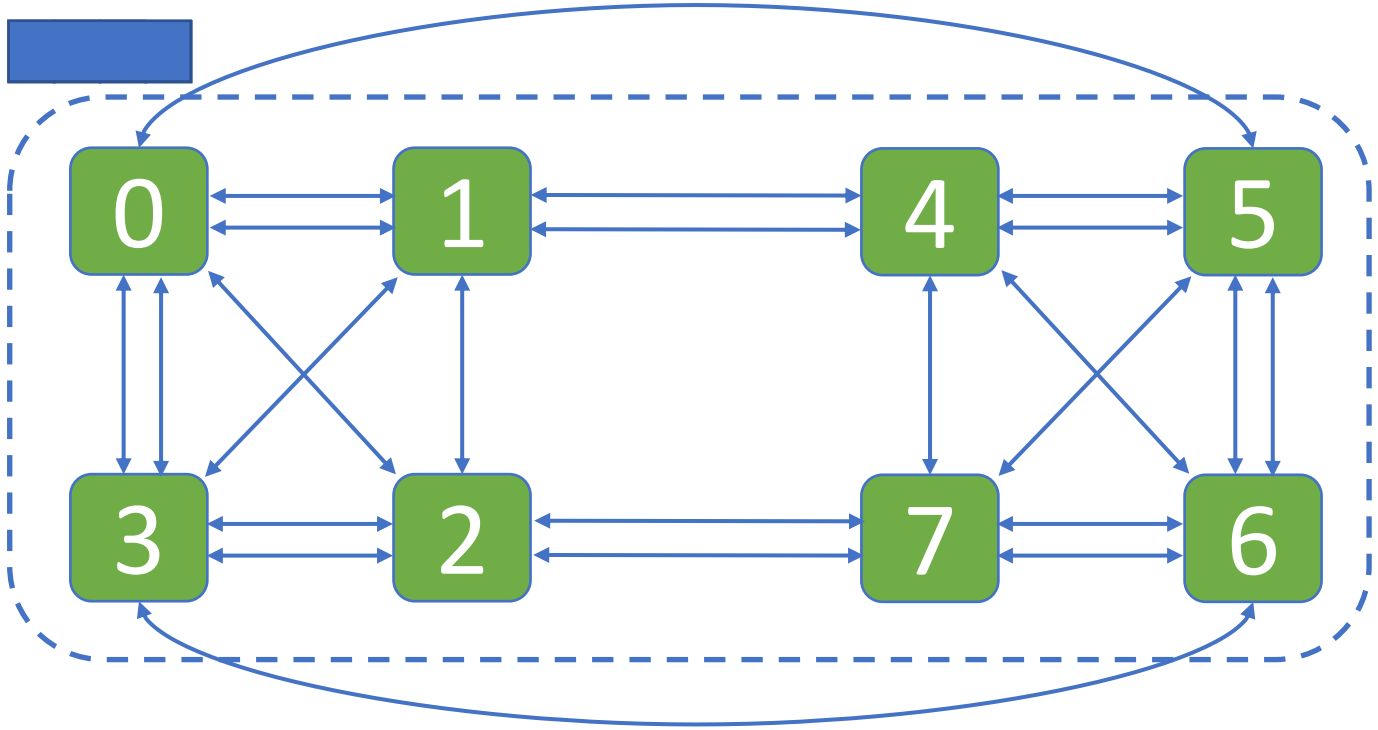


Maximize parallelism
Maximize link utilization
Minimize launch + synch overheads
Deal with relaxed consistency, deadlocks, …

image: Flaticon.com

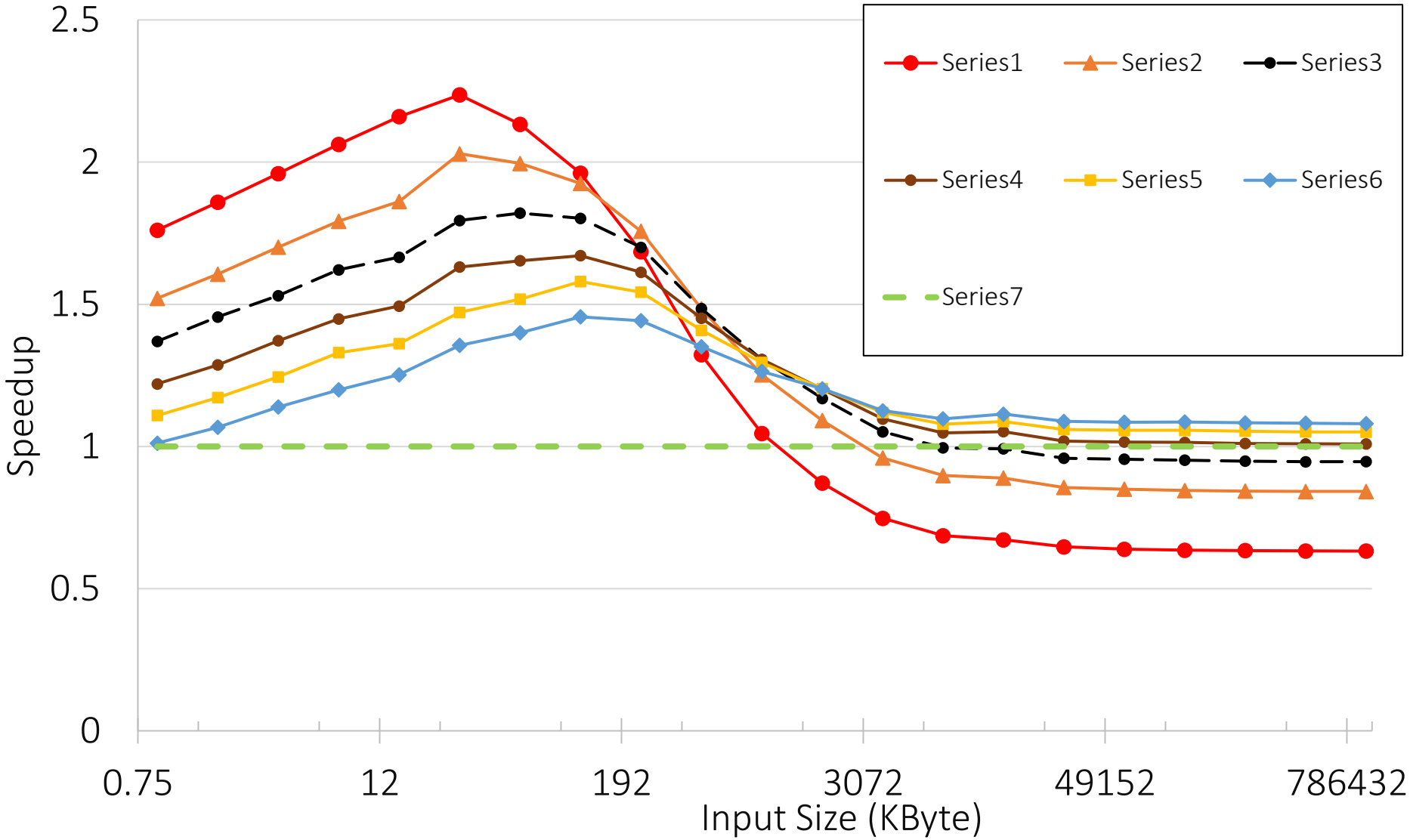
Synthesizing Optimal Collective Algorithms [PPoPP ‘21]

Pareto optimality: minimize latency (bandwidth) for a given bandwidth (latency) cost



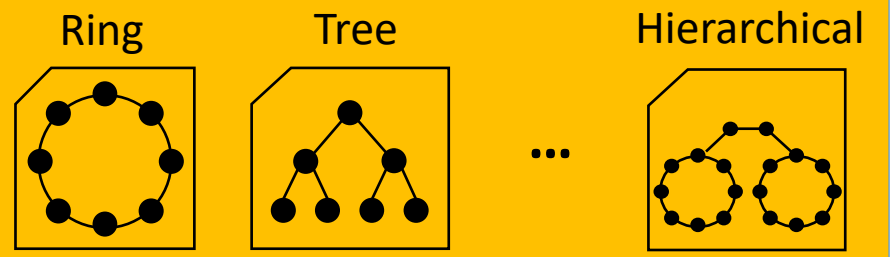
- Split input into c chunks
- Bound routing steps s
- Search (with z3) for algorithms

- Pareto frontier can be generated by minimizing s for various c


2.5
2
1.5
1
0.5
0
Speedup
0.75
12
192
3072
49152
786432
Input Size (KByte)
Series1
Series2
Series3
Series4
Series5
Series6
Series7

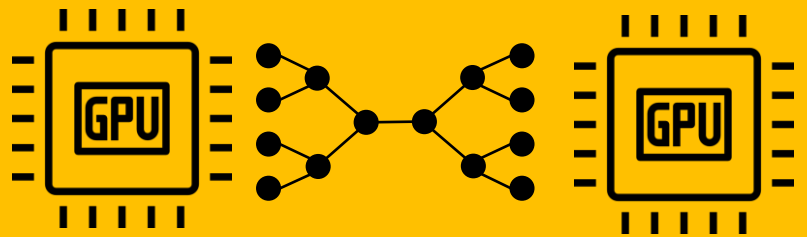
Implementing Communication Kernels

Algorithmic choices



Route data
Distribute compute
Latency vs bandwidth
Intra- vs inter- node

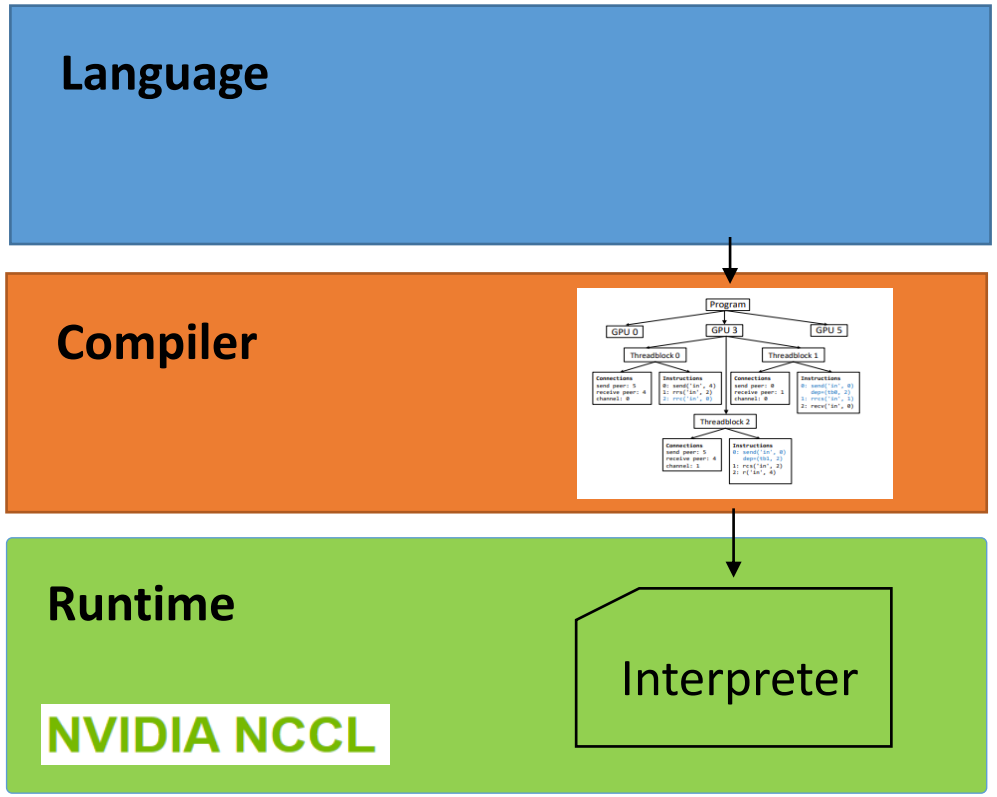
Implementation choices



Maximize parallelism
Maximize link utilization
Minimize launch + synch overheads
Deal with relaxed consistency, deadlocks, …

image: Flaticon.com

MSCCL [ASPLOS '23]



Pythonic chunk-oriented DSL

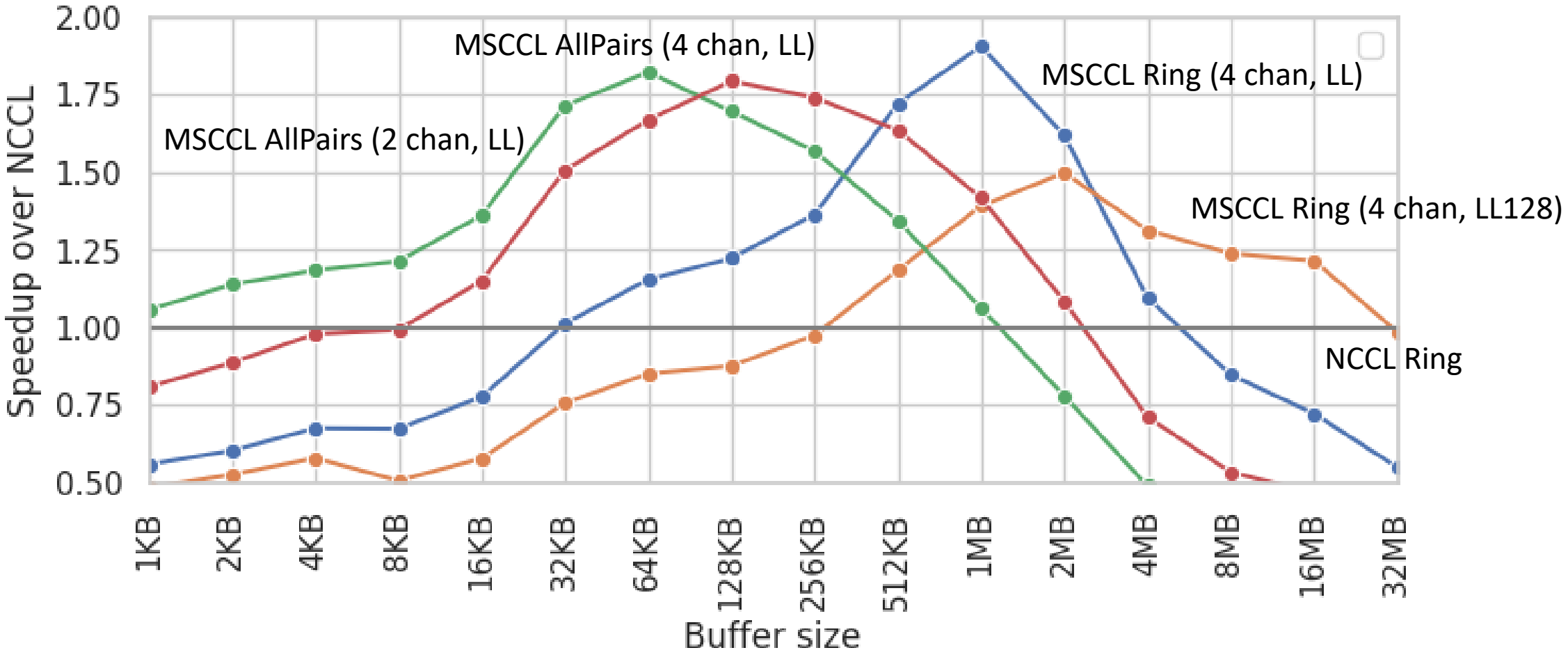
```
# N: number of nodes
# G: number of GPUs per node
def alltoall(N, G):
    for n in range(N):
        for g in range(G):
            for m in range(N):
                for i in range(G):
                    c = chunk((m,i), 'in', (n,g))
                    if m == n:
                        c.copy((n,g), 'out', (m,i))
                    else:
                        c.copy((m,g), 'sc', (n,i))

            # Coalesced IB send
            c = chunk((m,g), 'sc', n*G, sz=G)
            c.copy((n,g), 'out', m*G)
```

Map chunk-oriented comp. to threads blocks
Parallelization/tiling/scheduling optimizations
Ensure correctness (no deadlocks/data races)

Embedded interpreter in NCCL
Reuse NCCL transport mechanisms
API compatible with NCCL

All-Reduce on 8xA100



MSCCL

microsoft / msccl Public

Microsoft Collective Communication Library

View license

113 stars 17 forks