heFFTe

Highly Efficient FFT for Exascale

ICL Lunch talk – October 18th 2019

Alan Ayala, Stan Tomov, Azzam Haidar, Daniel Schultz,
Hejer Shaiek, Jack Dongarra

SUMMARY

- 1. Introduction, software stack
- 2. Algorithm design and optimization
- 3. Numerical experiments
- 4. Communication model and bottleneck
- 5. Further optimizations and future work

First Release

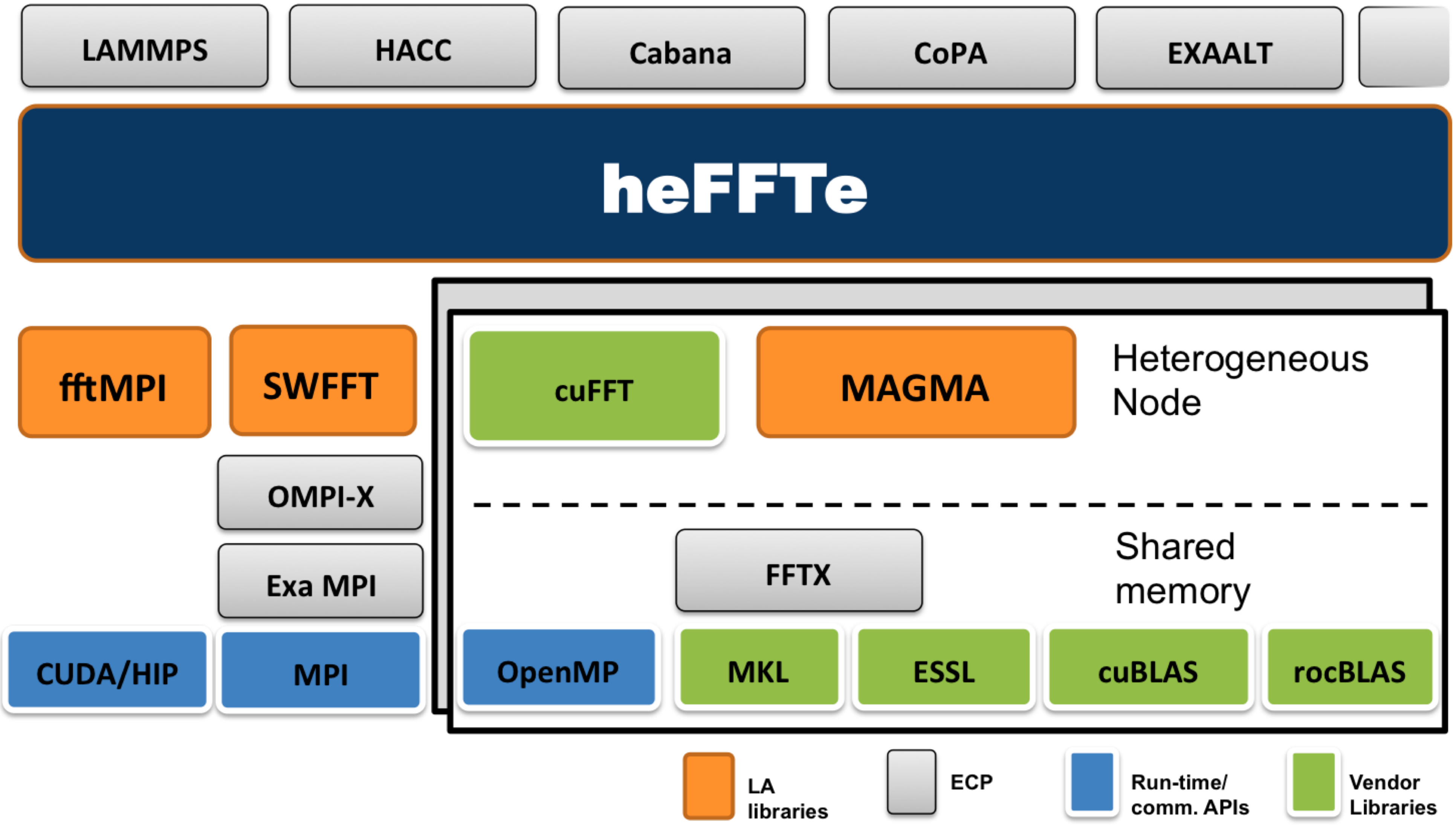
- We released the ECP-FFT FFT library, heFFTe version 0.1 on September 30th.

  https://bitbucket.org/icl/heffte/

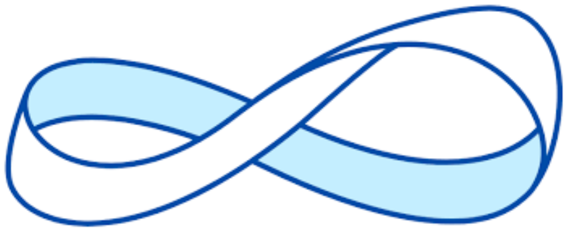
- Currently we support:
  - 3D FFTs on double and single precision.
  - CPU and GPU kernels for FFT computation and tensor transposition.
  - A single library for all these options.

- More than a dozen ECP applications use FFT in their codes, *e.g.* *EXAALT.*

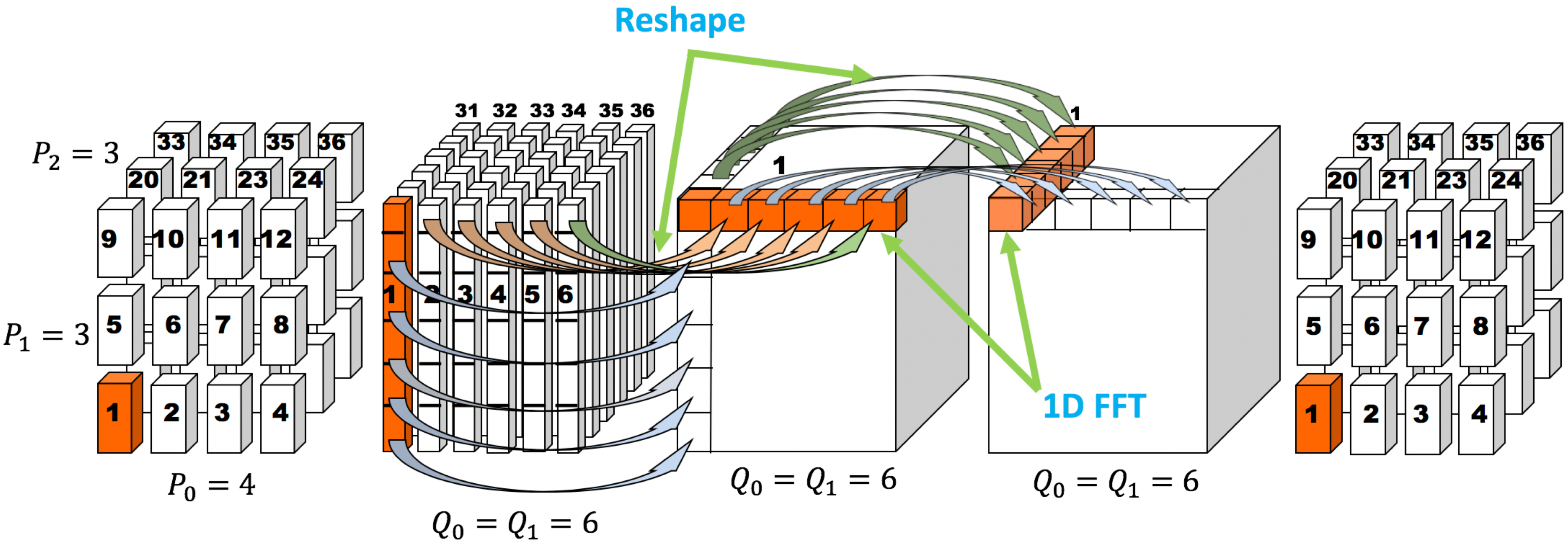
- Over 70 scientific software packages depend on and use FFTs, according to Spack package manager.


LAMMPS
HACC
Cabana
CoPA
EXAALT
heFFTe
fftMPI
SWFFT
OMPI-X
Exa MPI
CUDA/HIP
MPI
cuFFT
MAGMA
Heterogeneous Node
FFTX
Shared memory
OpenMP
MKL
ESSL
cuBLAS
rocBLAS
LA libraries
ECP
Run-time/ comm. APIs
Vendor Libraries
ICL
THE UNIVERSITY OF TENNESSEE KNOXVILLE

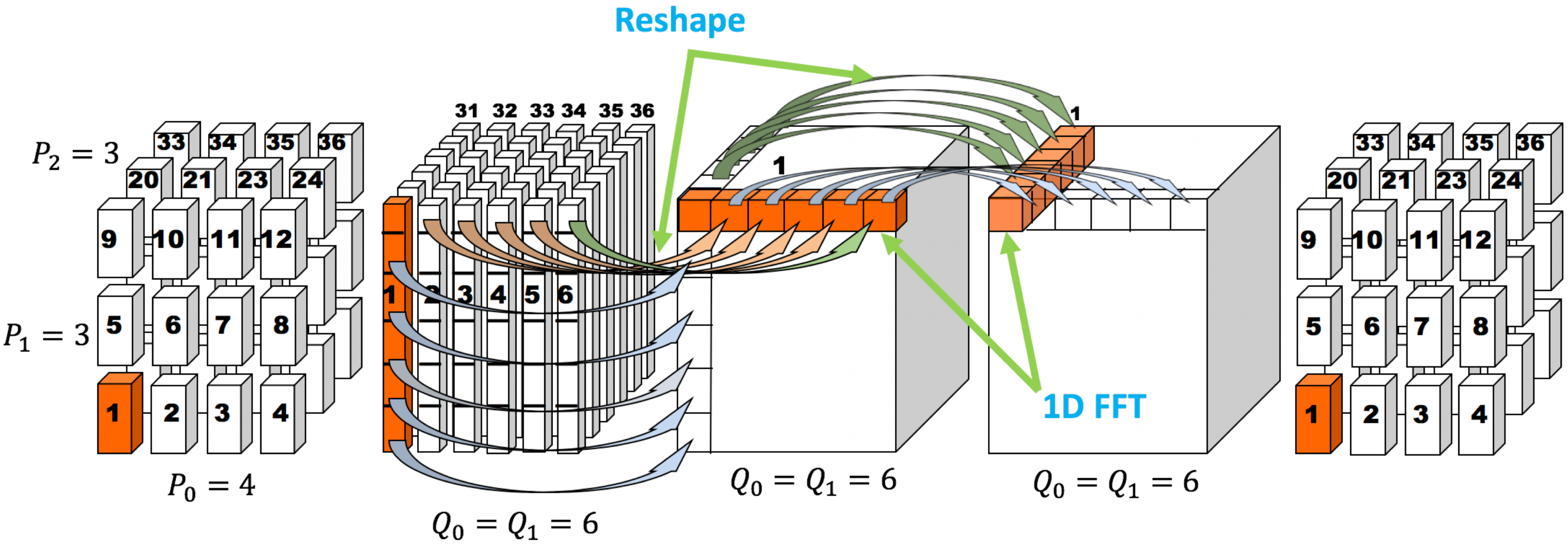
2. ALGORITHM DESIGN AND OPTIMIZATION



3D FFT Algorithm

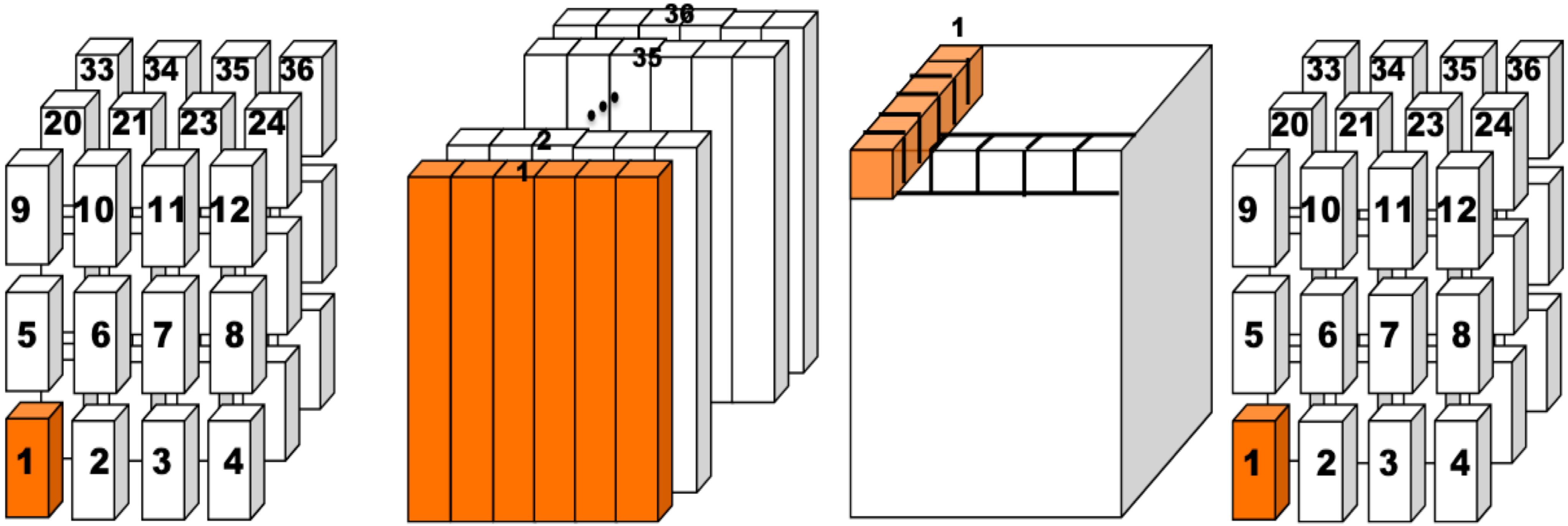


3D FFT Algorithm



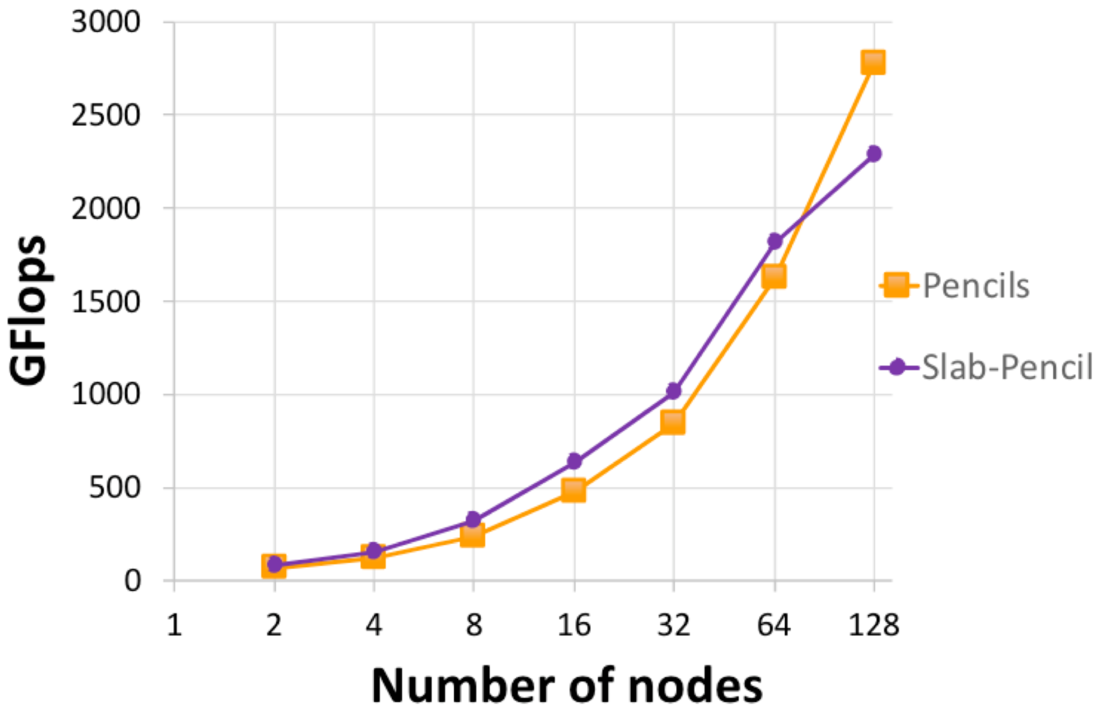
- Reshape process can be seen as a tensor transposition.
- HEFFTE leverages third-party 1D FFTs from vendors or open-source libraries (FFTW3, CUFFT, MKL)

3D FFT Algorithm (SLAB version)



- Using 2D FFTs and then 1D FFT (which can be done by FFTW3, CUFFT, MKL)

3D FFT Algorithm (SLAB version)



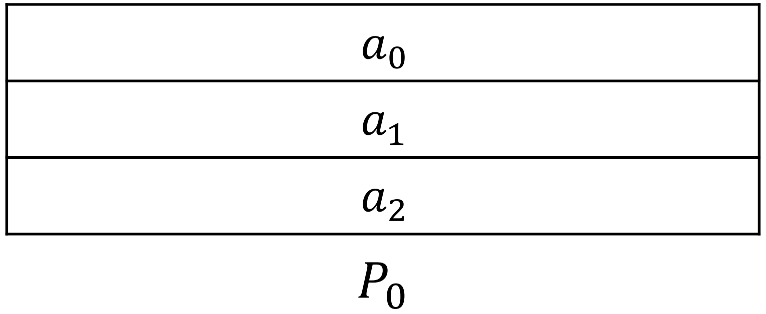
For large node count, the pencil approach is faster.

- Using 2D FFTs and then 1D FFT (which can be done by FFTW3, CUFFT, MKL)
- SLAB version algorithm is also supported by heFFTe

Function: heffte_reshape

1. Data processing, **packing**:

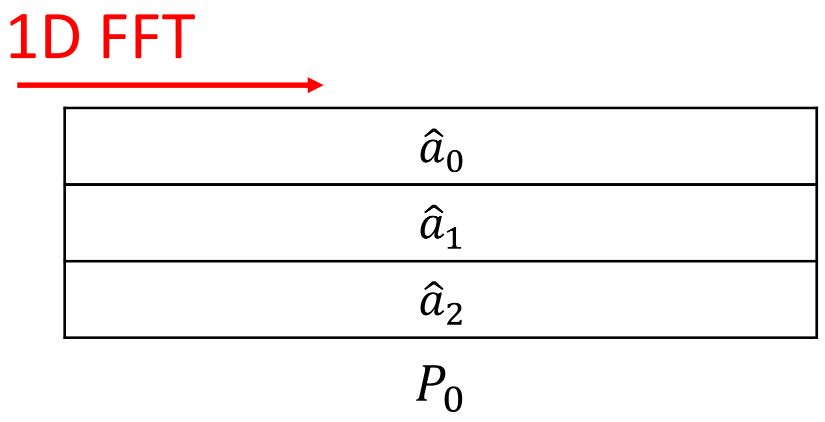
Consider 3 processes having data in pencil format ready for 1D FFTs.



Function: heffte_reshape

1. Data processing, **packing**:

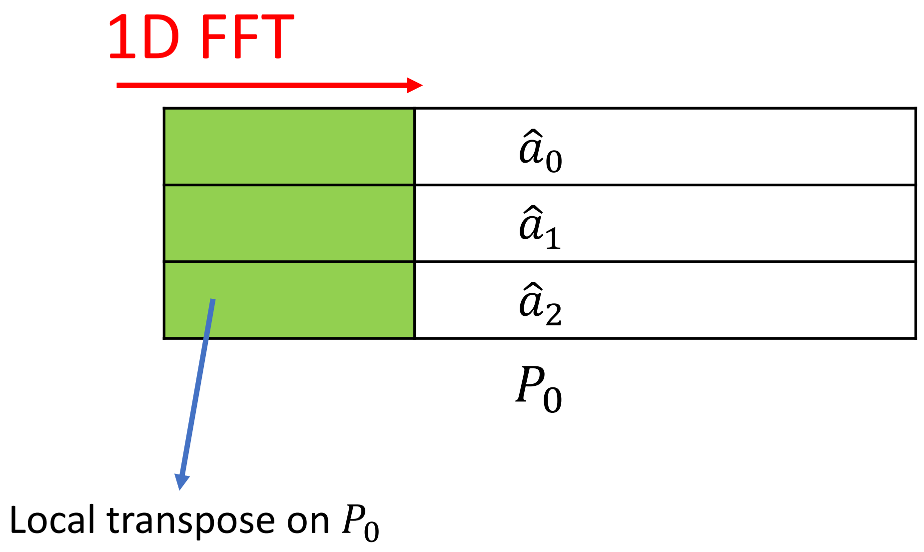
Consider 3 processes having data in pencil format ready for 1D FFTs.



Function: heffte_reshape

1. Data processing, **packing**:

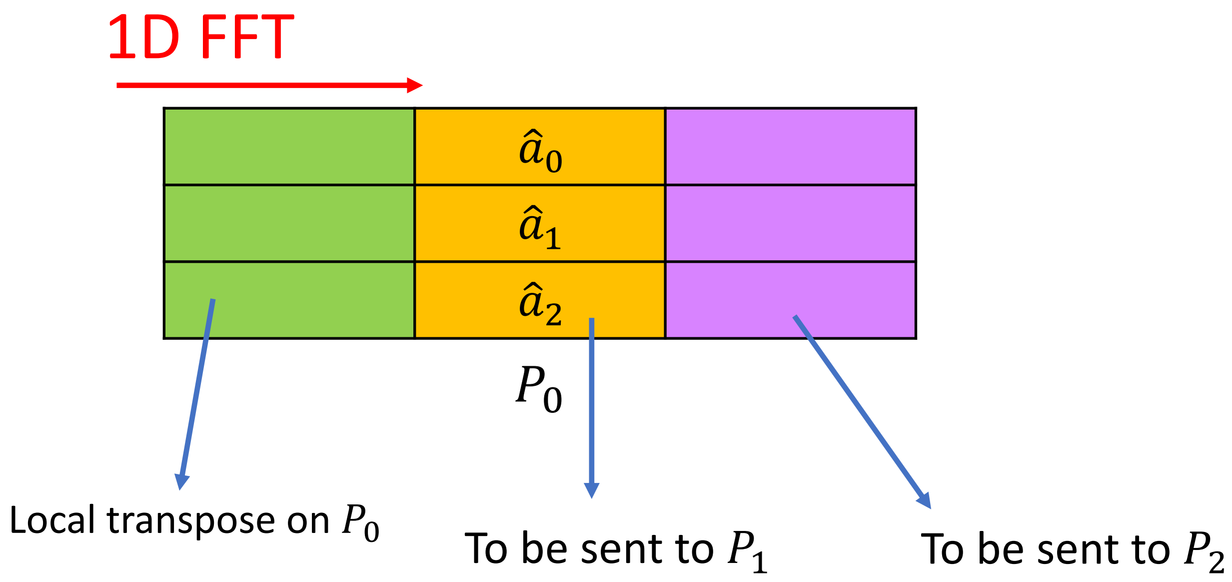
Consider 3 processes having data in pencil format ready for 1D FFTs.



Function: heffte_reshape

1. Data processing, **packing**:

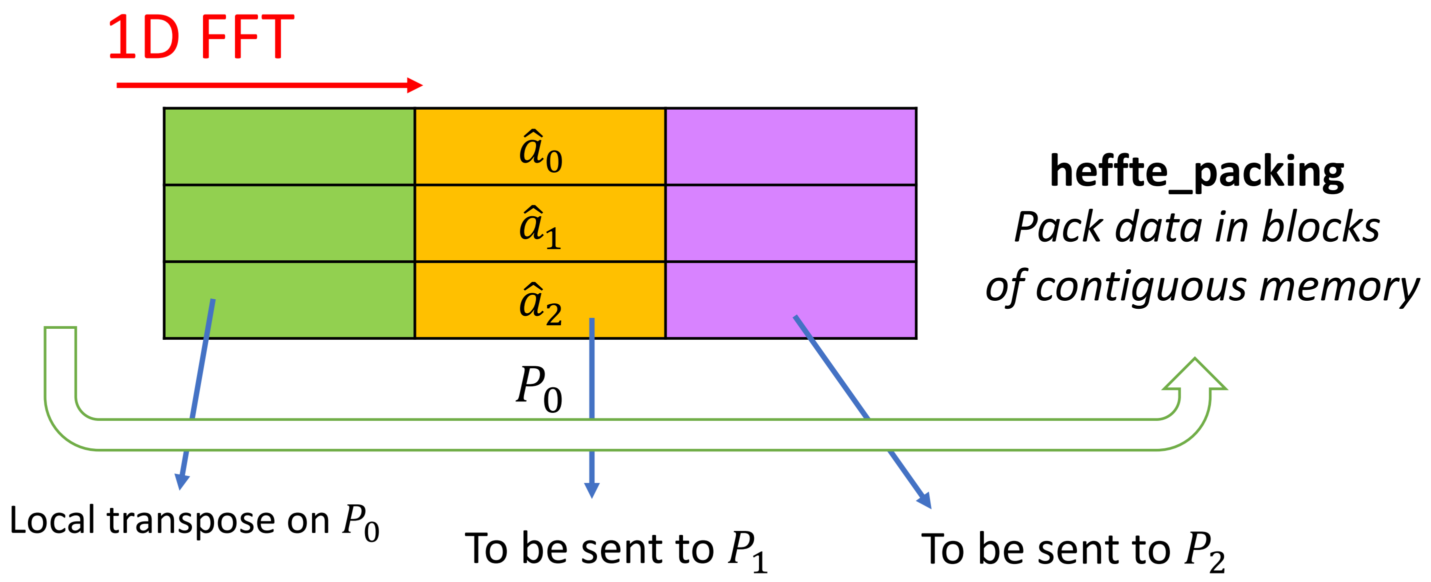
Consider 3 processes having data in pencil format ready for 1D FFTs.



Function: heffte_reshape

1. Data processing, **packing** :

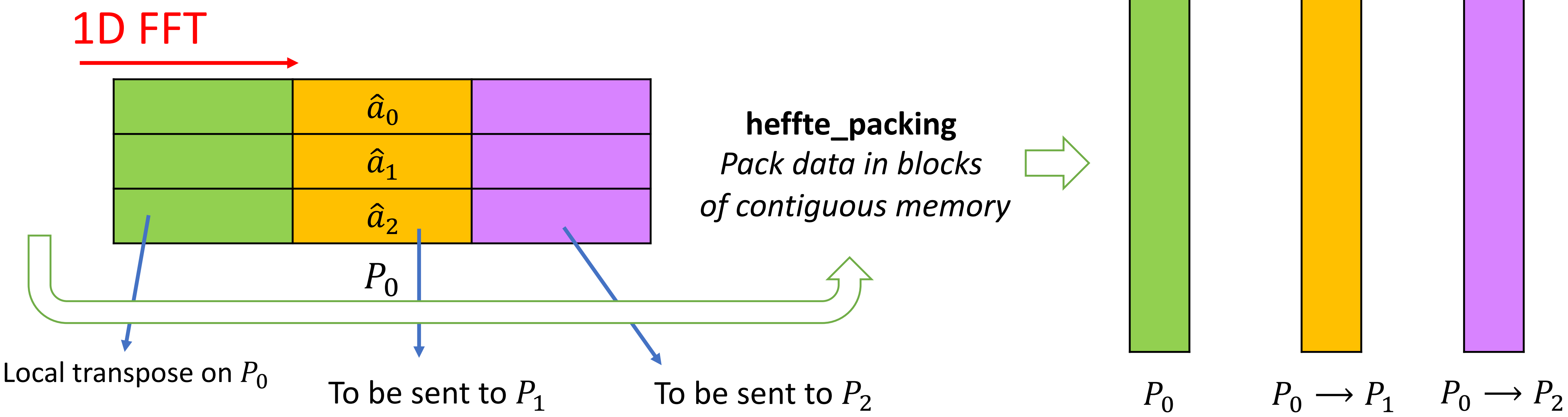
Consider 3 processes having data in pencil format ready for 1D FFTs.



Function: heffte_reshape

1. Data processing, **packing** :

Consider 3 processes having data in pencil format ready for 1D FFTs.



Function: heffte_reshape

2. Multi-process (Multi-GPU) communication:

   Can be done with Point-to-Point or All-to-all communication.

   This step is the bottleneck for parallel FFT libraries.

Function: heffte_reshape

2. Multi-process (Multi-GPU) communication:

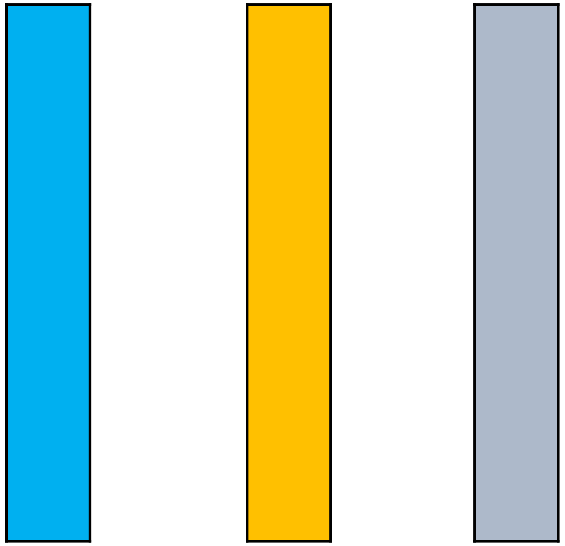
   Can be done with Point-to-Point or All-to-all communication.
   This step is the bottleneck for parallel FFT libraries.

Libraries	Point-to-point routines		Collective routines		Process Topology
	Blocking	Non-blocking	Blocking	Non-blocking	
FFTMPI	MPI_Send	MPI_Irecv	MPI_Alltoallv	None	MPI_Comm_create
			MPI_Allreduce		MPI_Group
			MPI_Barrier		
SWFFT	MPI_Sendrecv	MPI_send	MPI_Allreduce	None	MPI_Cart_create
		MPI_Irecv	MPI_Barrier		MPI_Cart_sub
HEFFTE	MPI_Send	MPI_Isend	MPI_Alltoallv	Magma_Alltoallv	MPI_Comm_create
	MPI_Recv	MPI_Irecv	MPI_Allreduce		MPI_Group
			MPI_Barrier		

Function: heffte_reshape

3. Data processing, **unpacking** :

Consider process P_1 is receiving data from other processes.

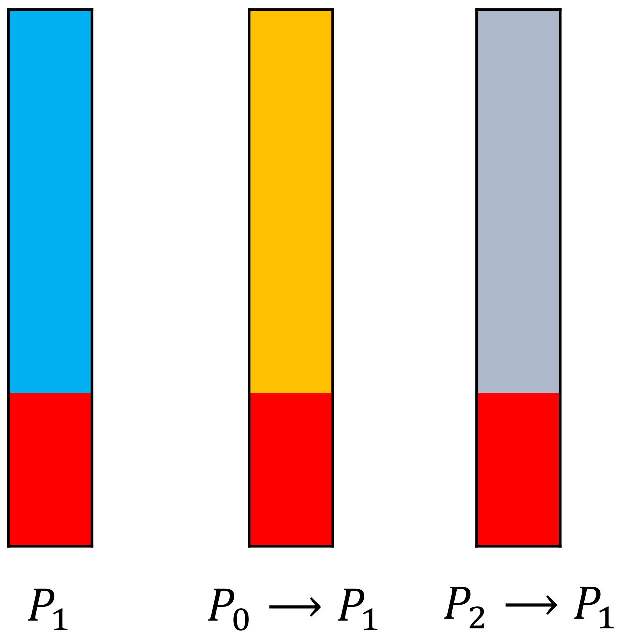


P_1 $\quad$ $P_0 \longrightarrow P_1$ $\quad$ $P_2 \longrightarrow P_1$

Function: heffte_reshape

3. Data processing, **unpacking** :

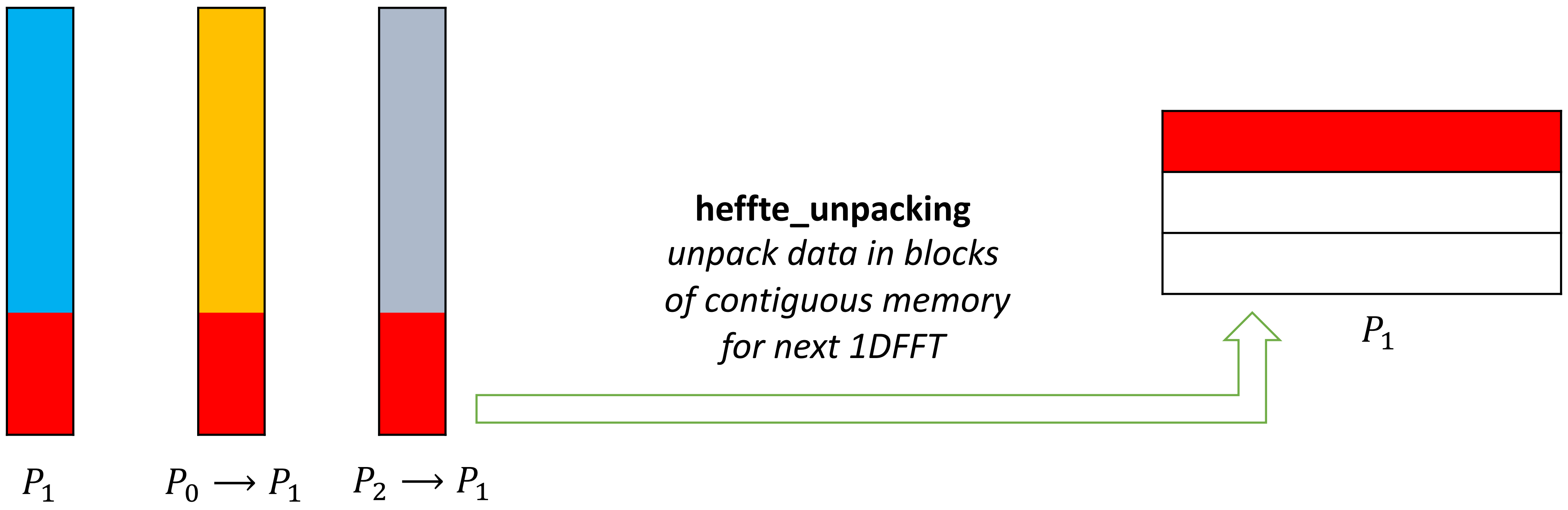
Consider process P_1 is receiving data from other processes.



Function: heffte_reshape

3. Data processing, **unpacking** :

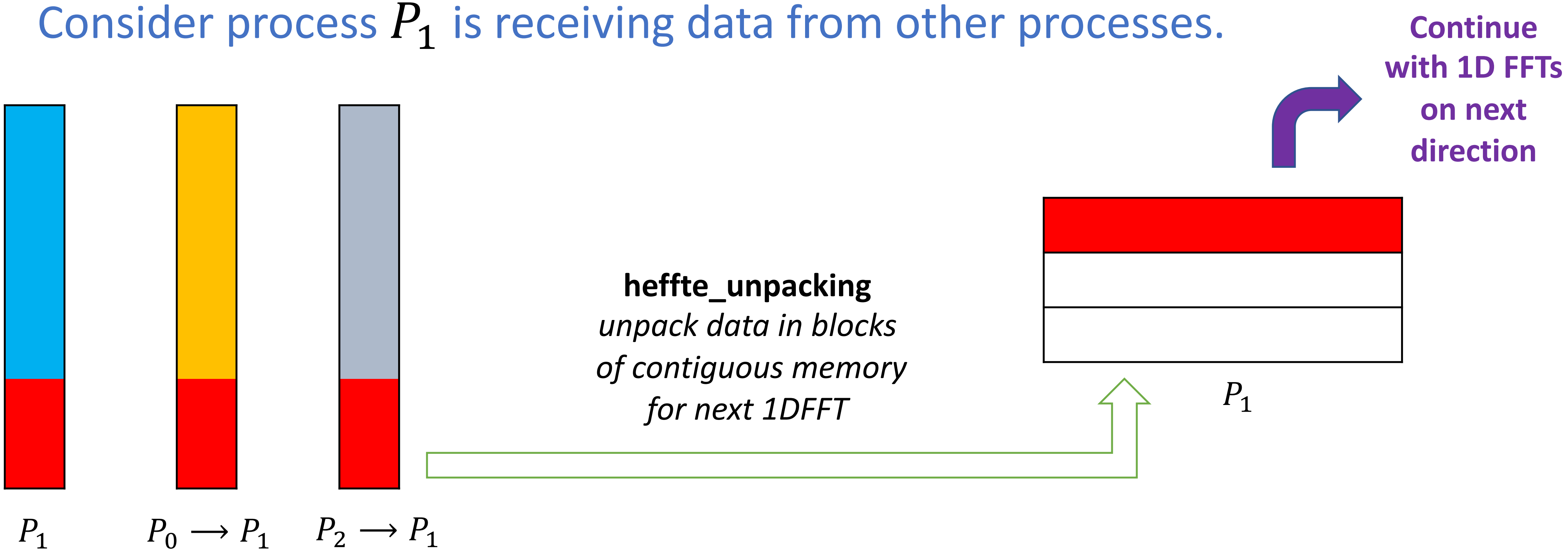
Consider process P_1 is receiving data from other processes.



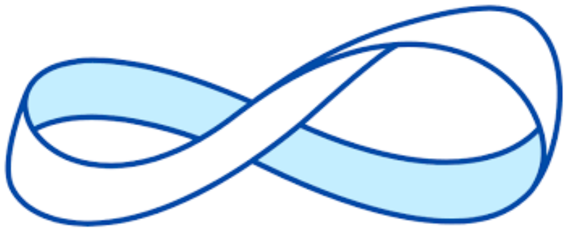
Function: heffte_reshape

3. Data processing, **unpacking** :

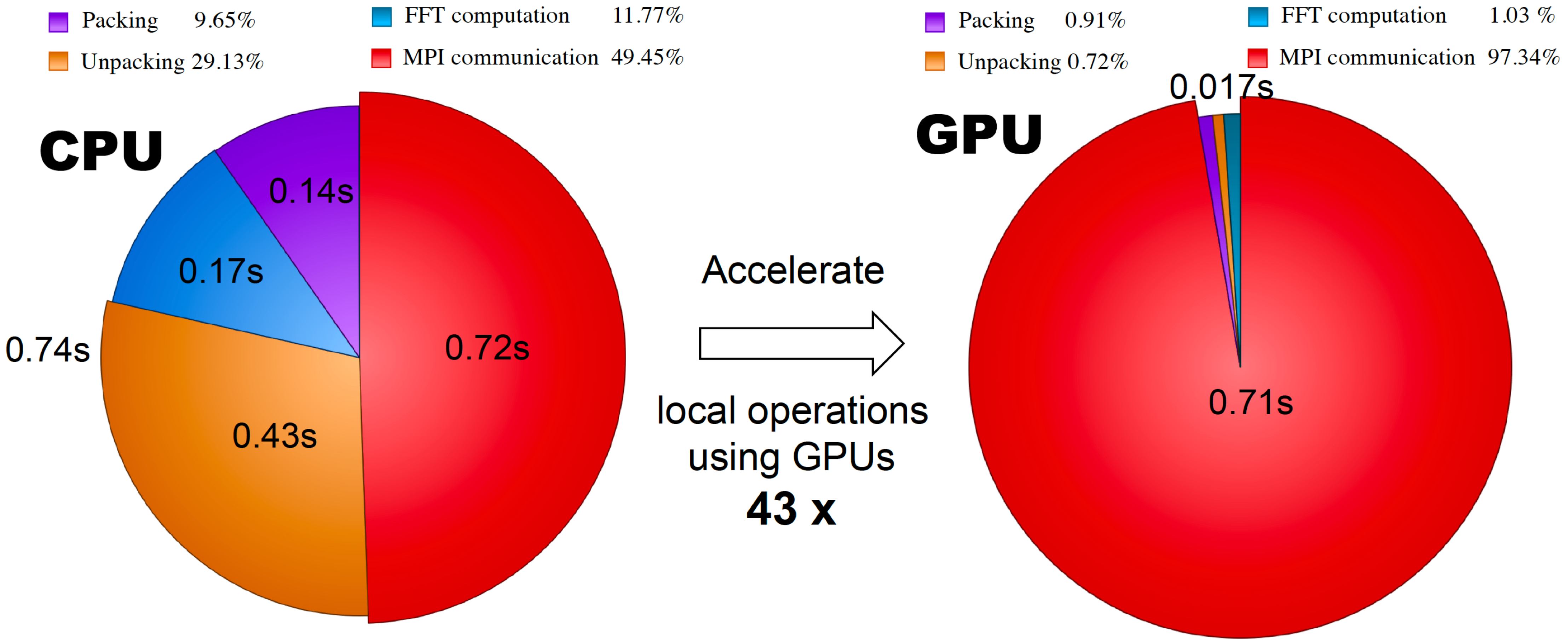
Consider process P_1 is receiving data from other processes.



3. NUMERICAL EXPERIMENTS



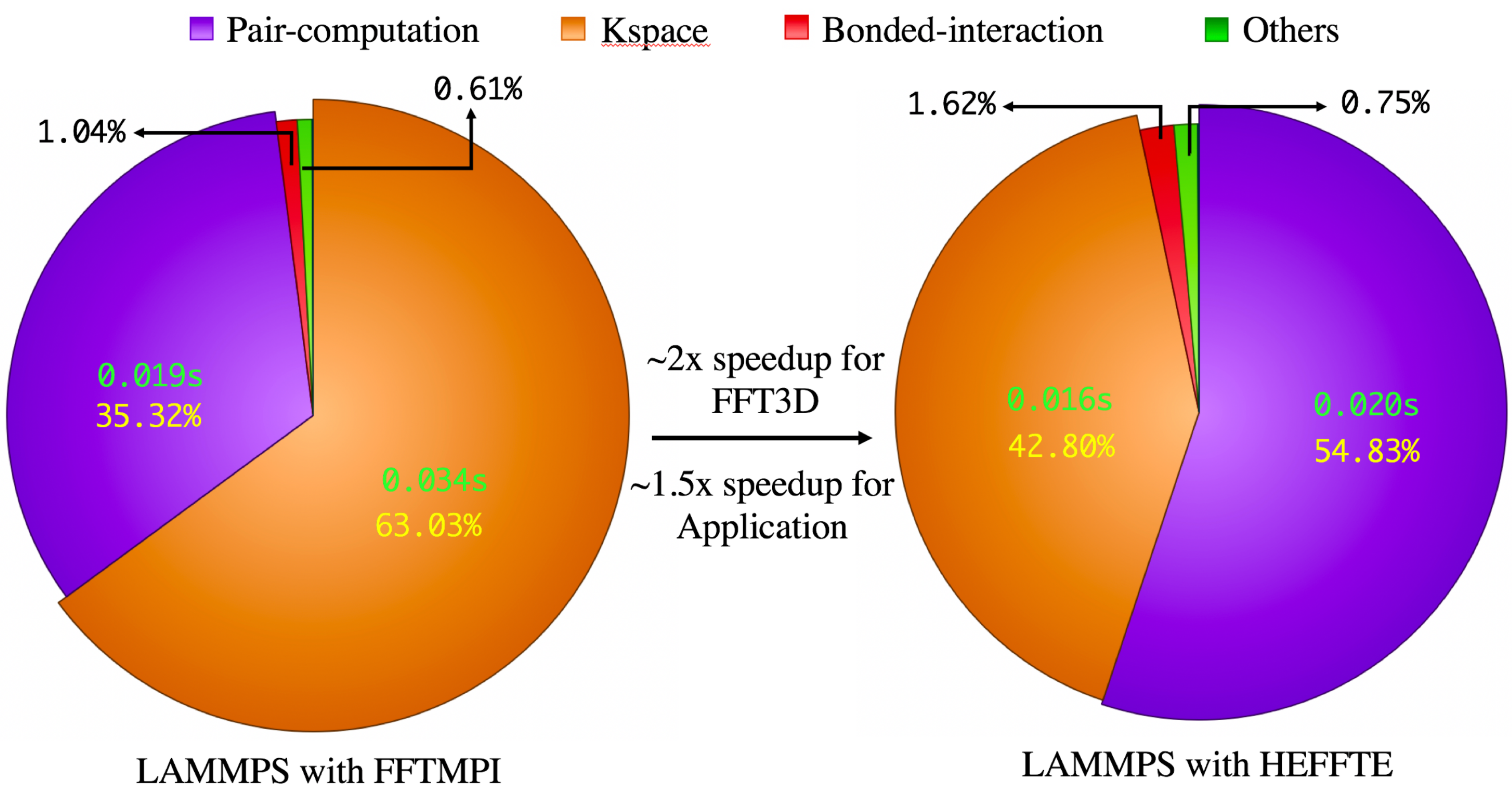
HEFFTE kernels performance



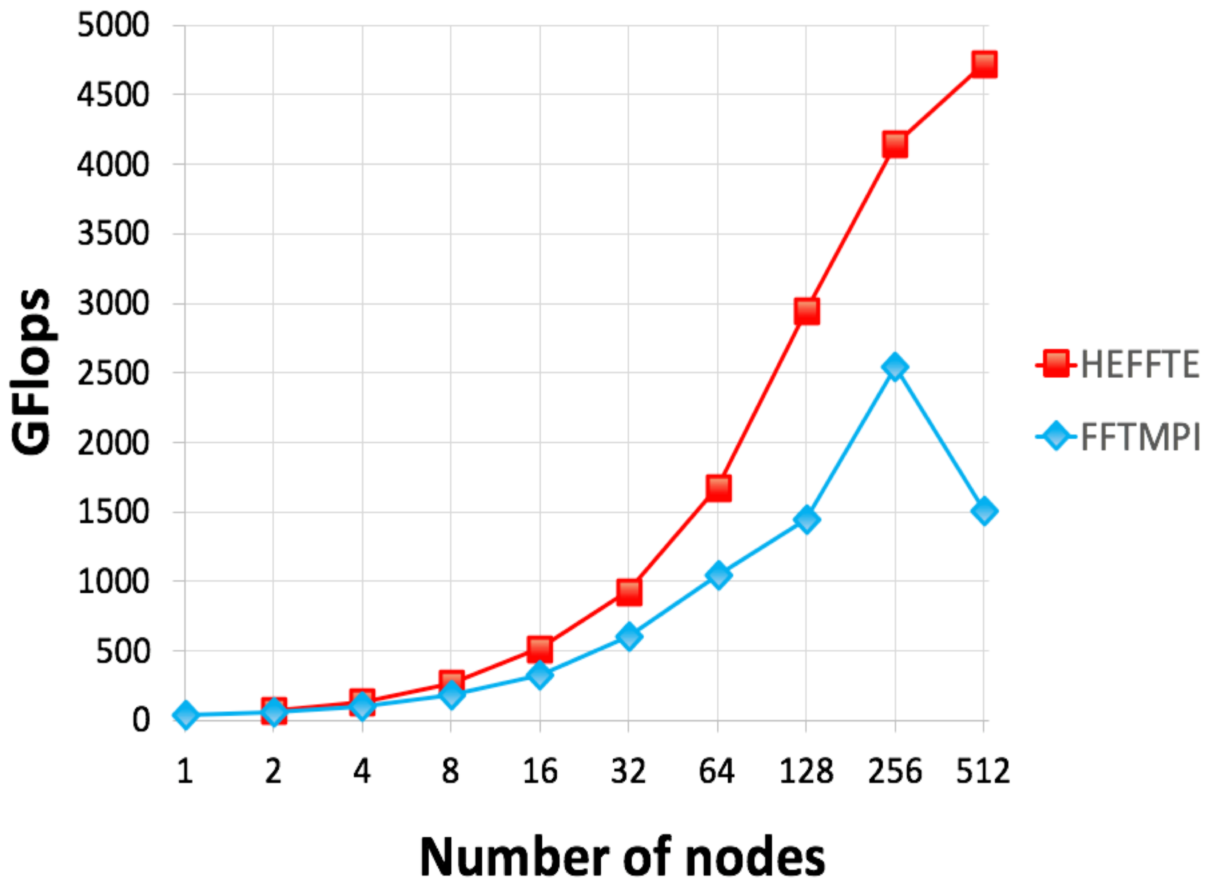
3D FFT of size 1024^3 on 4 CPU nodes – using 128 MPI processes, i.e., 32 MPIs per node, 16 MPIs per socket (Left) vs. 4 GPU nodes – using 24 MPI processes, i.e., 6 MPIs per node, 3 MPI per socket, 1 GPUs per MPI (Right)

Integration of heFFTe to ECP projects

We have started with the integration to EXAALT.
Below a benchmark of LAMMPS is analyzed.

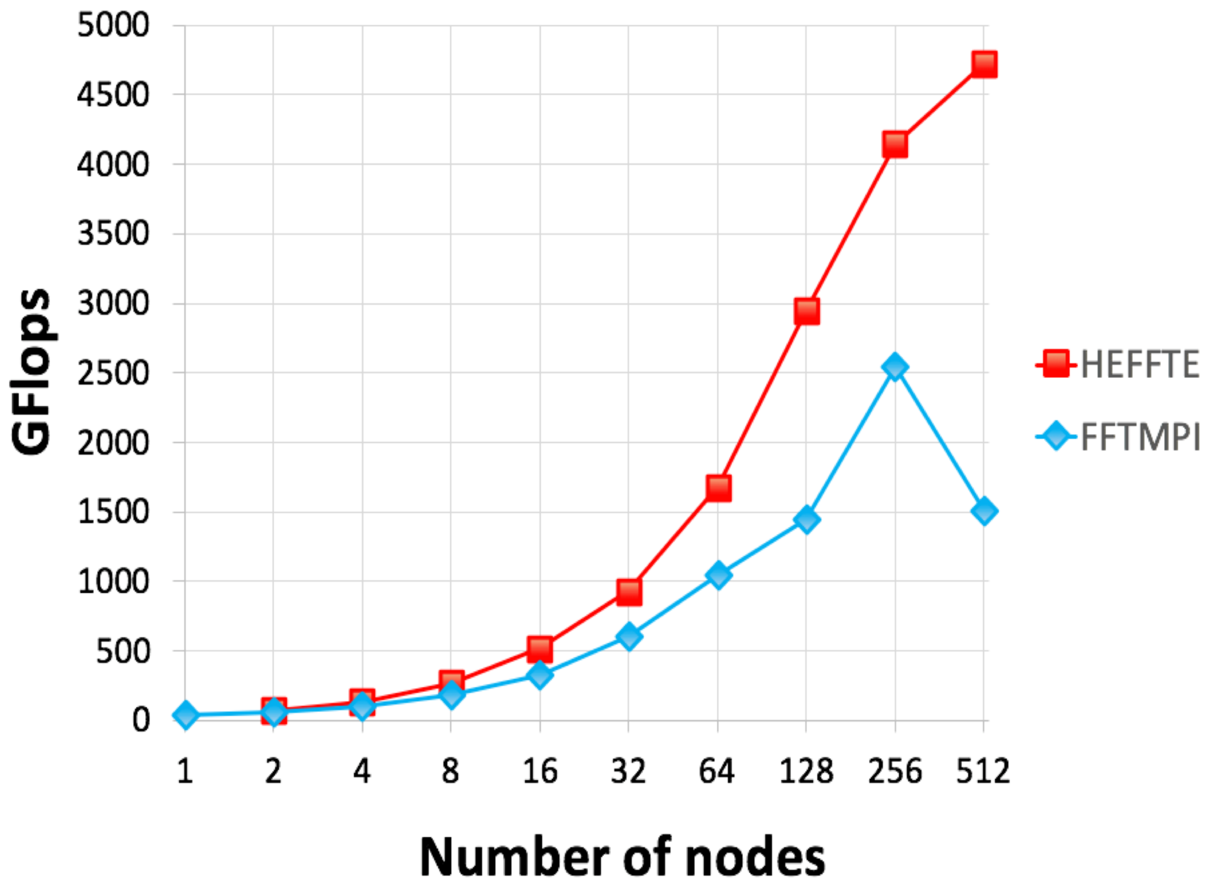


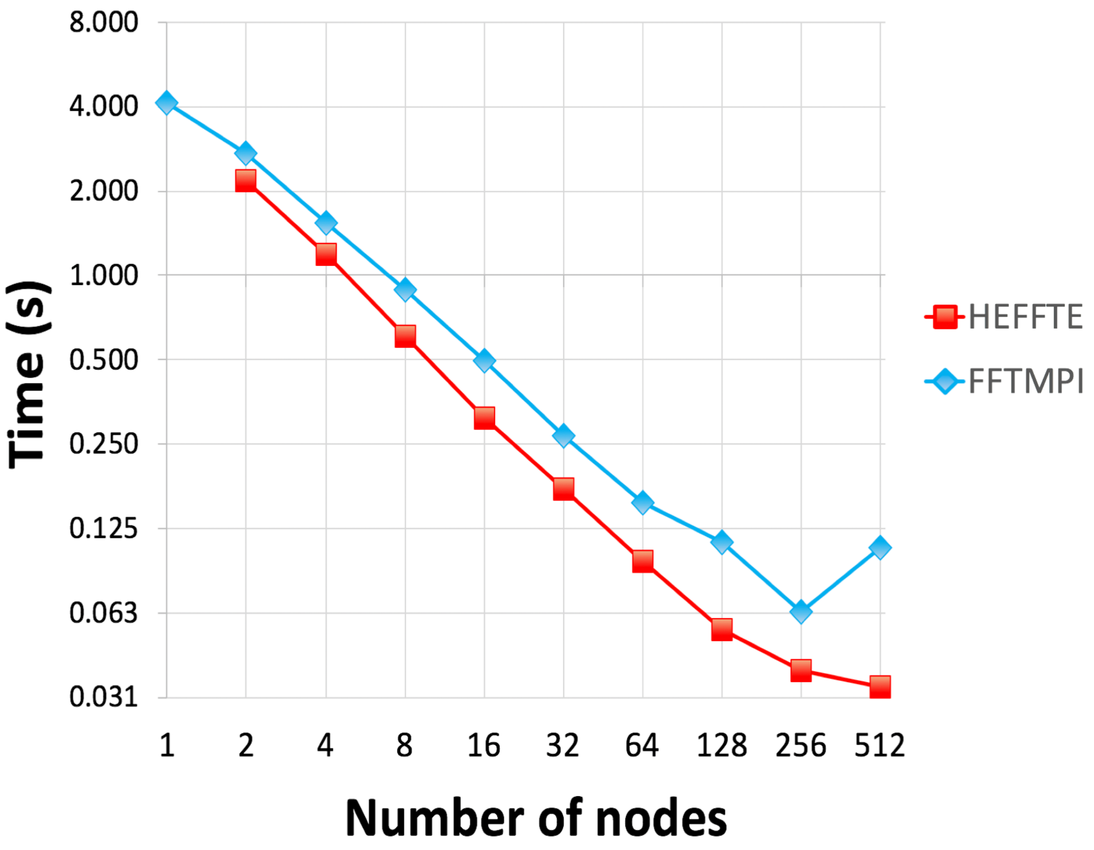
HEFFTE Strong scalability



Performance comparison HEFFTE - FFTMPI, $N = 1024^3$
40 cores/ node – 6 GPUs/ node

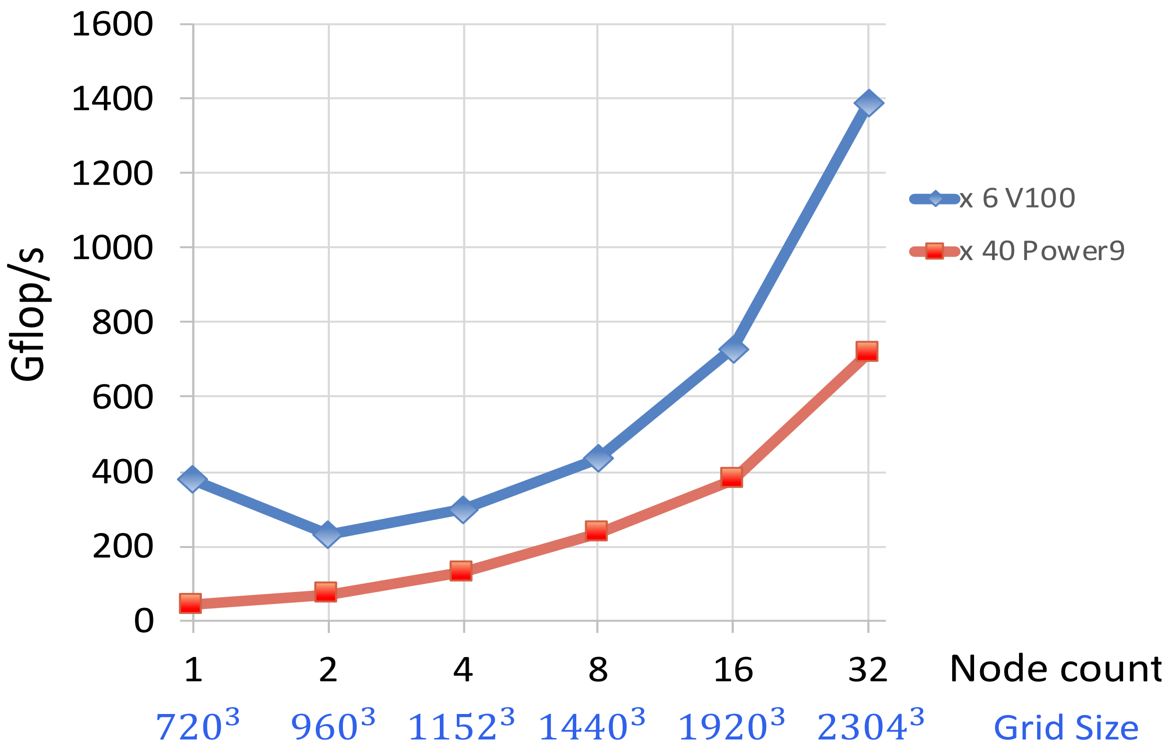
HEFFTE Strong scalability



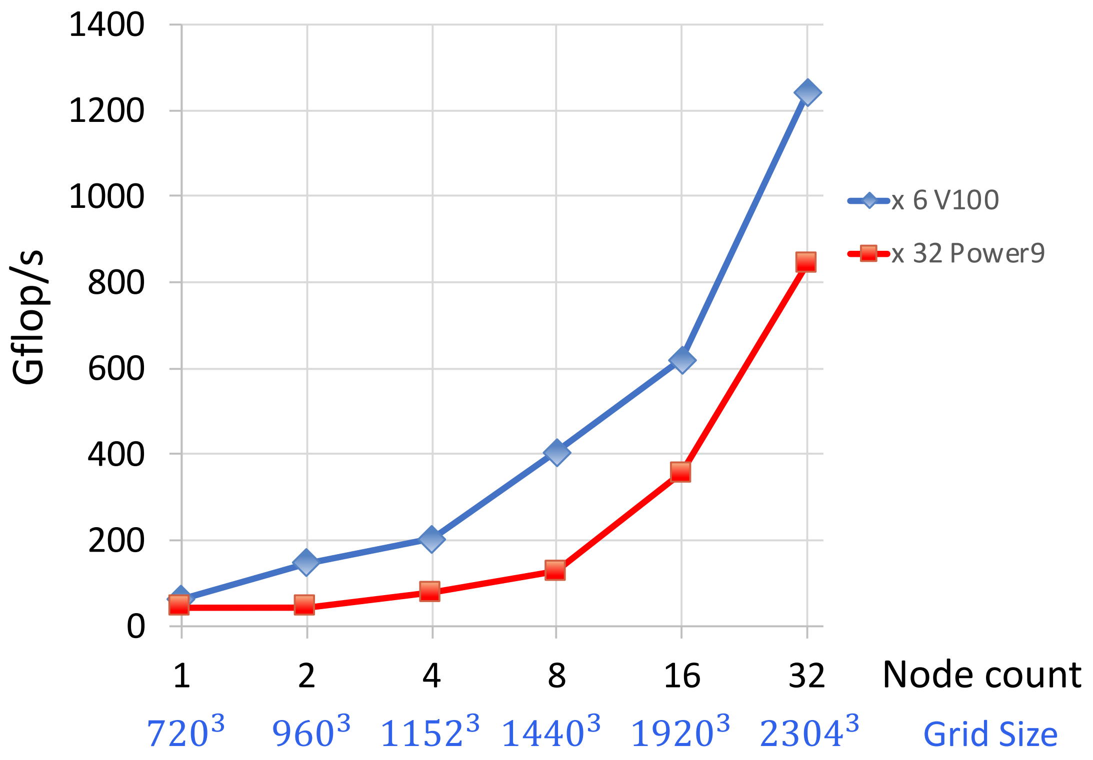


Performance comparison HEFFTE - FFTMPI, $N = 1024^3$
40 cores/ node – 6 GPUs/ node

HEFFTE weak scalability



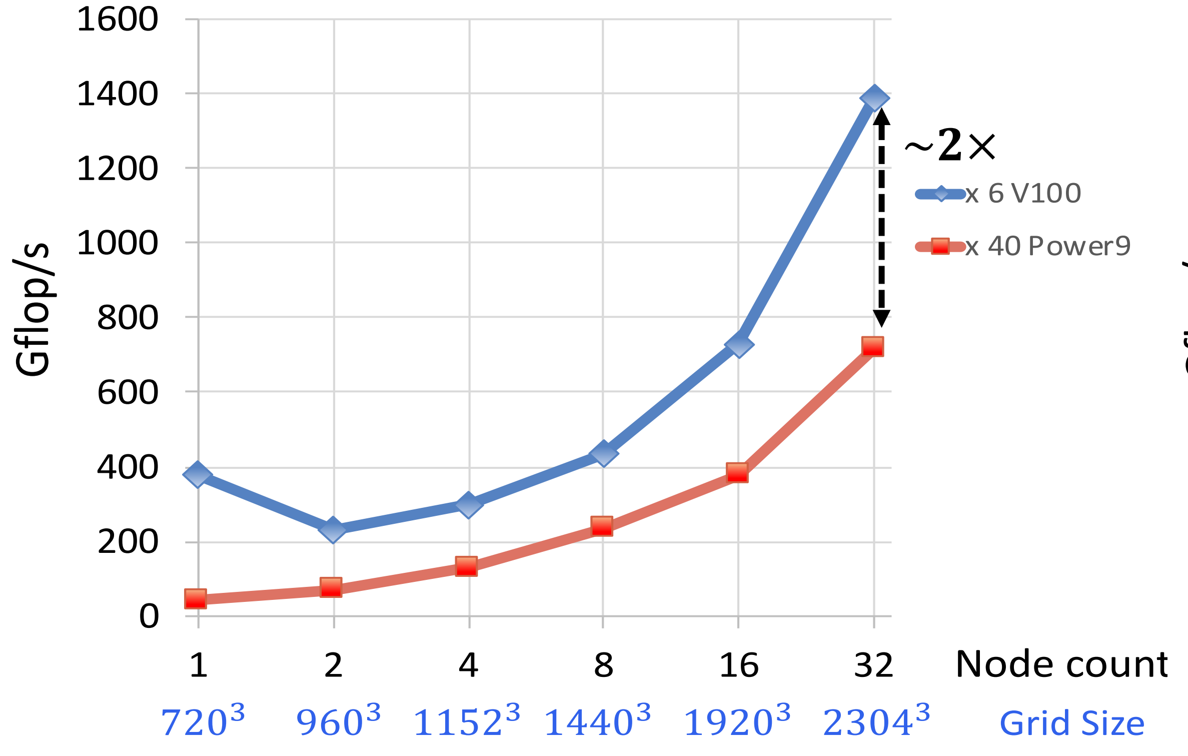
Performance comparison HEFFTE - FFTMPI



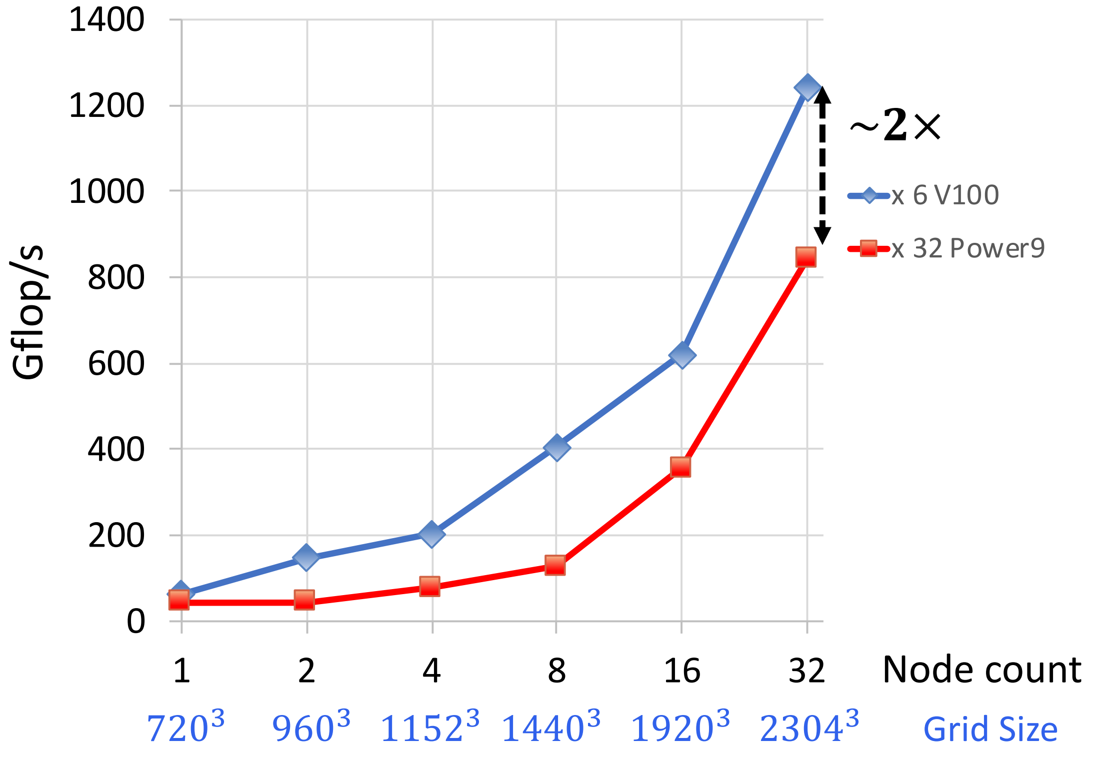
Performance comparison HEFFTE - SWFFT

HEFFTE weak scalability

HEFFTE gets over 40 Gflops/node

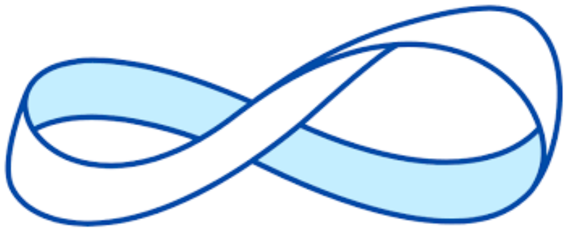


Performance comparison HEFFTE - FFTMPI



Performance comparison HEFFTE - SWFFT

4. COMMUNICATION MODEL



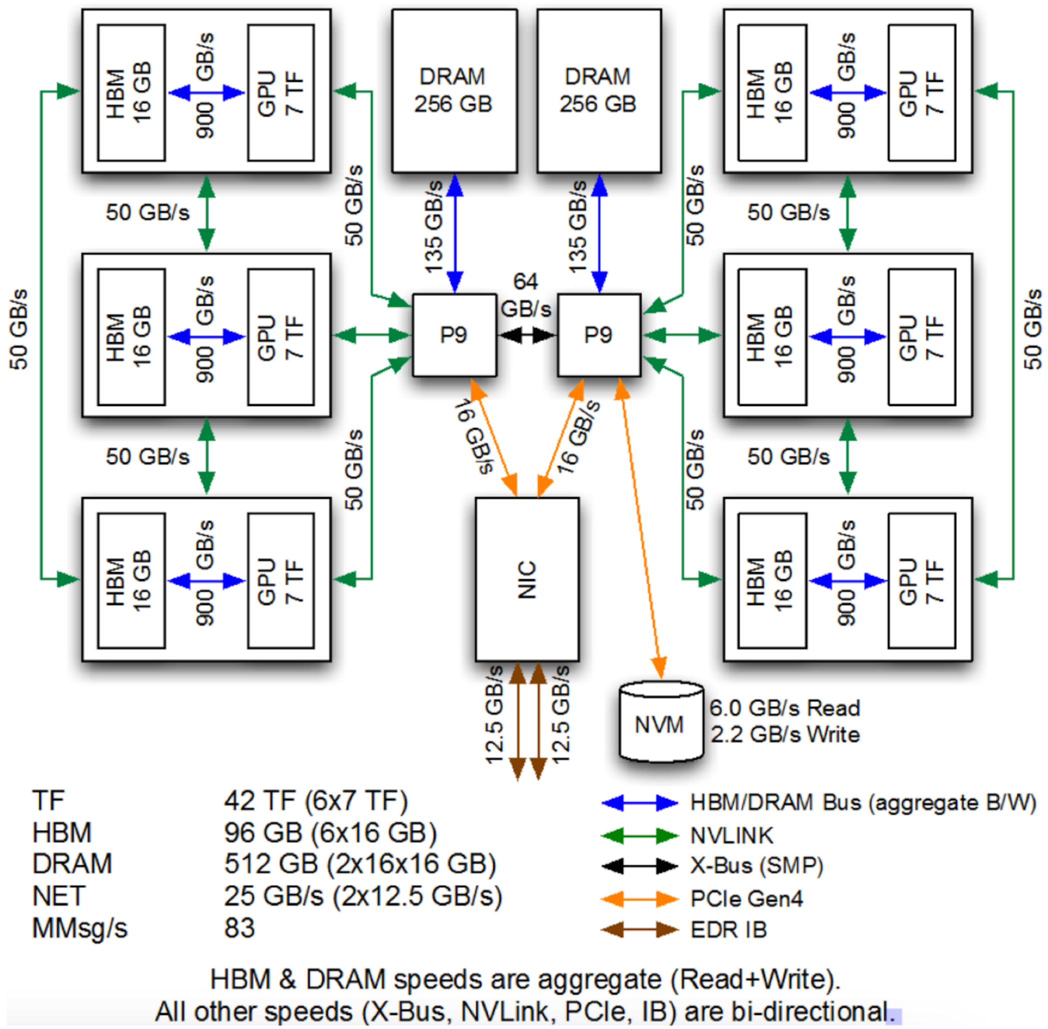
Multi-process (Multi-GPU) comm.

- We rely on CUDA-aware MPI implementations
- Intranode: P2P GPU communication
- Internode: MPI CUDA aware.

Multi-process (Multi-GPU) comm.

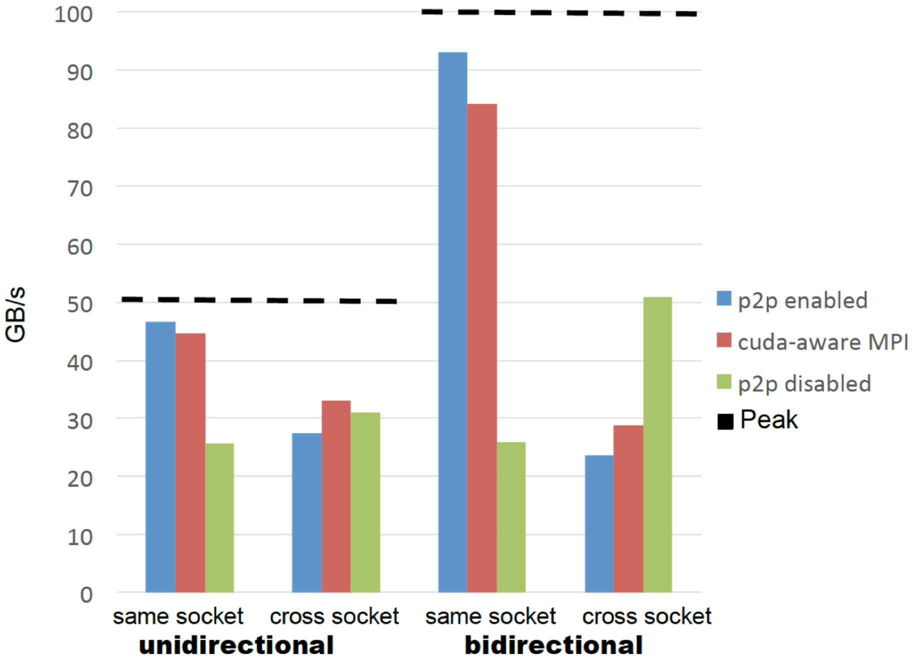
- We rely on CUDA-aware MPI implementations
- Intranode: P2P GPU communication
- Internode: MPI CUDA aware.

A single Summit node



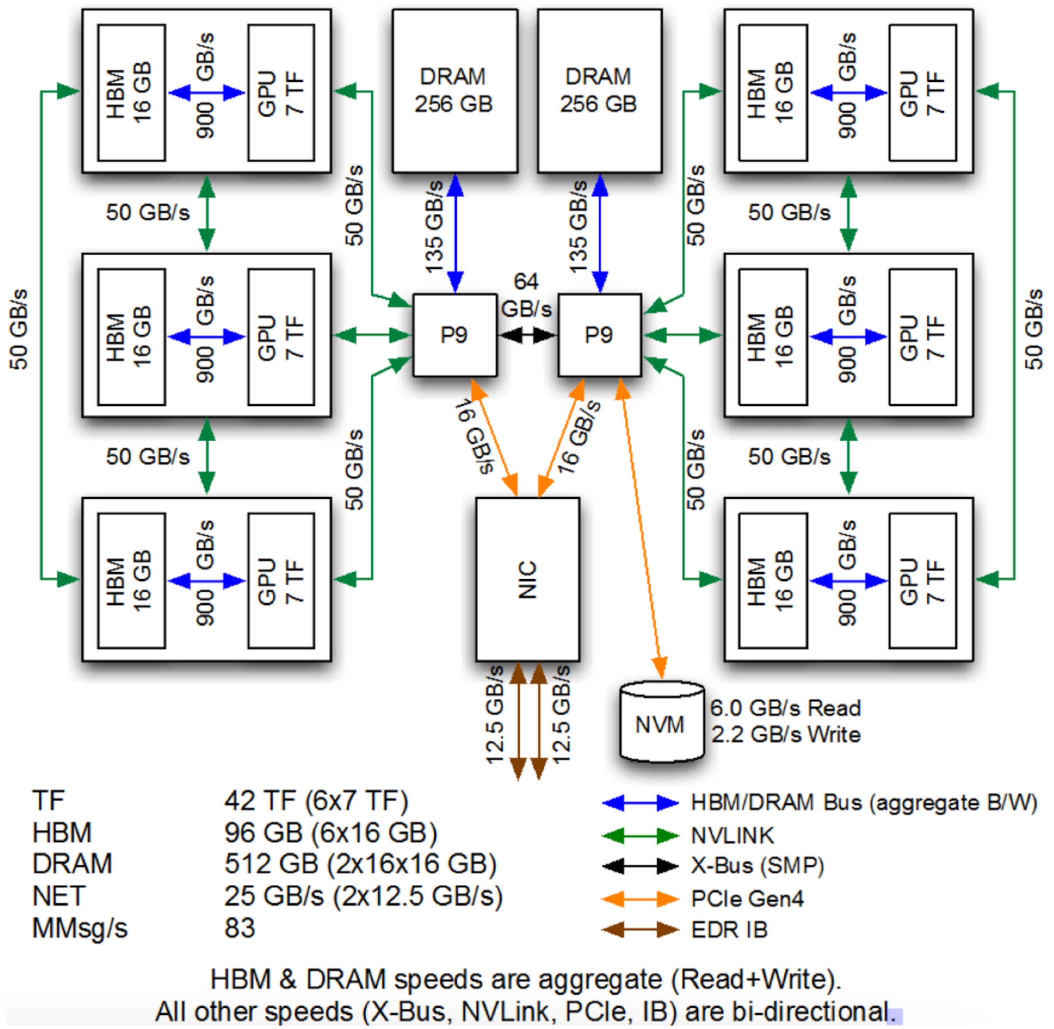
Multi-process (Multi-GPU) comm.

- We rely on CUDA-aware MPI implementations
- Intranode: P2P GPU communication
- Internode: MPI CUDA aware.



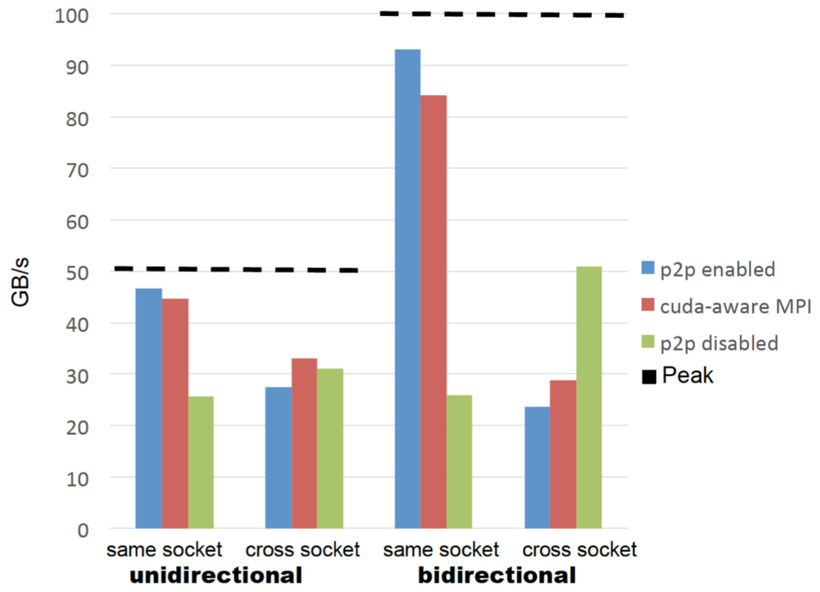
Bandwidth Benchmark for different types of p2p with message size = 40MB

A single Summit node

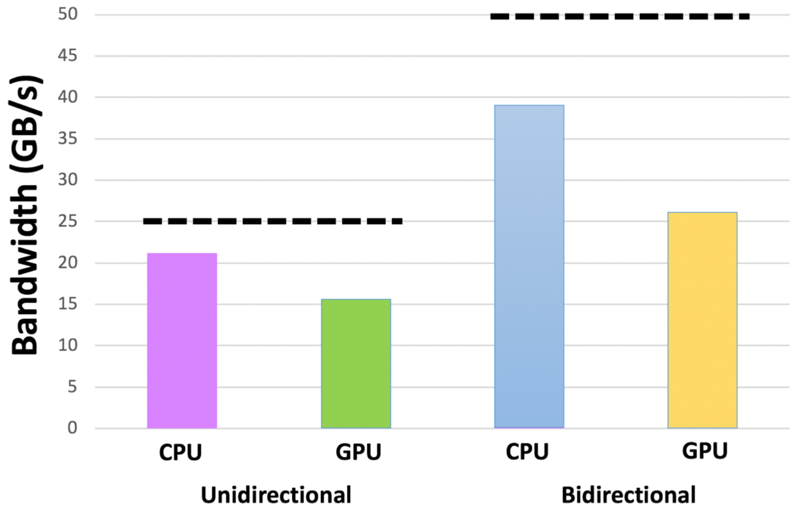


Communication model

1 NODE



2 NODES



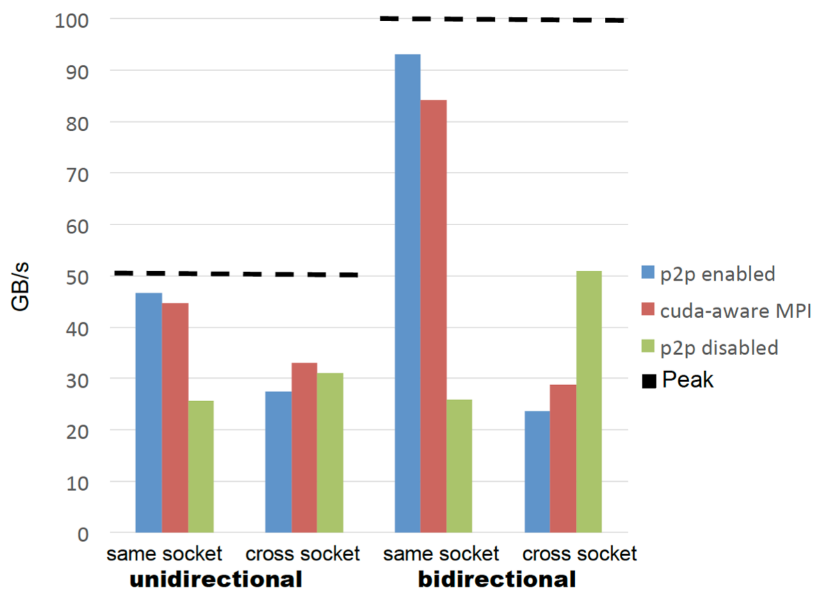
$$Perf\left(\frac{Gflops}{s}/node\right) \leq \min(C, 0.156 B \log_2 N)$$

Where:

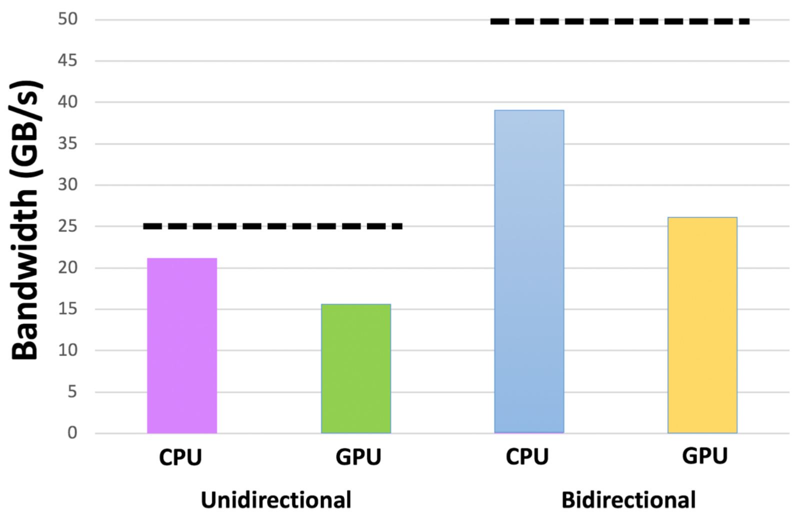
- B: internode bandwidth
- C: performance (Gflops/s) we can extract from a node for batch 1D FFTs
- N: FFT size

Communication model

1 NODE



2 NODES



$$Perf\left(\frac{Gflops}{s}/node\right) \leq \min(C, 0.156B\log_2 N)$$

Where:

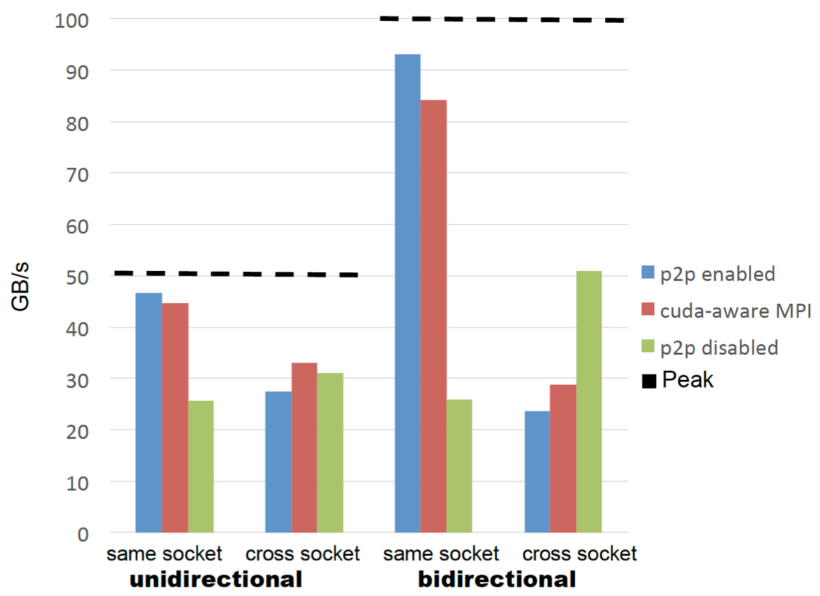
- B: internode bandwidth
- C: performance (Gflops/s) we can extract from a node for batch 1D FFTs
- N: FFT size

Considering:

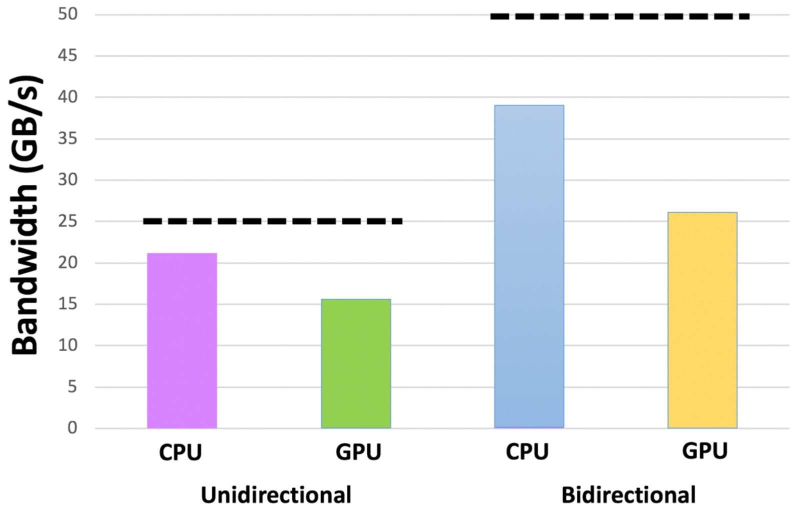
- B = 50, C = 600, N = 10,000

$$Perf\left(\frac{Gflops}{s}/node\right) \leq 103.6$$

1 NODE



2 NODES



Communication model

$$Perf\left(\frac{Gflops}{s}/node\right) \leq \min(C, 0.156B\log_2 N)$$

Where:

- B: internode bandwidth
- C: performance (Gflops/s) we can extract from a node for batch 1D FFTs
- N: FFT size

Considering:

- B = 50, C = 600, N = 1024^3

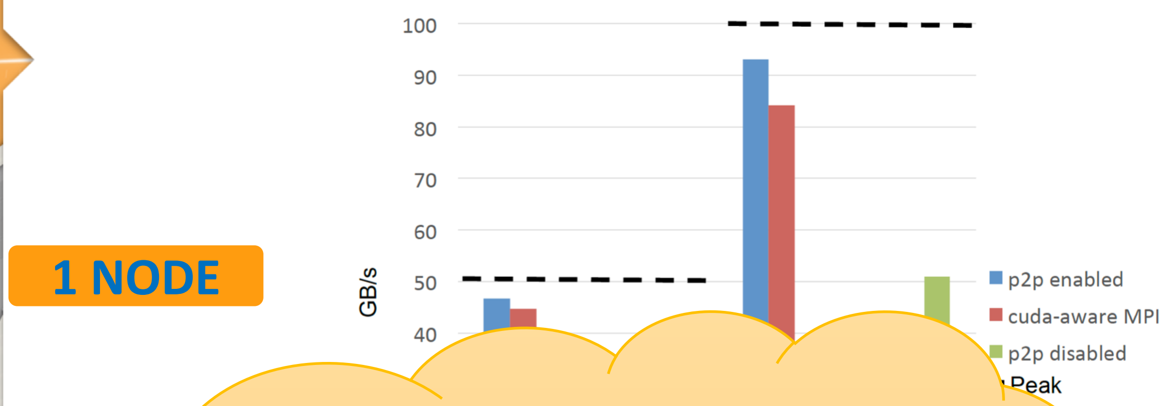
$$Perf\left(\frac{Gflops}{s}/node\right) \leq 234$$

Considering:

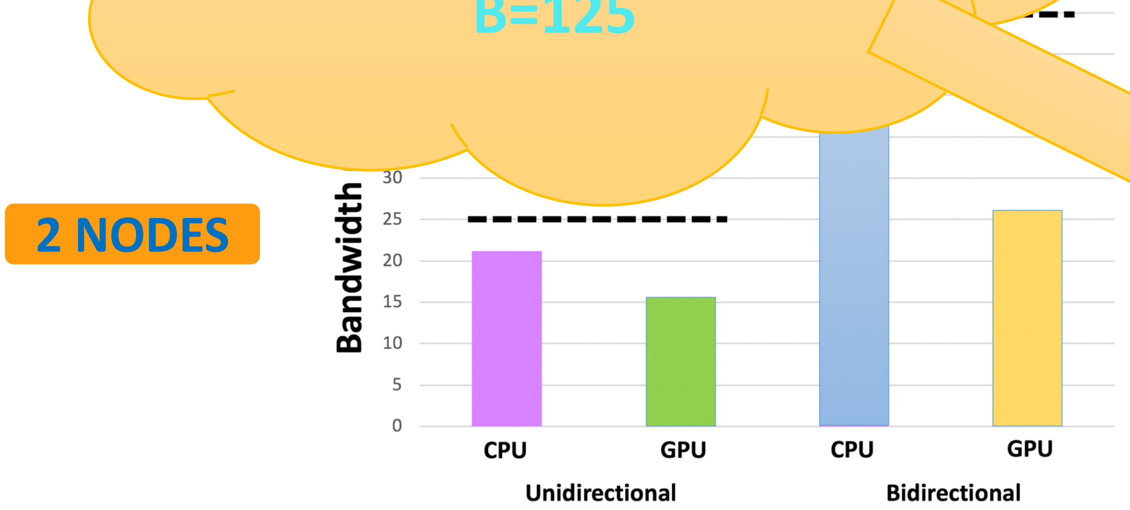
- B = 26, C = 600, N = 1024^3

$$Perf\left(\frac{Gflops}{s}/node\right) \leq 121.7$$

Communication model



For this case, we would not longer be bounded by communication if B=125



$$Perf\left(\frac{Gflops}{s}/node\right) \leq \min(C, 0.156B\log_2 N)$$

Where:

- B: internode bandwidth
- C: performance (Gflops/s) we can extract from a node for batch 1D FFTs
- N: FFT size

Considering:

- B = 50, C = 600, N = 1024^3

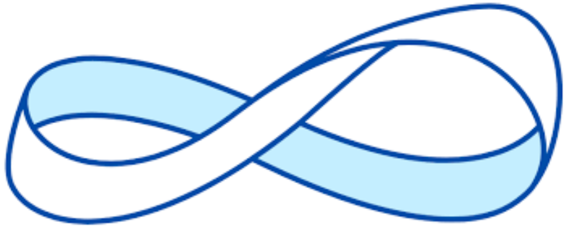
$$Perf\left(\frac{Gflops}{s}/node\right) \leq 234$$

Considering:

- B = 26, C = 600, N = 1024^3

$$Perf\left(\frac{Gflops}{s}/node\right) \leq 121.7$$

5. FURTHER OPTIMIZATION & FUTURE WORK



Magma_Alltoallv

Features several options to perform all-to-all communication, via:
Scatter and Gather, Sendrecv, non-blocking versions, IPC memory handlers

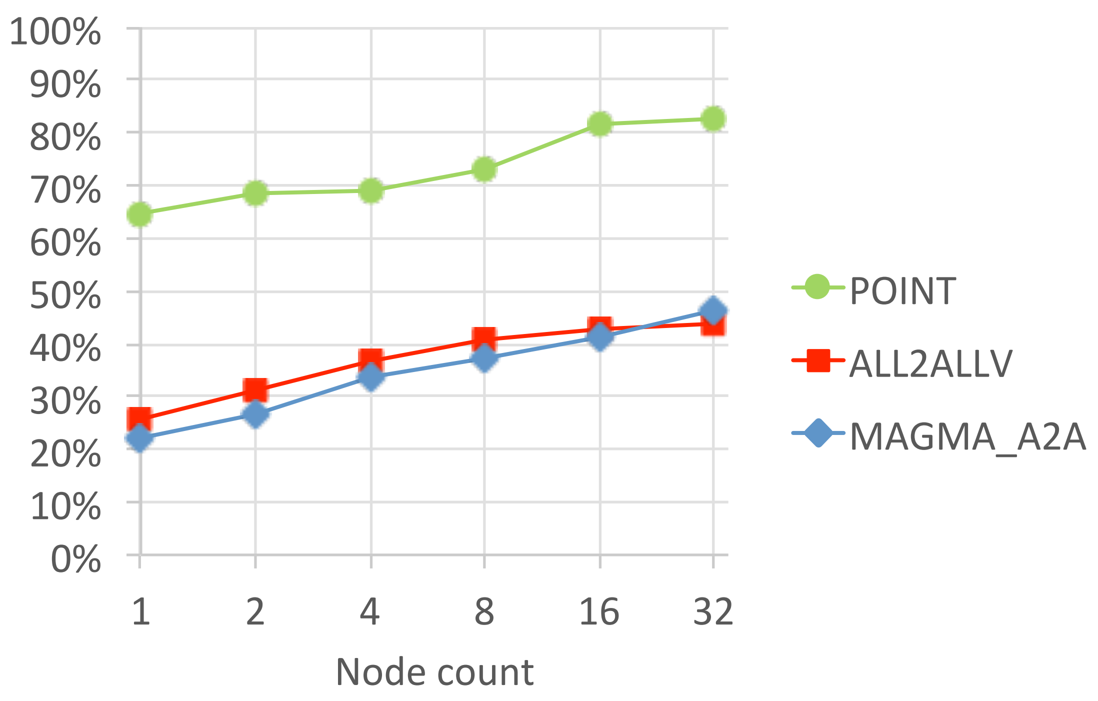
Motivation: Lack of optimized all-to-all multi-gpu communication kernel, e.g. not provided by Nvidia's NCLL

Magma_Alltoallv

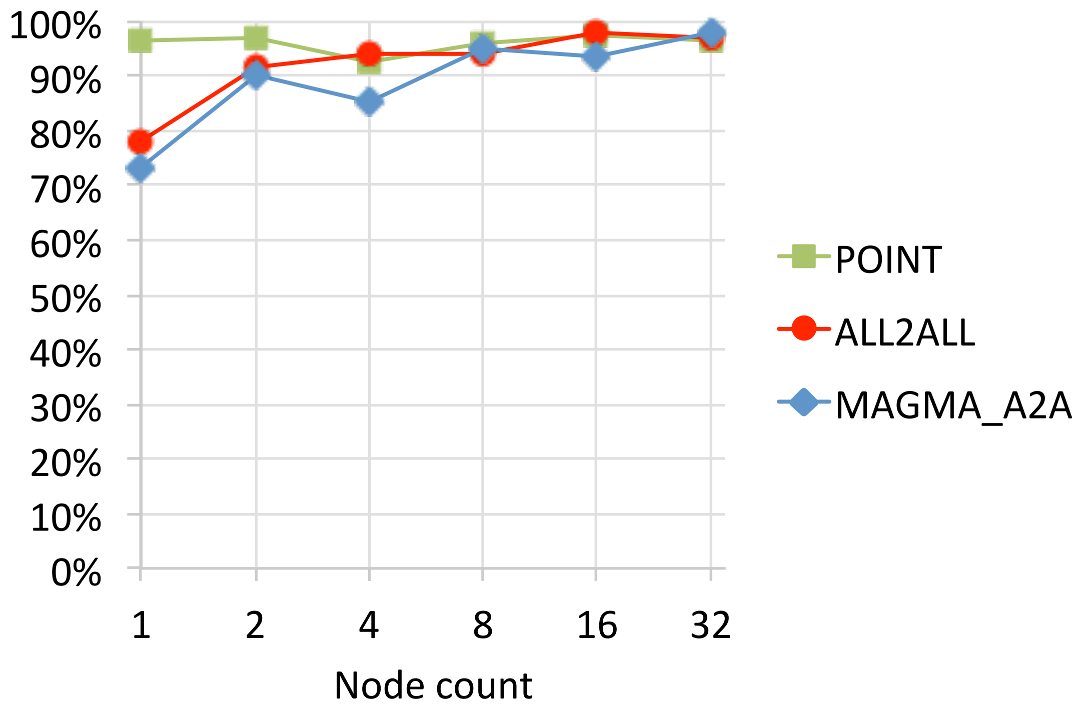
Features several options to perform all-to-all communication, via:
Scatter and Gather, Sendrecv, non-blocking versions, IPC memory handlers

Motivation: Lack of optimized all-to-all multi-gpu communication kernel, e.g. not provided by Nvidia's NCLL

Percent of time spent on MPI communications in FFT on CPUs



Percent of time spent on MPI communications in FFT on GPUs



MPI_Alltoallv – multirail optimization

Collaborators: Xi Luo, George Bosilca.

We analyzed a communication model considering Intranode communication is entirely overlapped with Internode communication.

For MPI_All-to-all, each node transfers. $(n-1) \times p^2 \times M$ bytes of data, where:

Symbol	Description
n	Number of Nodes
p	Number of Processes per Node
M	Message size per Processes
B_{inter}	Inter-node P2P bandwidth

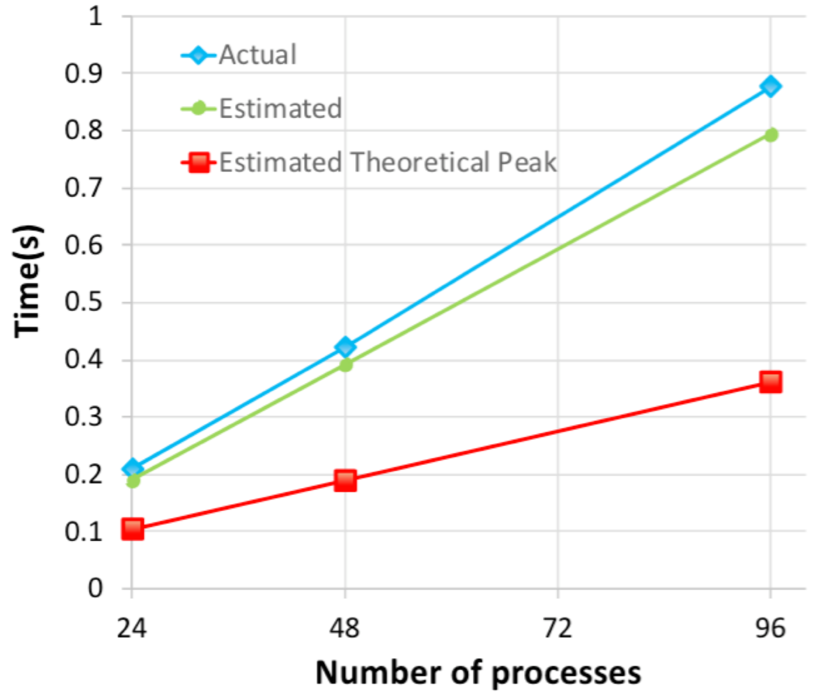
Then the communication time can be modeled as:

$$T_{alltoall} = T_{inter} = (n-1)p^2 M / B_{inter}.$$

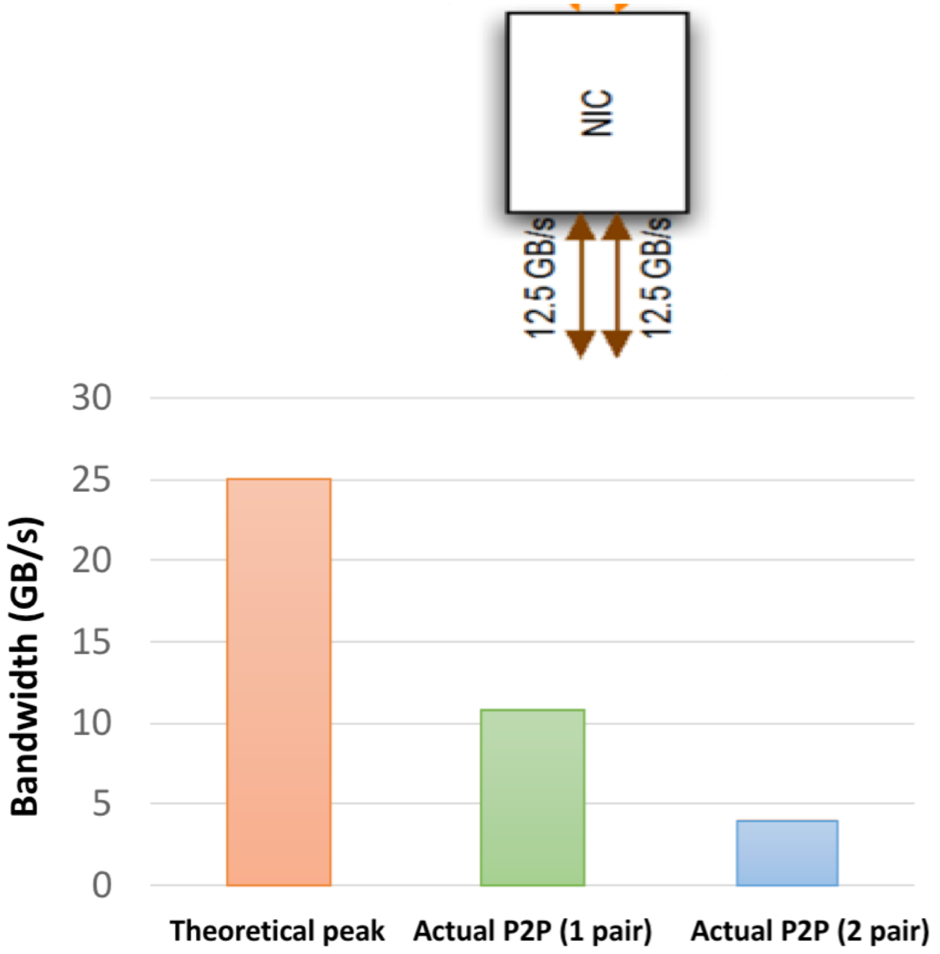
MPI_Alltoallv – multirail optimization

Collaborators: Xi Luo, George Bosilca.

Numerical results on Summit:



MPI_Alltoall performance on Summit (16MB)



MPI P2P performance on Summit (16MB).

ONGOING WORK

1. Bandwidth usage optimization
2. HEFFTE tuning models
3. Integration to other ECP projects, CoPA, HACC
4. Add real-to-complex version
5. Add multi-dimensional FFT
6. Provide fast operators, e.g. Laplace, convolution