EXASCALE COMPUTING PROJECT

An Overview of FFT Computation towards Exascale - Accelerating the Communication Cost of parallel 3-D FFT

The 9th Annual MVAPICH User Group (MUG) Meeting

Alan Ayala[*], Stan Tomov[*],

Miroslav Stoyanov[+], Azzam Haidar[Δ],

Jack Dongarra[*+⊡]

[*] University of Tennessee at Knoxville – Innovative Computing Laboratory

[+] Oak Ridge National Laboratory

[Δ] NVIDIA Corporation

[⊡] University of Manchester

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

CONTENT

1. **Fast Fourier Transform**
   - Single-device computation
   - Parallel algorithm and libraries capabilities
   - heFFTe library
2. **FFT on large scale systems**
   - Communication bottleneck
   - Scalability
   - MPI challenges
     - Phase diagrams
     - Effect of MPI distribution
3. **Performance results and conclusions**

1. FFT COMPUTATION

Algorithms and libraries

ICL THE UNIVERSITY OF TENNESSEE KNOXVILLE

1. Fast Fourier Transform Computation

In essence, the FFT of x, an m-dimensional vector of size $N := N_1 \times N_2 \times \cdots \times N_m$ is defined by $y := FFT(x)$, which is obtained as follows,

$$\tilde{y} := \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \cdots \sum_{n_m=0}^{N_m-1} \tilde{x} \cdot e^{-2\pi i \left(\frac{k_1 n_1}{N_1} + \frac{k_2 n_2}{N_2} \cdots + \frac{k_m n_m}{N_m} \right)},$$

where $\tilde{y} = y(k_1, k_2, \ldots, k_m)$, and $\tilde{x} := x(n_1, n_2, \ldots, n_m)$.

ICL THE UNIVERSITY OF TENNESSEE KNOXVILLE

1. Fast Fourier Transform Computation

In essence, the FFT of x, an m-dimensional vector of size $N := N_1 \times N_2 \times \cdots \times N_m$ is defined by $y := FFT(x)$, which is obtained as follows,

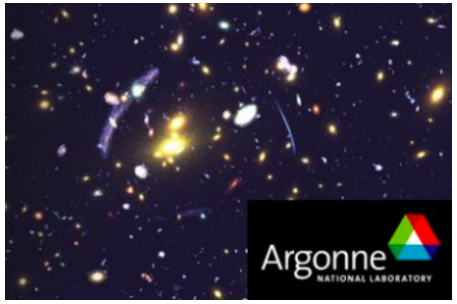
$$\tilde{y} := \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \cdots \sum_{n_m=0}^{N_m-1} \tilde{x} \cdot e^{-2\pi i \left(\frac{k_1 n_1}{N_1} + \frac{k_2 n_2}{N_2} \cdots + \frac{k_m n_m}{N_m} \right)},$$

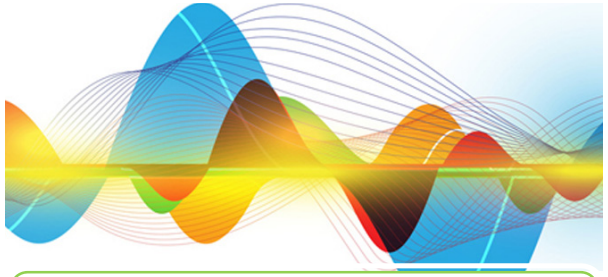
where $\tilde{y} = y(k_1, k_2, \ldots, k_m)$, and $\tilde{x} := x(n_1, n_2, \ldots, n_m)$.

Hence, the FFT could be directly computed by a tensor product; however, this would cost $O(N \sum_{i=1}^{m} N_i)$. The advantage of the FFT is that the cost can be reduced to $O(N \log_2 N)$ operations by exploiting the structure of the tensor.

ICL THE UNIVERSITY OF TENNESSEE KNOXVILLE

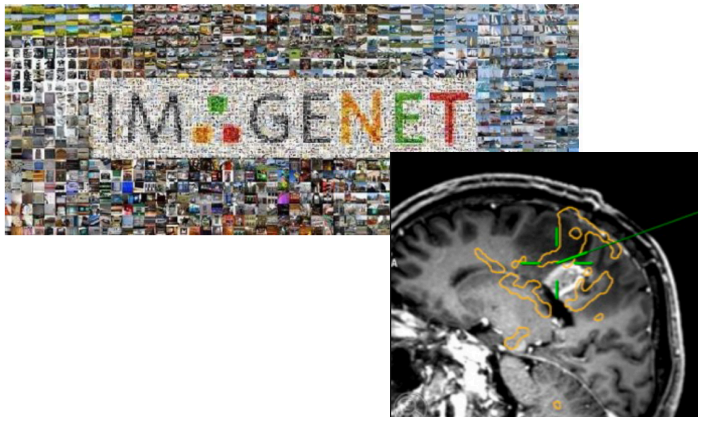
Cosmology
HACC





Signal processing

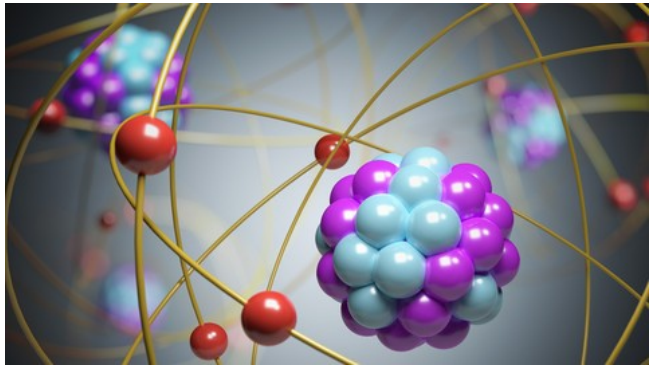
Deep Learning



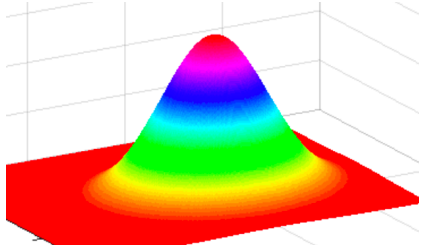
Fast Fourier Transform

Let $\tilde{x} \in \mathbb{R}^N$ and $\tilde{y} = FFT(\tilde{x})$, then:

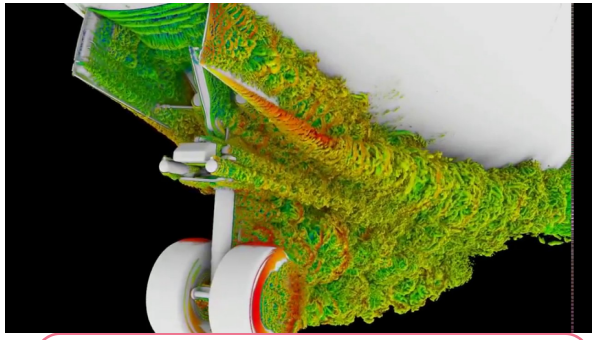
$$\tilde{y} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \cdots \sum_{n_m=0}^{N_m-1} \tilde{x} \cdot e^{-2\pi i \left(\frac{k_1 n_1}{N_1} + \frac{k_2 n_2}{N_2} \cdots + \frac{k_m n_m}{N_m} \right)}$$



Molecular Dynamics
Exaalt



PDE solutions

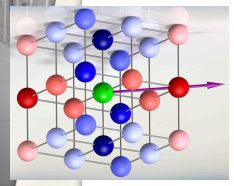


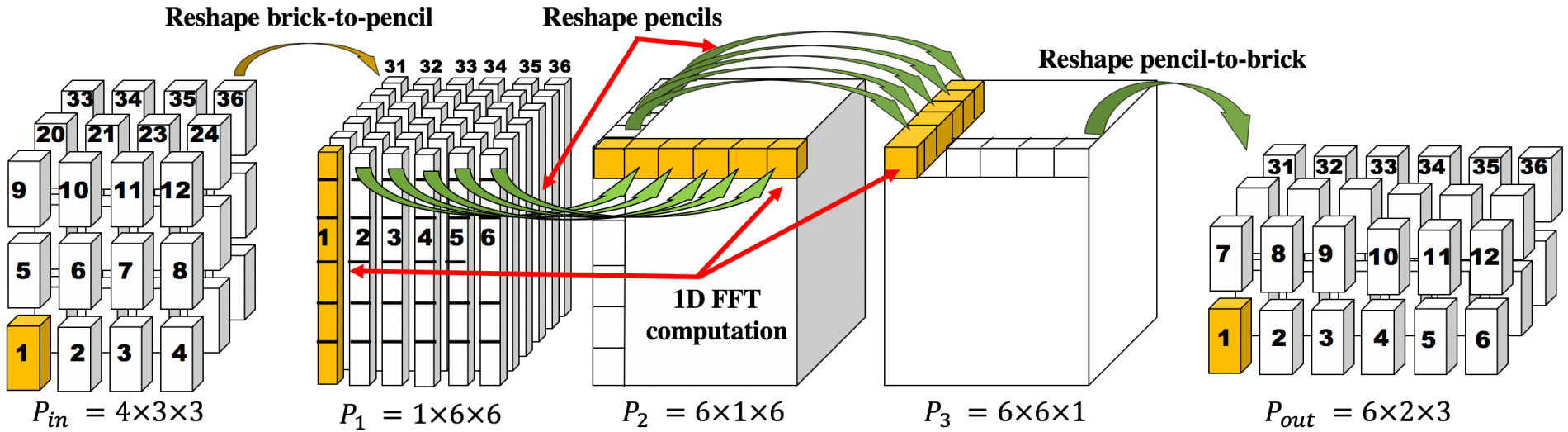
Particle Simulations
CoPa

1.1. State-of-the-art: Single-device FFT libraries

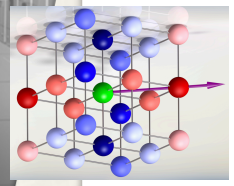
Library	Language	Developer	GPU support	Open Source	2D & 3D support	Stride data support
CUFFT	C	NVIDIA	✓		✓	✓
ESSL	C++	IBM			✓	✓
FFTE	Fortran	Riken		✓	✓	✓
FFTPACK	Fortran	NCAR		✓		
FFTS	C	U. Waikato		✓		
FFTW3	C	MIT		✓	✓	✓
FFTX	C	LBNL	✓	✓	✓	✓
KFR	C++	KFR		✓		✓
KISS	C++	Sandia		✓	✓	✓
OneMKL	C	Intel	✓		✓	✓
ROCM	C++	AMD	✓	✓	✓	✓
VkFFT	C++	D. Tolmachev	✓	✓	✓	✓

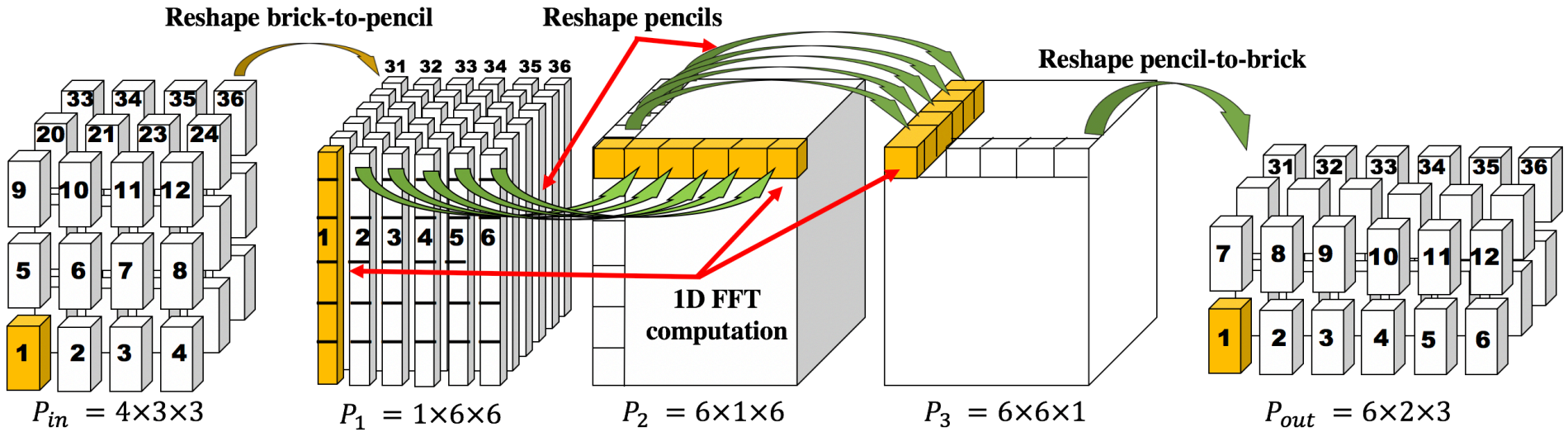
1.2. Distributed FFT computation (3-D case)





1.2. Distributed FFT computation (3-D case)





3-D FFT parallel algorithm

1: From the computation resources, create grid of processes (heuristic approach)

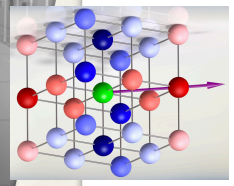
2: Distribute data, define shapes at every transpose phase

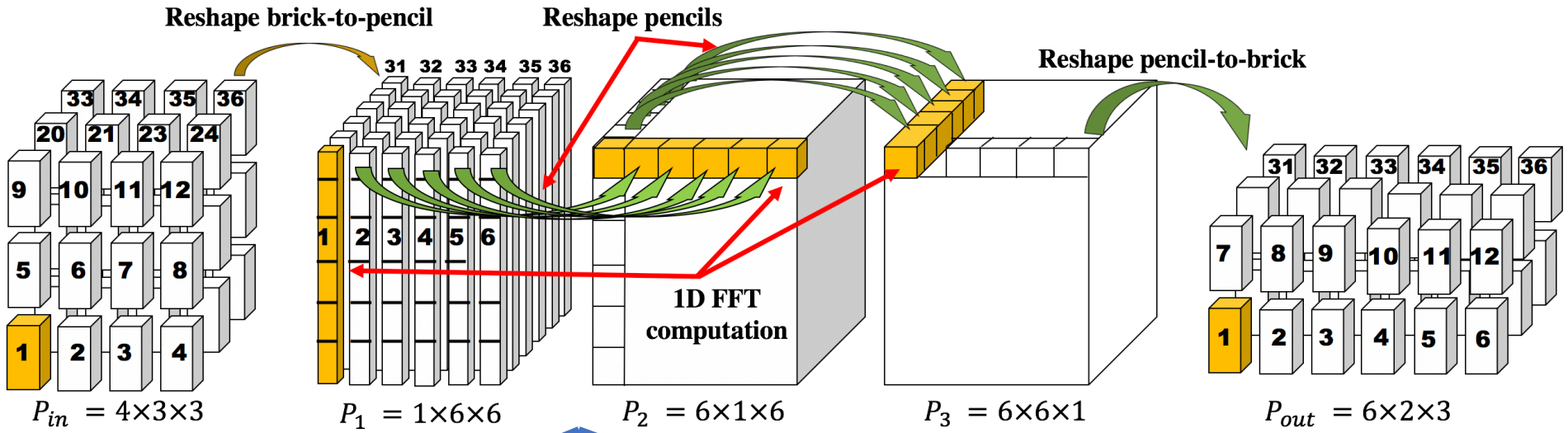
3: Choose algorithms and define or tune parameters

4: For each direction *(x,y,z)*:
- Compute 1D or 2D FFTs
- Locally transpose data
- Pack data (either in contiguous or stride memory)
- Transfer data (Binary or Collective MPI Communication)

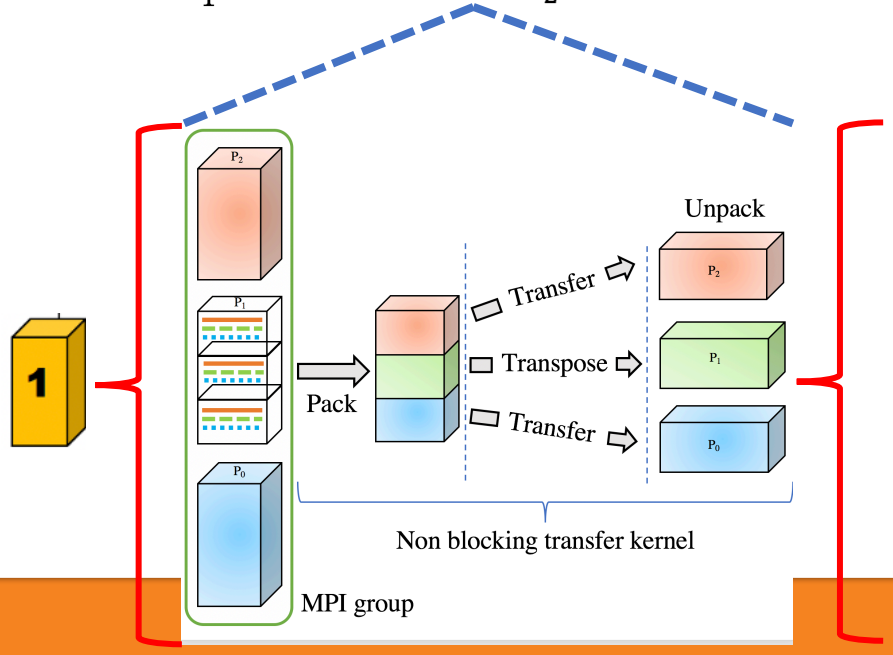
5: At output stage, scale and reorder indices, if required

1.2. Distributed FFT computation (3-D case)





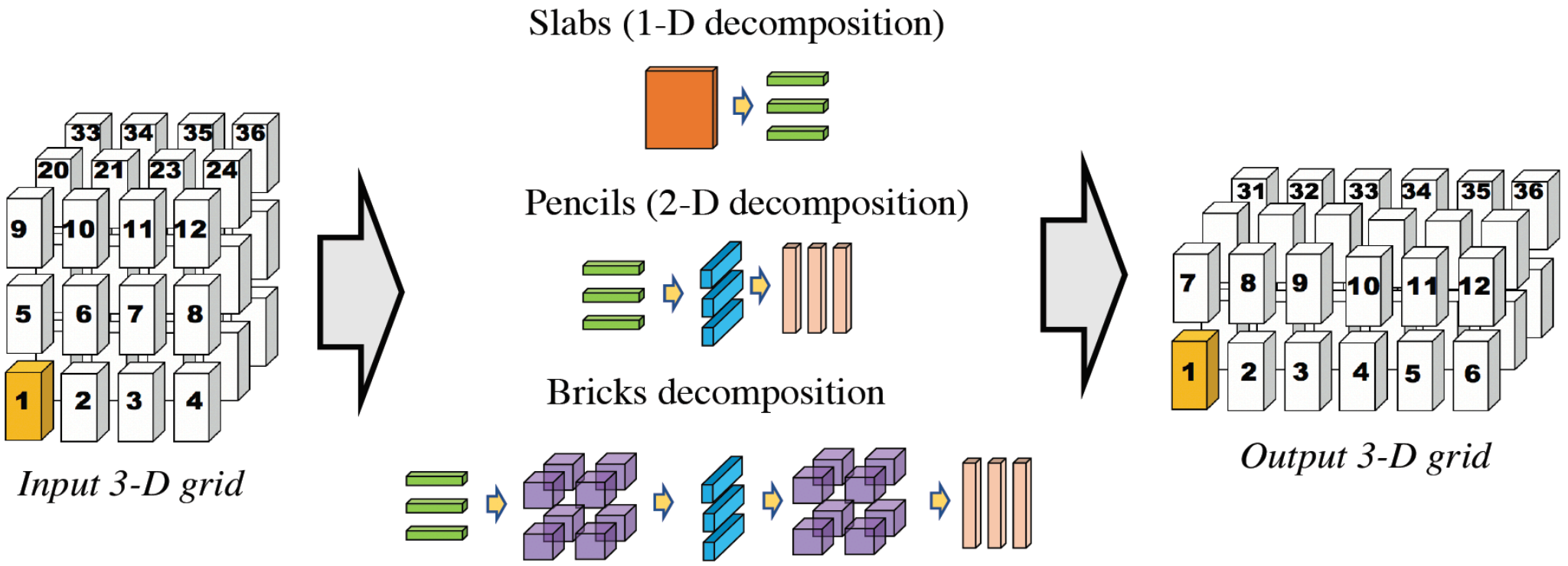
Schematically, Processor 1 is doing the following tasks:



Kernels need:
- 1D FFT (e.g. FFTW3, CUFFT)
- Packers/Unpackers
- A message passing interface (MPI)

1.2. Distributed FFT computation (3-D case)

Different implementations for parallel FFT computation:



1.3 Distributed FFT libraries

Library	Developer	Language	CPU Backend	GPU Backend	Real-to-Complex	Slab Decomp.	Brick Decomp.
2DECOMP&FFT	NAG	Fortran	FFTW3, ESSL	CUFFT	✓	✓	
AccFFT	Georgia Tech	C++	FFTW3	CUFFT	✓		
Cluster FFT	Intel	Fortran	MKL	GenFFT			
CRAFFT	Cray	Fortran	FFTW3	ACML	✓		
FFTE	U. Tsukuba / Riken	Fortran	FFTE	CUFFT	✓		
FFTMPI	Sandia	C++	FFTW3, MKL, KISS				✓
FFTMPI-KK	Sandia	C++		CUFFT			
FFTW3	MIT	C	FFTW3		✓		
heFFTe	ICL - UTK	C++	FFTW3, MKL, Stock	CUFFT, ROCM, OneMKL	✓	✓	✓
nb3DFFT	RWTH Aachen	Fortran	ESSL		✓		
P3DFFT	UC San Diego	C++	FFTW3		✓	✓	
spFFT	ETH	C++	FFTW3	CUFFT, ROCM	✓	✓	
SWFFT	Argonne	C++	FFTW3				✓

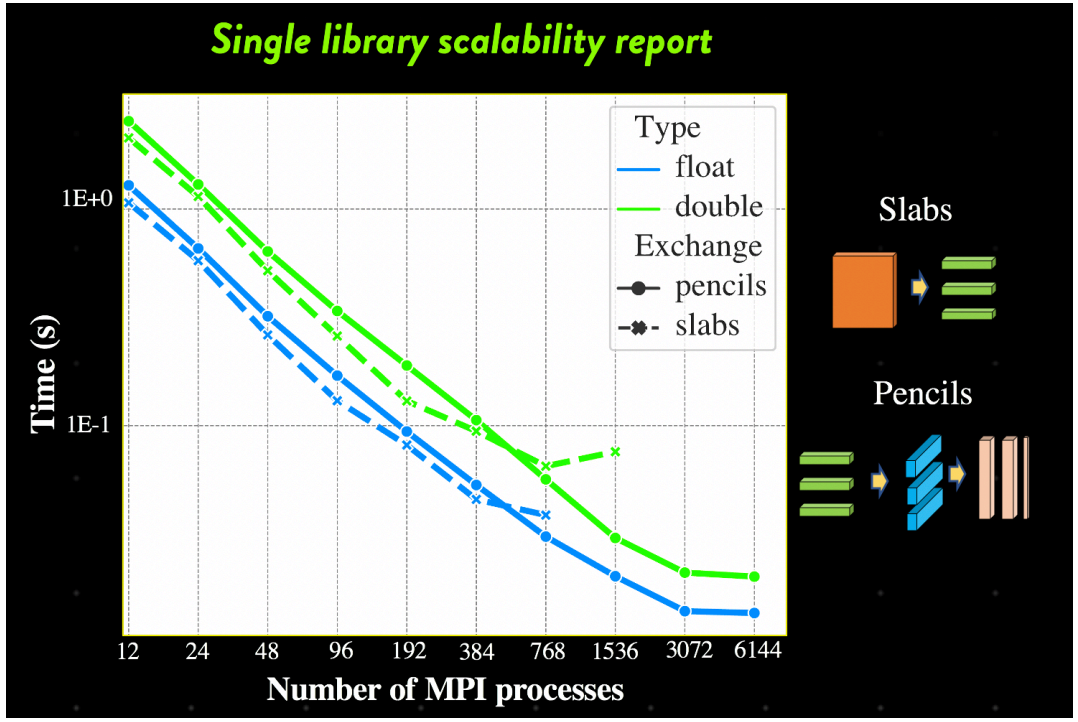
1.4 MPI frameworks on distributed FFT libraries

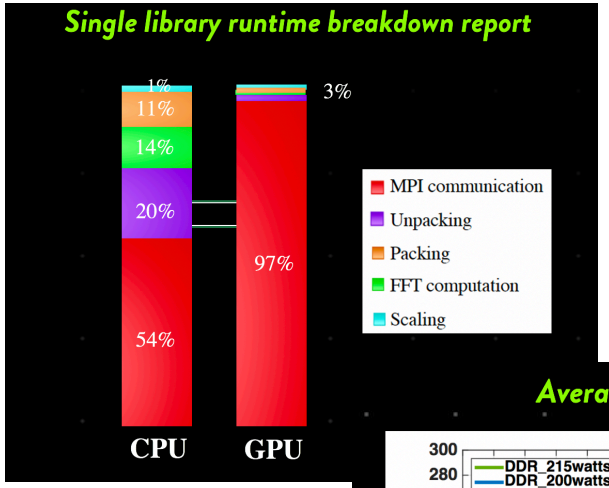
MPI ROUTINES USED FOR TENSOR TRANSPOSITION WITHIN DISTRIBUTED FFT LIBRARIES.

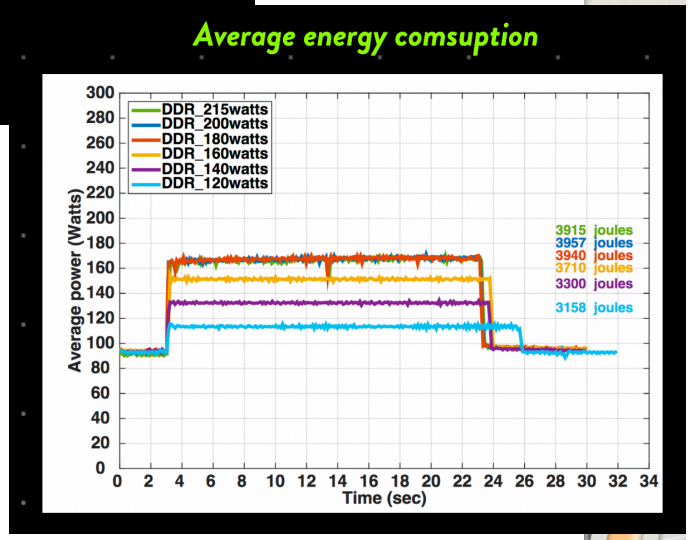
Library Name	Language	Developer	Point-to-Point exchange		Collective exchange		Process Topology
			Blocking	Non-Blocking	Blocking	Non-Blocking	
AccFFT	C++	GA Tech.	MPI_Sendrecv	MPI_Isend MPI_Irecv	MPI_Alltoallv MPI_Bcast	-	MPI_Cart_create
2DECOMP&FFT	Fortran	NAG	MPI_Send	MPI_Irecv	MPI_Alltoall MPI_Alltoallv	-	MPI_Cart_create MPI_Cart_sub
FFTE	Fortran	U. Tsukuba Riken	-	MPI_Isend MPI_Irecv	MPI_Alltoallv MPI_Alltoallw	MPI_Ialltoallv	-
FFTMPI	C++	Sandia	MPI_Send	MPI_Irecv	MPI_Allreduce MPI_Alltoallv	-	MPI_Group MPI_Comm_create
FFTW	C	MIT	-	-	MPI_Alltoallv	-	MPI_Group MPI_Comm_create
heFFTe	C++	UTK	MPI_Send MPI_Recv	MPI_Isend MPI_Irecv	MPI_Allreduce MPI_Alltoallv	heFFTe_Alltoallv	MPI_Group MPI_Comm_create
nb3dFFT	Fortran	RTWH Aachen	MPI_Send MPI_Recv	-	MPI_Allreduce MPI_Alltoallv	-	MPI_Group
P3DFFT	C++	UCSD	-	-	MPI_Alltoallv	-	MPI_Cart_create MPI_Cart_sub
SpFFT	C++	ETH Zürich	MPI_Send	MPI_Irecv	MPI_Allreduce MPI_Alltoallv	-	MPI_Group MPI_Comm_create
SWFFT	C++	Argonne	MPI_Send	MPI_Irecv	MPI_Allreduce MPI_Alltoallv	-	MPI_Cart_create MPI_Cart_sub

FFT Benchmarking Initiative - ICL

At ICL we are working a benchmark harness for evaluating the performance and scalability of state-of-the-art libraries towards Exascale computing.

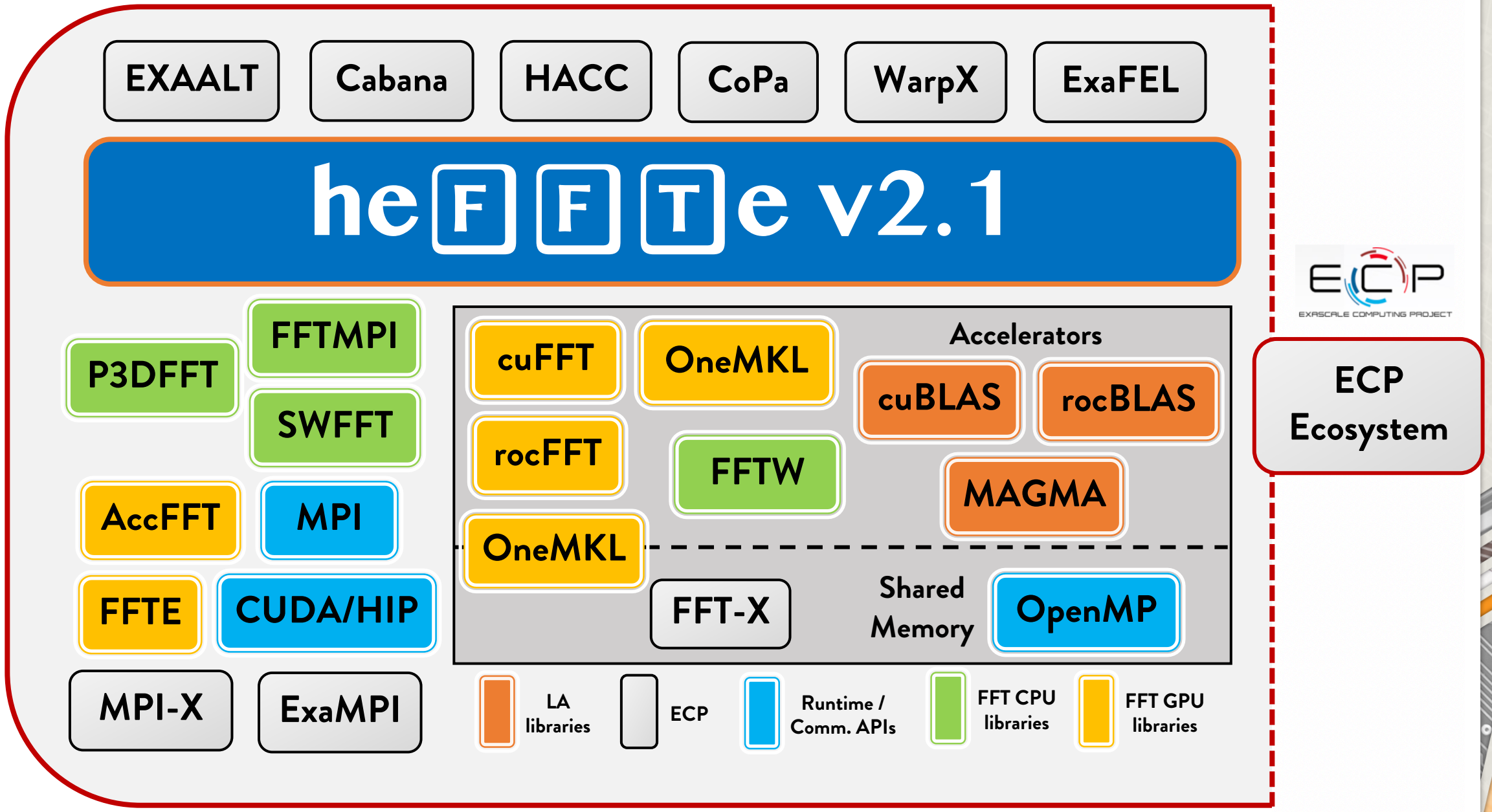






Refer to: *Interim Report on Benchmarking FFT Libraries on High Performance Systems*
Innovative Computing Laboratory Technical Report, July 2021
University of Tennessee

2. heFFTe library:
FFTs for exascale



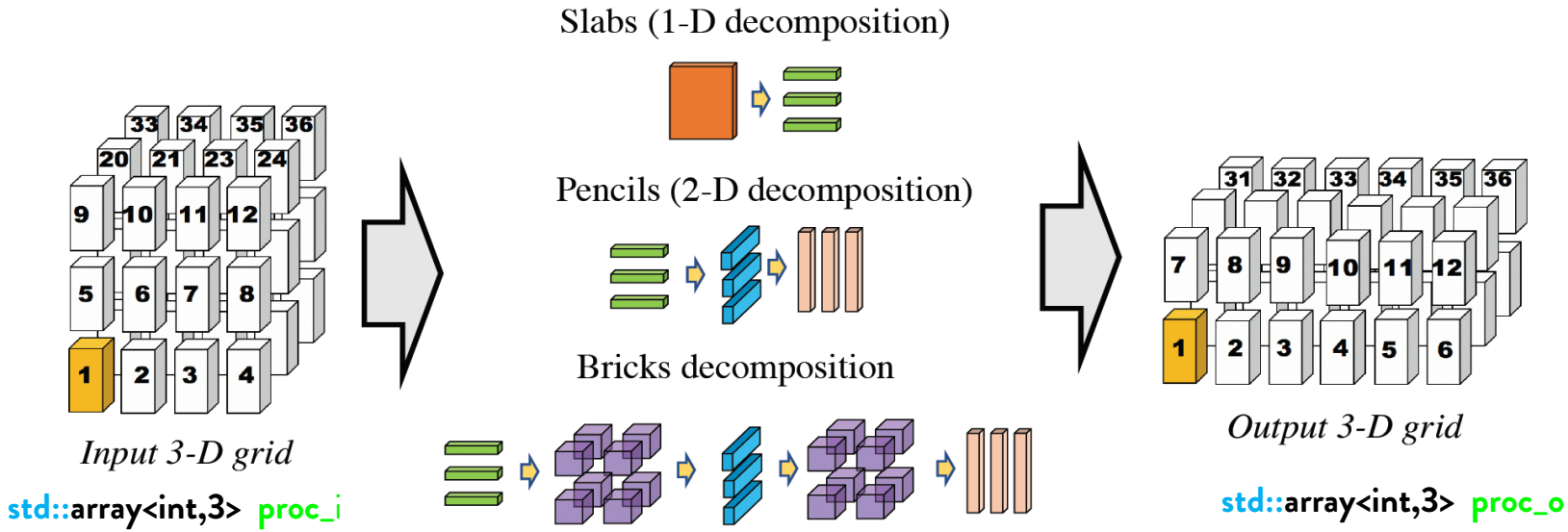
ECP Ecosystem

bitbucket.org/icl/heffte/

icl.utk.edu/fft/

heFFTe API

1. Definition of input/output processors grids (normally provided by users):

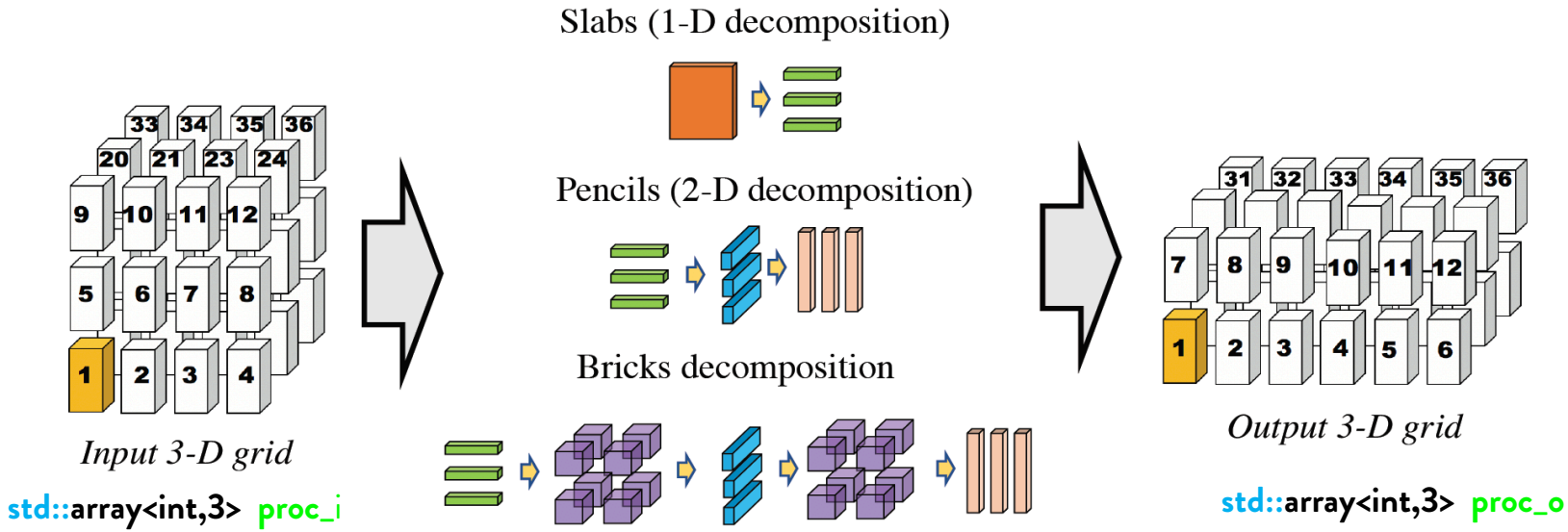


If user only has their MPI communicator and number of processors, we provide a routine to generate above grid of processors:

heffte ::*proc_setup_min_surface*(my_mpi_comm, nprocs);

heFFTe API

1. Definition of input/output processors grids (normally provided by users):



std::array<int,3> proc_i

std::array<int,3> proc_o

If user only has their MPI communicator and number of processors, we provide a routine to generate above grid of processors:

heffte ::*proc_setup_min_surface*(my_mpi_comm, nprocs);

2. Distribute data among processors using ***box3D*** *objects at input and output :*

```
std::vector<box3d<index>> inboxes  = heffte::split_world(world, proc_i);
std::vector<box3d<index>> outboxes = heffte::split_world(world, proc_o);
```

heFFTe API

4. Create FFT plan:

```
auto fft = heffte :: make_fft3d<backend_tag>(inboxes[me], outboxes[me], my_mpi_comm, options);
```

backend_tag: *Corresponds to the FFT library for local computations (e.g., FFTW3, CUFFT, MKL)*
options: *Contains information from flags set by users*

heFFTe API

4. Create FFT plan:

```
auto fft = heffte :: make_fft3d<backend_tag>(inboxes[me], outboxes[me], my_mpi_comm, options);
```

backend_tag: *Corresponds to the FFT library for local computations (e.g., FFTW3, CUFFT, MKL)*
options: *Contains information from flags set by users*

5. Compute an in-place parallel 3D FFT:

```
std::complex<my_precision_type> *output_array;

fft.forward(output_array, output_array, workspace.data(), scale::full);
fft.backward(output_array, output_array, workspace.data() );
```

workspace.data(): *Can be given by the user or calculated by heFFTe for stablishing a computation workspace*
scale::... : The scaling options are *full*, *none* and *symmetric*

heFFTe API

4. Create FFT plan:

```
auto fft = heffte :: make_fft3d<backend_tag>(inboxes[me], outboxes[me], my_mpi_comm, options);
```

backend_tag: *Corresponds to the FFT library for local computations (e.g., FFTW3, CUFFT, MKL)*
options: *Contains information from flags set by users*

5. Compute an in-place parallel 3D FFT:

```
std::complex<my_precision_type> *output_array;

fft.forward(output_array, output_array, workspace.data(), scale::full);
fft.backward(output_array, output_array, workspace.data() );
```

workspace.data(): *Can be given by the user or calculated by heFFTe for stablishing a computation workspace*
scale::... : The scaling options are *full*, *none* and *symmetric*

6. Tracing functionality can be added within your code to generate a runtime trace for performance analysis.

-D_HEFFTE_ENABLE_TRACE=ON

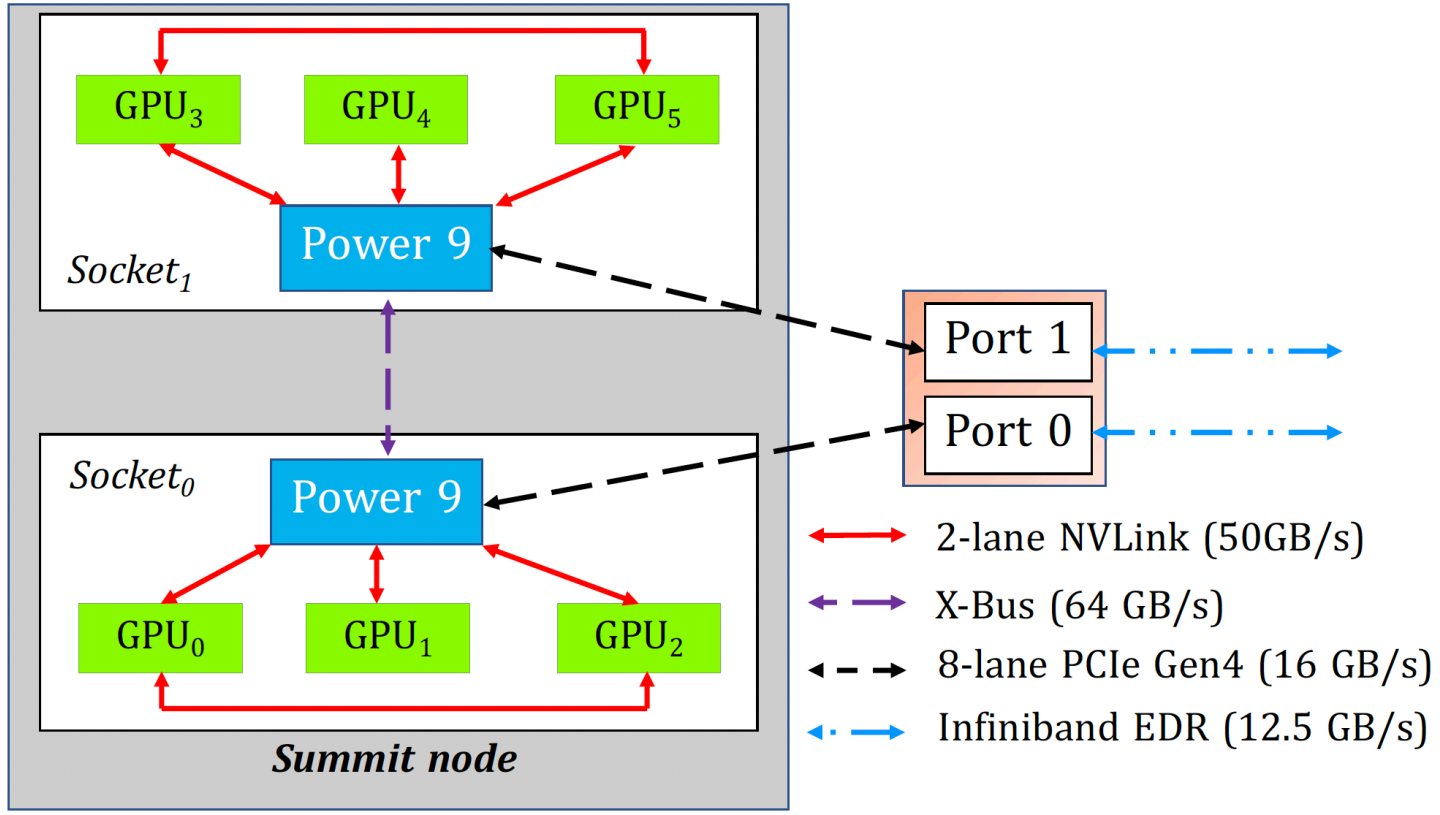
```
heffte::add_trace("Initiating tracing");

    ---Code to be traced ---

heffte::add_trace("Ending tracing");
```

Experimental Setup

Our experiments were performed using up to 1,024 Summit nodes, out of a total of 4,608. Each node consists of two sockets, each composed of a 22-core IBM POWER9 CPU and 3 NVIDIA Volta V100 GPUs.



Architecture of Summit nodes: computing units and network connections.

3. Experiments and results

2.1 Performance improvement from CPU to GPU

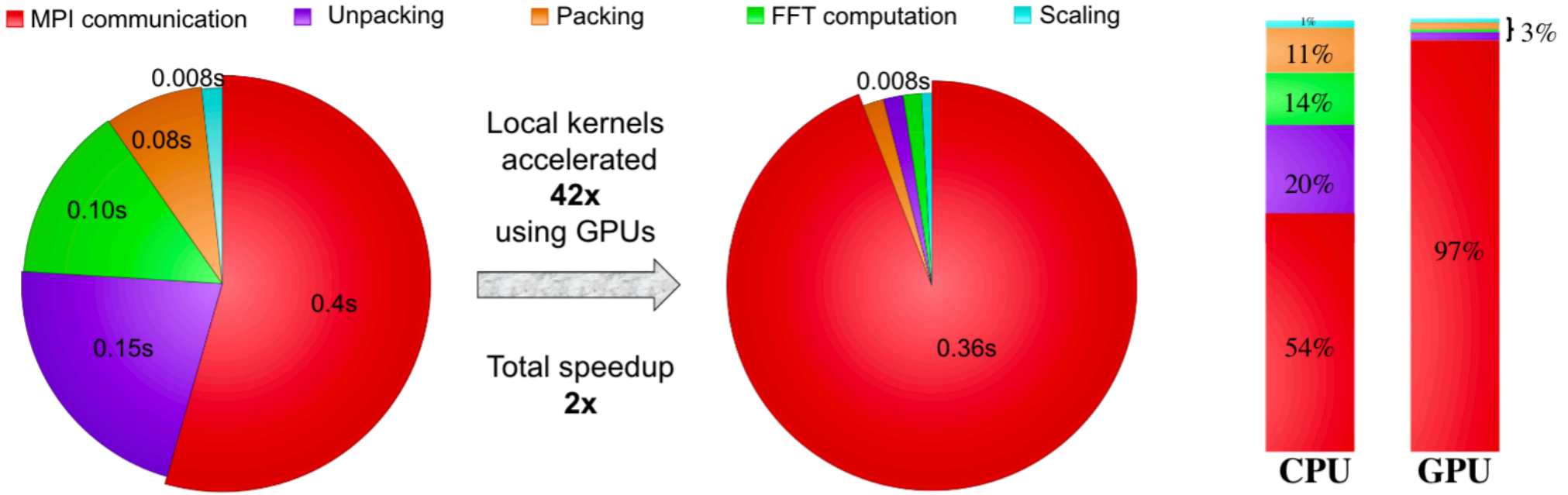
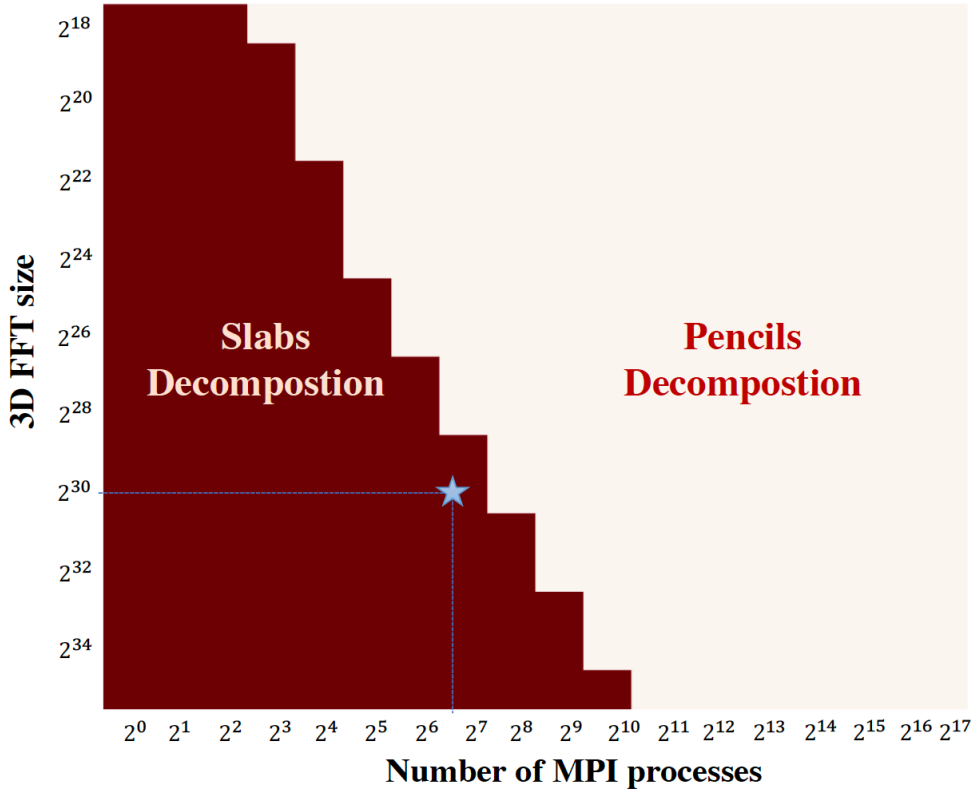
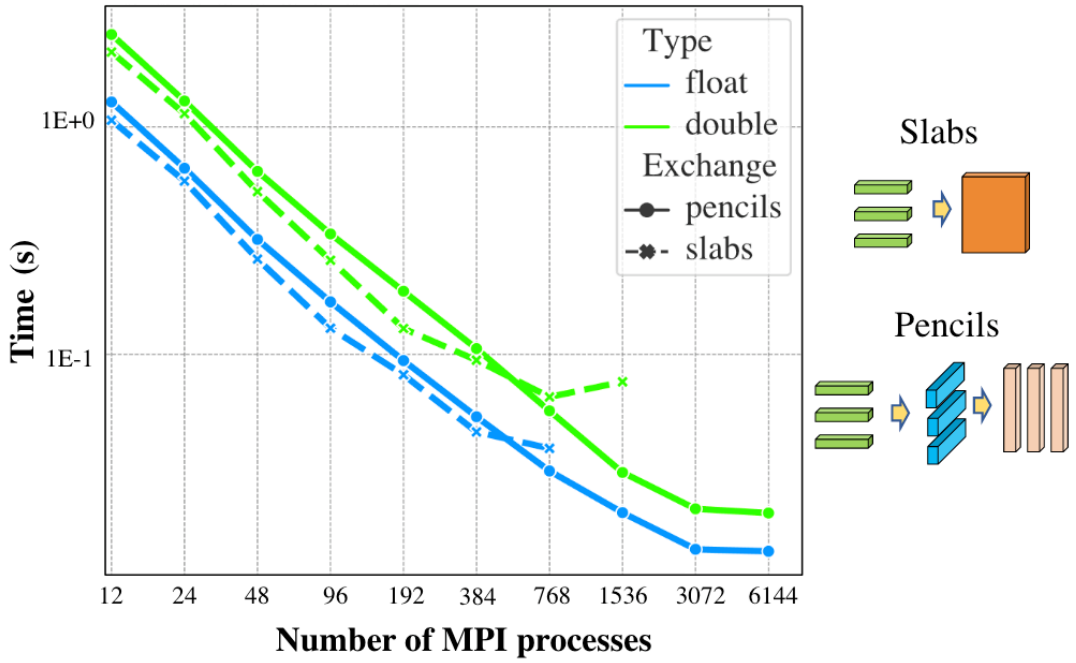


Fig. 4. Profile of a 3D FFT of size 1024^3 on 32 Summit nodes with all-to-all communication – using 40 MPI processes per node (left pie) and 6 MPIs per with 1 GPU per MPI (right pie).

2.2 Strong Scalability - Summit

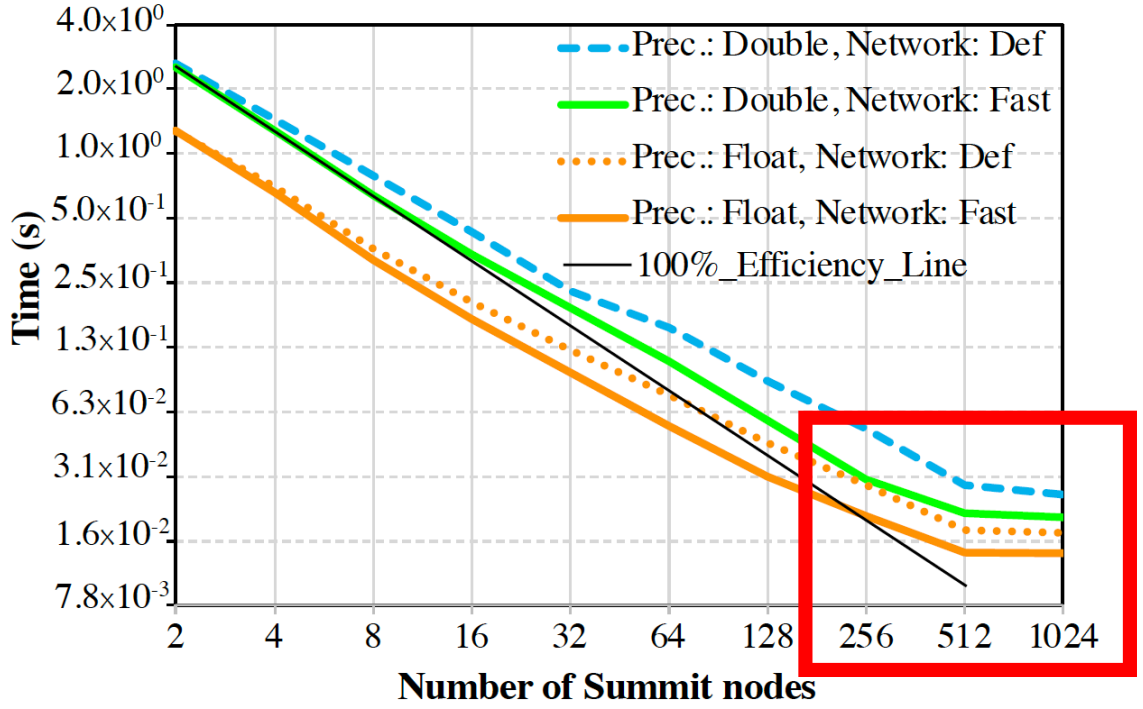


Selection of the best reshape approach based on the 3-D FFT size and the number of resources.

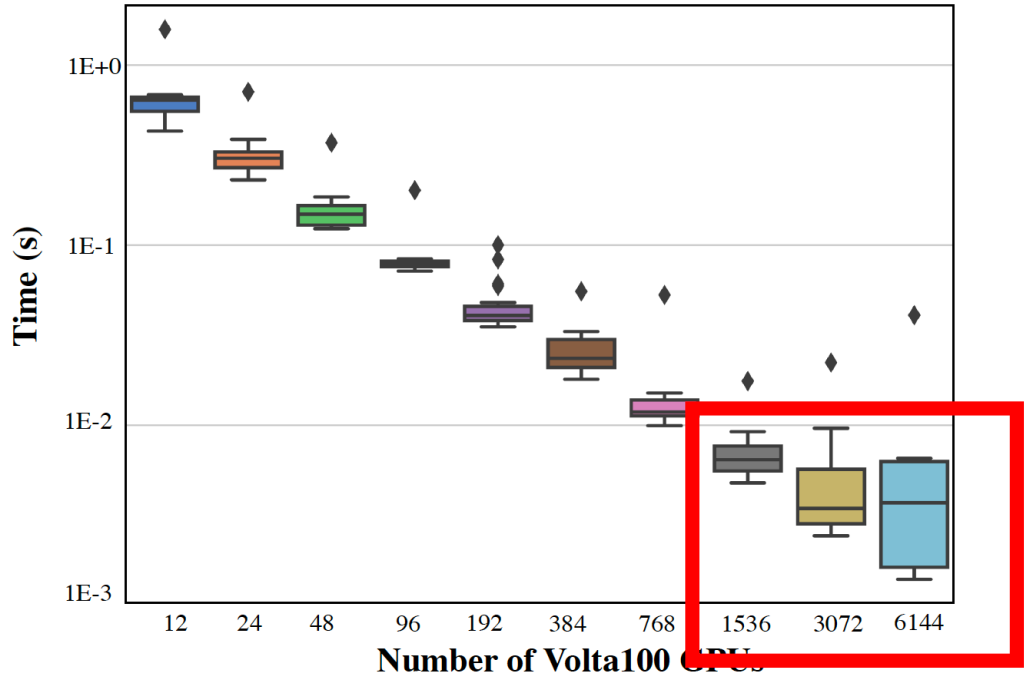


Comparison of pencil and slab decompositions for strong scaling of a 3-D FFT of size 1024^3. Using *heFFTe* with cuFFT backend, 6 MPI processes (1 MPI processes per GPU-V100) per node, and single-precision complex data.

2.3 Communication bottleneck

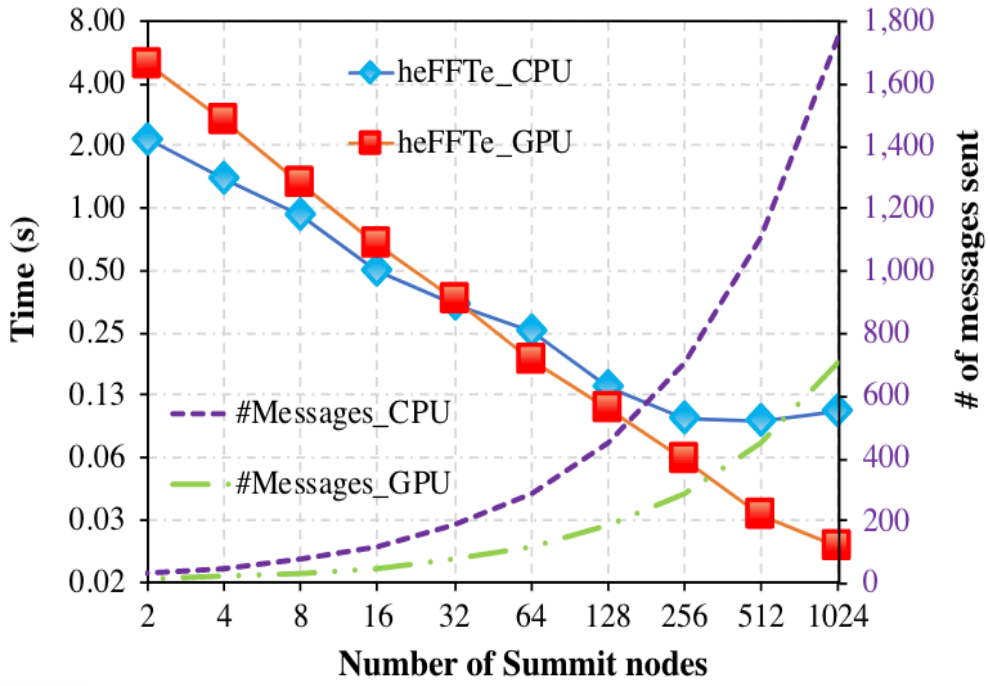


Comparison of strong scaling (with fixed size 1024^3) using 6 MPI processes (1 MPI processes per GPU-V100) per node, for different interconnection options, using MPI_Alltoall communication.

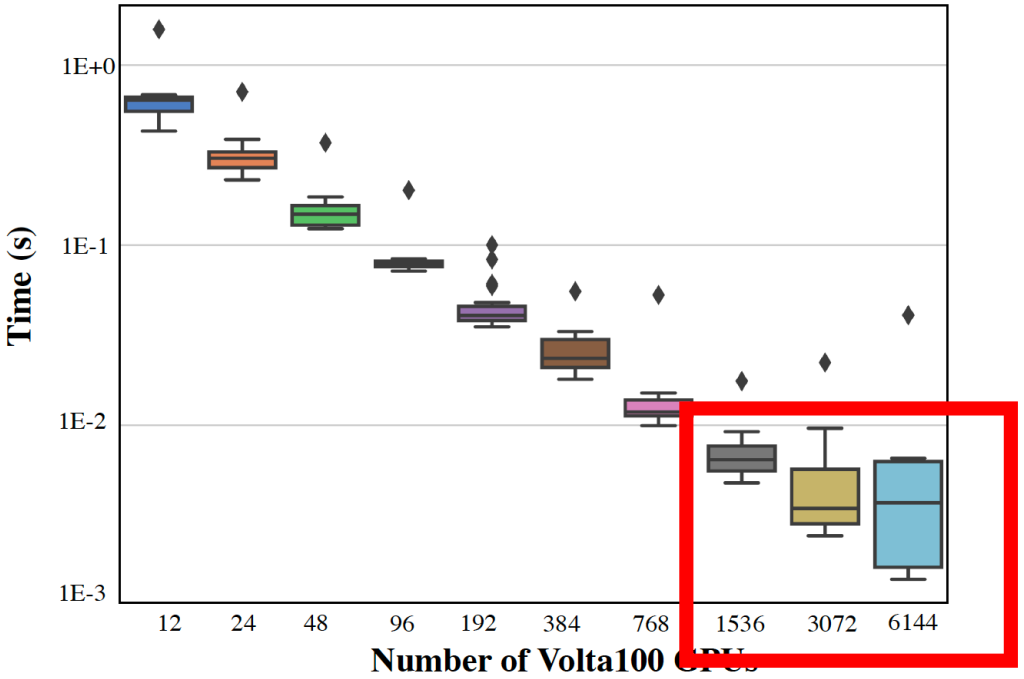


Scalability of MPI_Alltoall communication for a double-complex precision of a 3-D FFT of size 1024^3 with cuFFT backend and 6 V100-GPUs per node.

2.3 Communication bottleneck



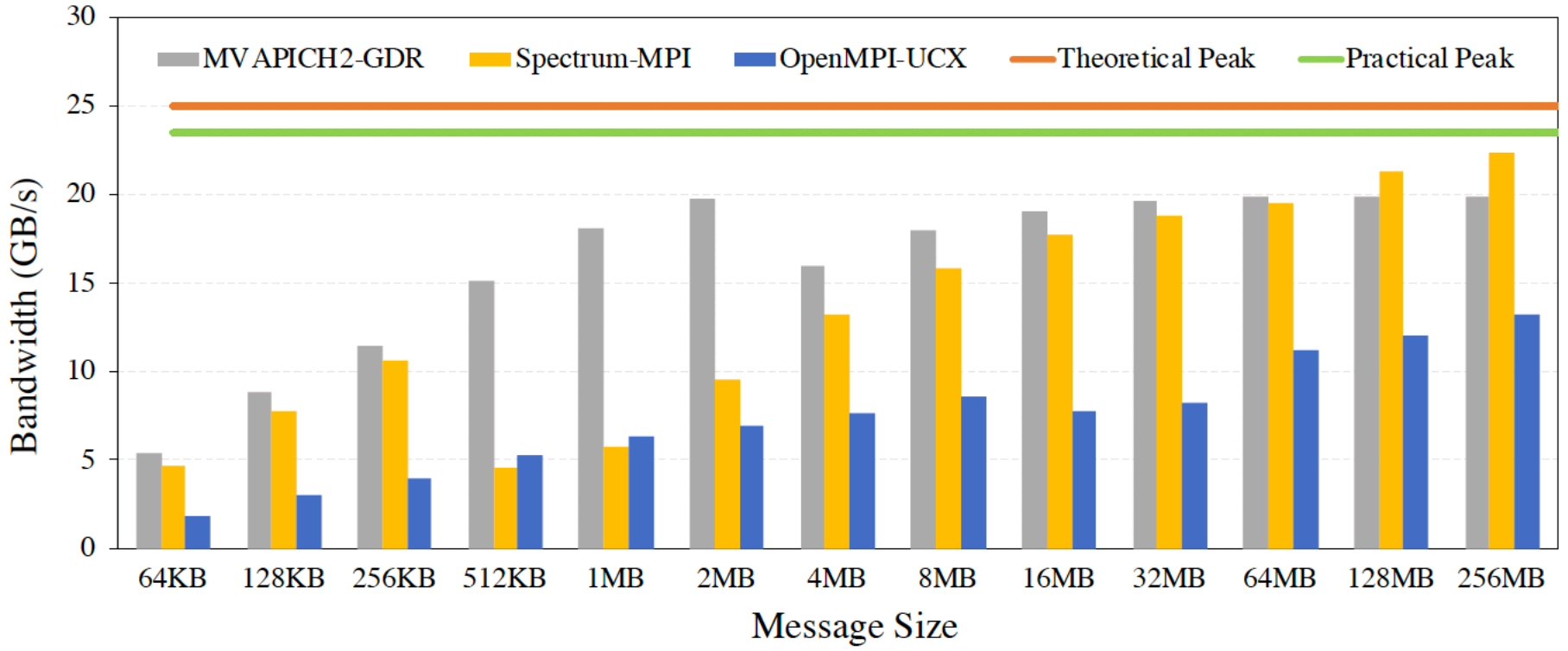
Time for MPI_Alltoall communication and number of message sent for a 3D FFT of size 1024^3. Using 40 MPIs and 6 GPUs per node.



Scalability of MPI_Alltoall communication for a double-complex precision of a 3-D FFT of size 1024^3 with cuFFT backend and 6 V100-GPUs per node.

3.1 Performance analysis - MPI

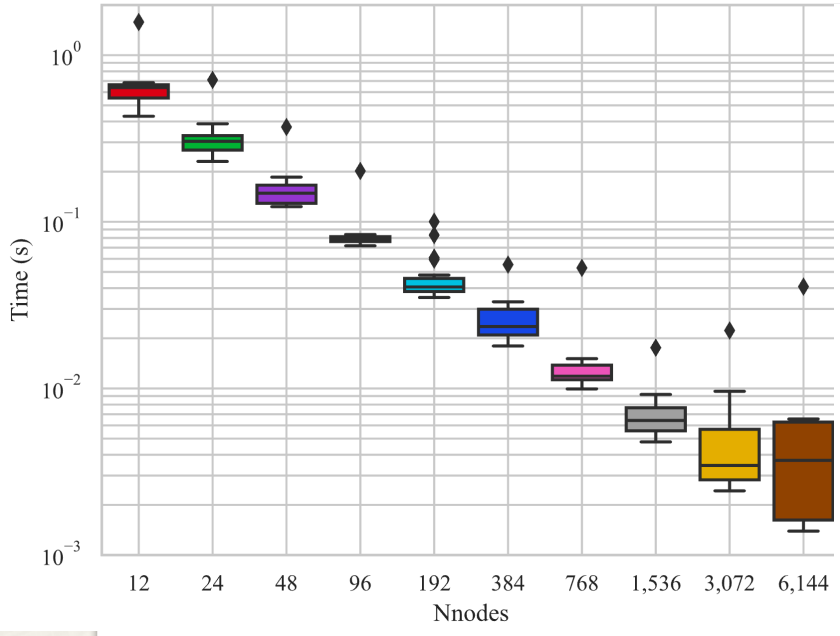
Our experiments show that further tuning of MPI parameters and network topologies can help to get faster computation, specially for small size number of resources.



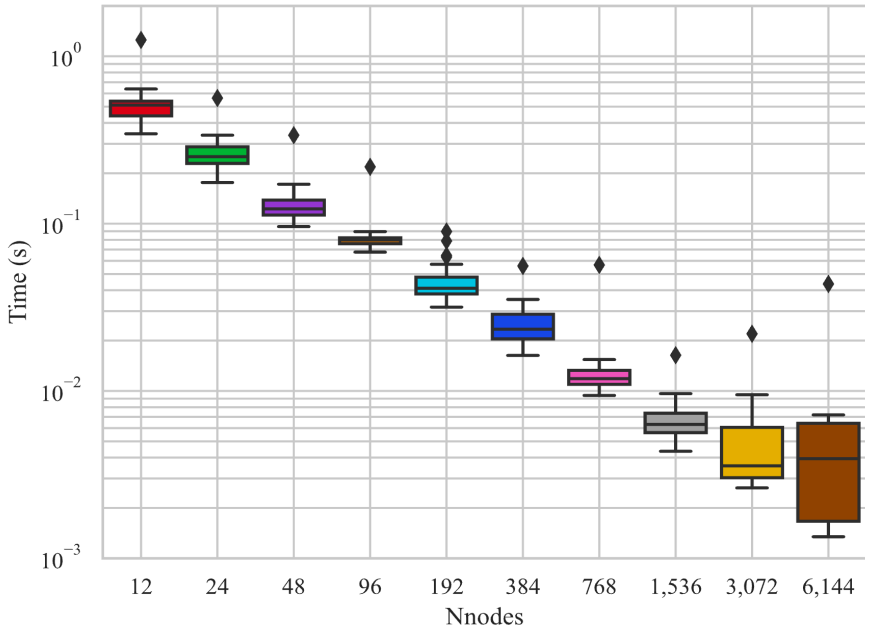
Comparison of achievable bandwidth from two-node exchange via MPI_Send, using of MVAPICH, SpectrumMPI and OpenMPI-UCX on Summit.

3.2 Impact of MPI on scaling

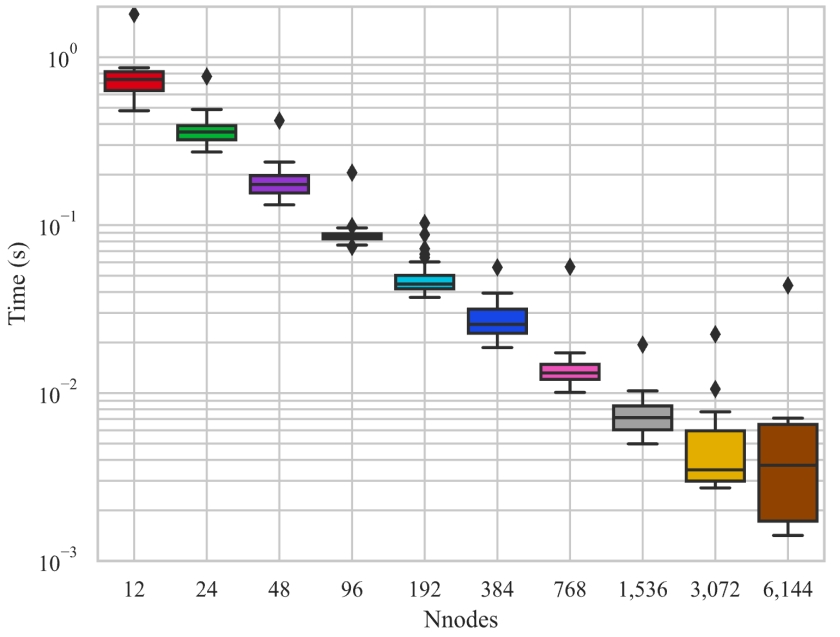
In the following figures we explore the scalability of MPI_AlltoAllv via a heFFTe experiment on a 1024^3 complex-to-complex 3-D FFT, using 6 NVIDIA V100GPUs per node.



Spectrum MPI 10.3

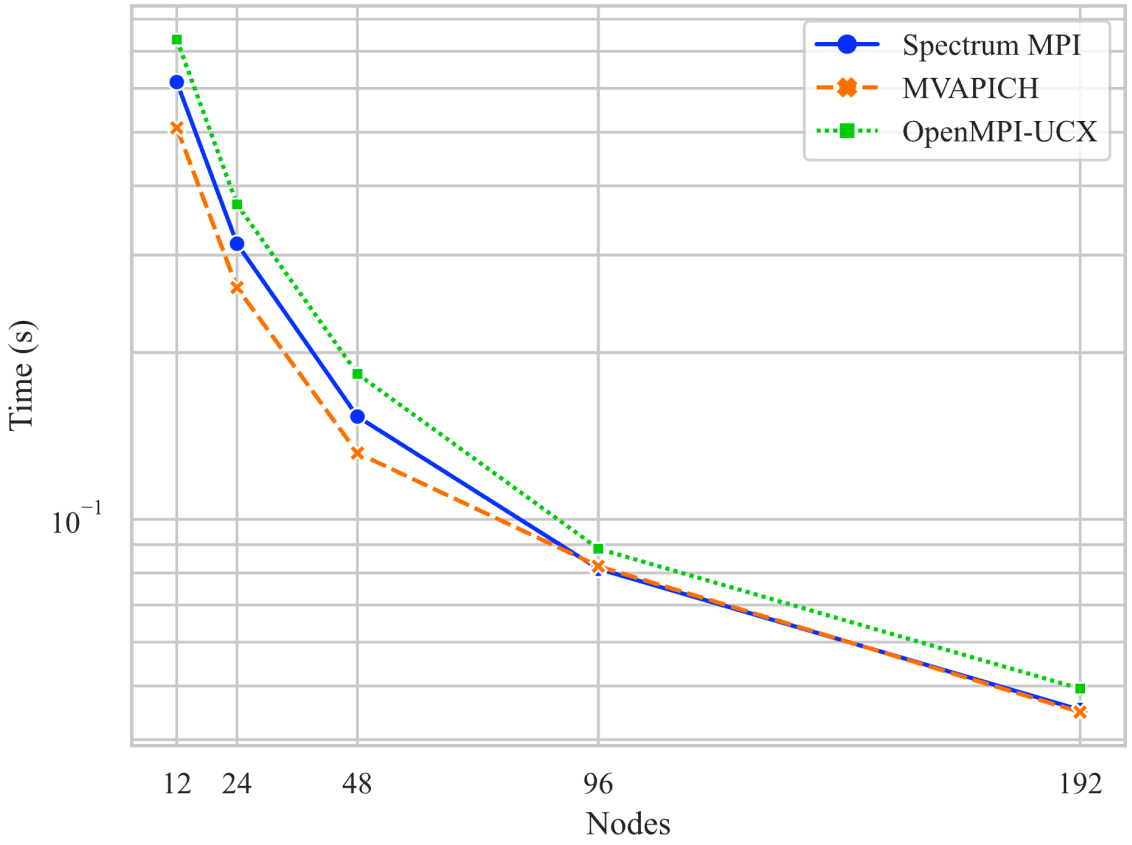


MVPICH-GDR 2.3.5

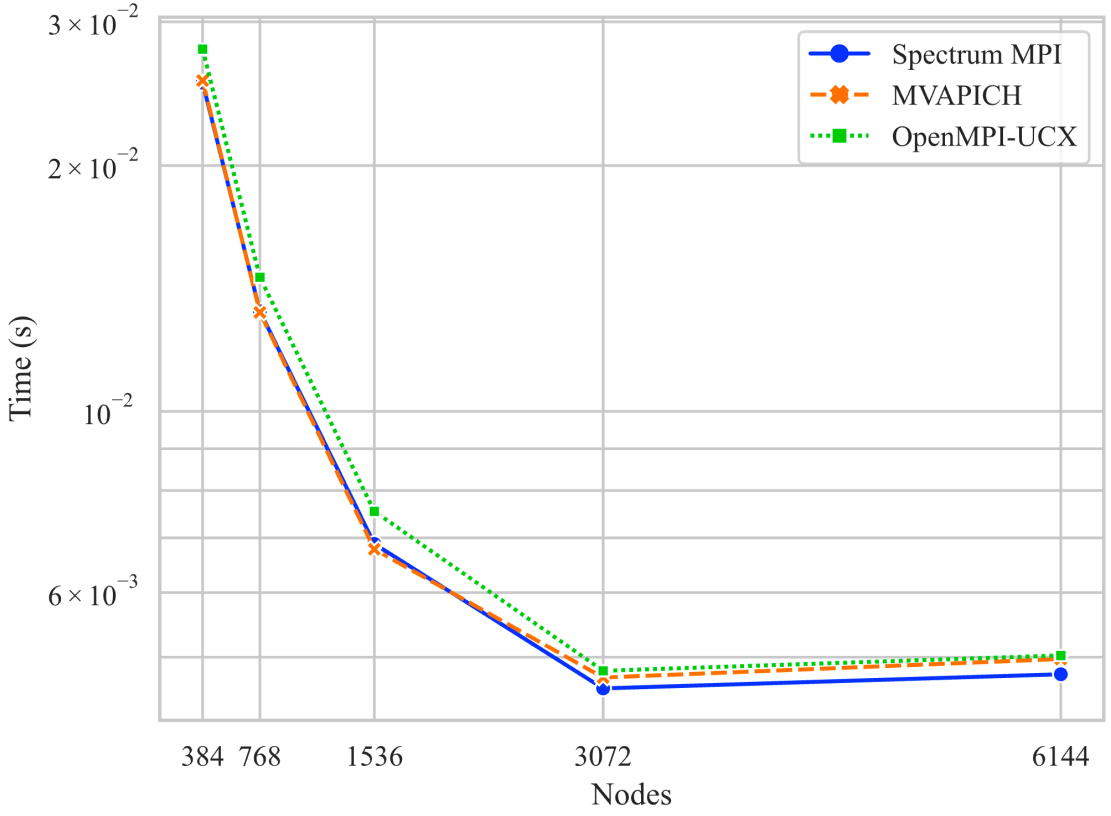


OpenMPI + UCX

3.2 Impact of MPI on scaling



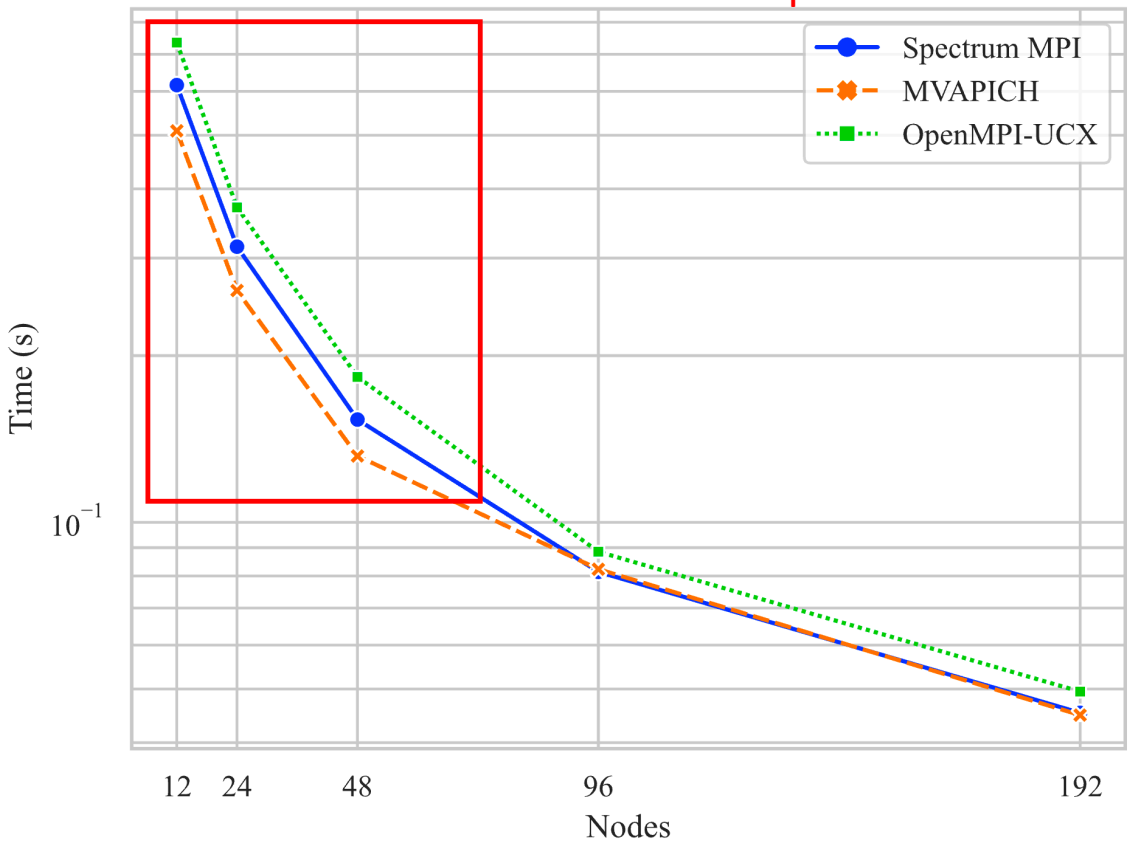
Up to 64 nodes (192 GPUs)



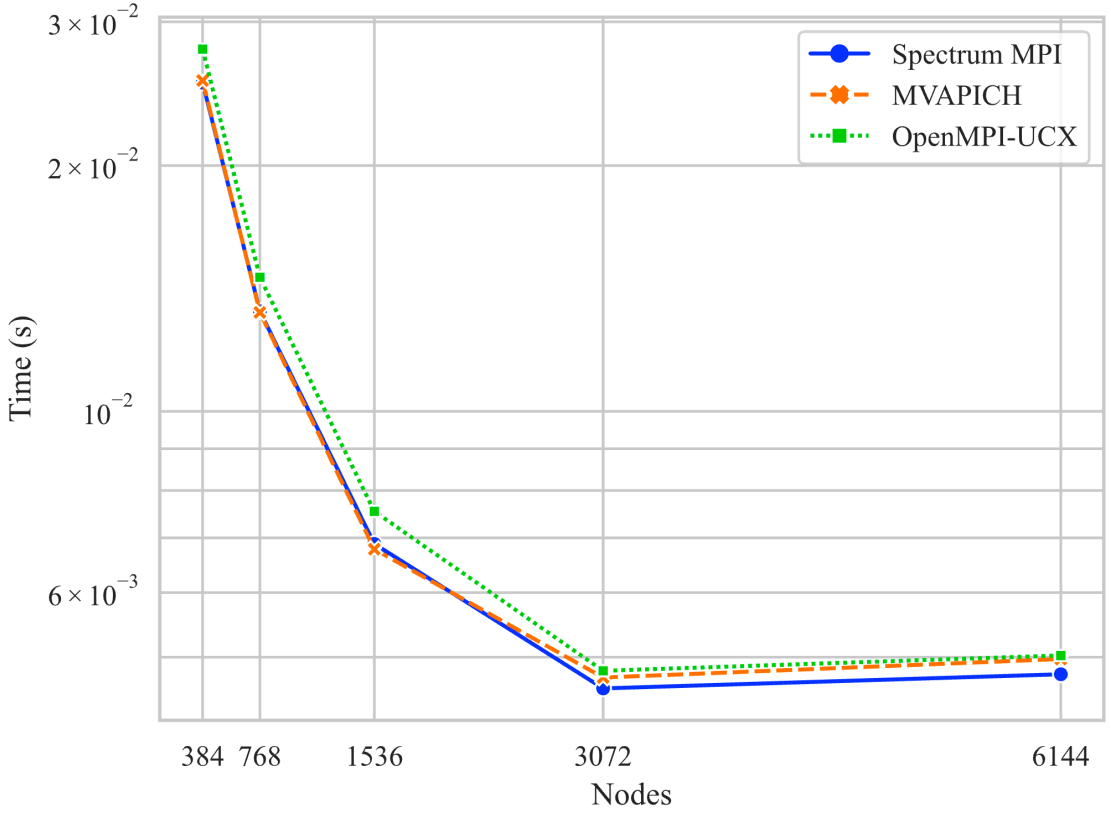
From 128 to 1024 nodes (6144 GPUs)

3.2 Impact of MPI on scaling

MVAPICH is around 10-20% faster than SpectrumMPI 10.3



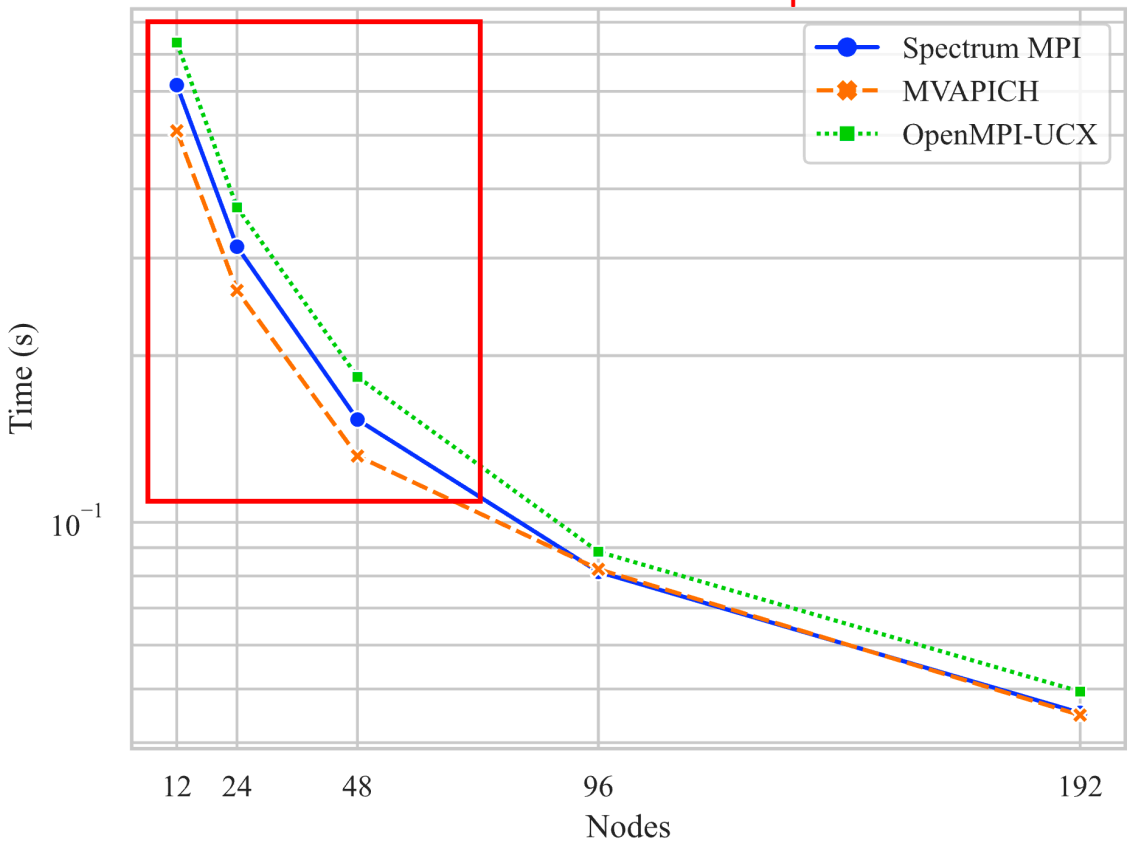
Up to 64 nodes (192 GPUs)



From 128 to 1024 nodes (6144 GPUs)

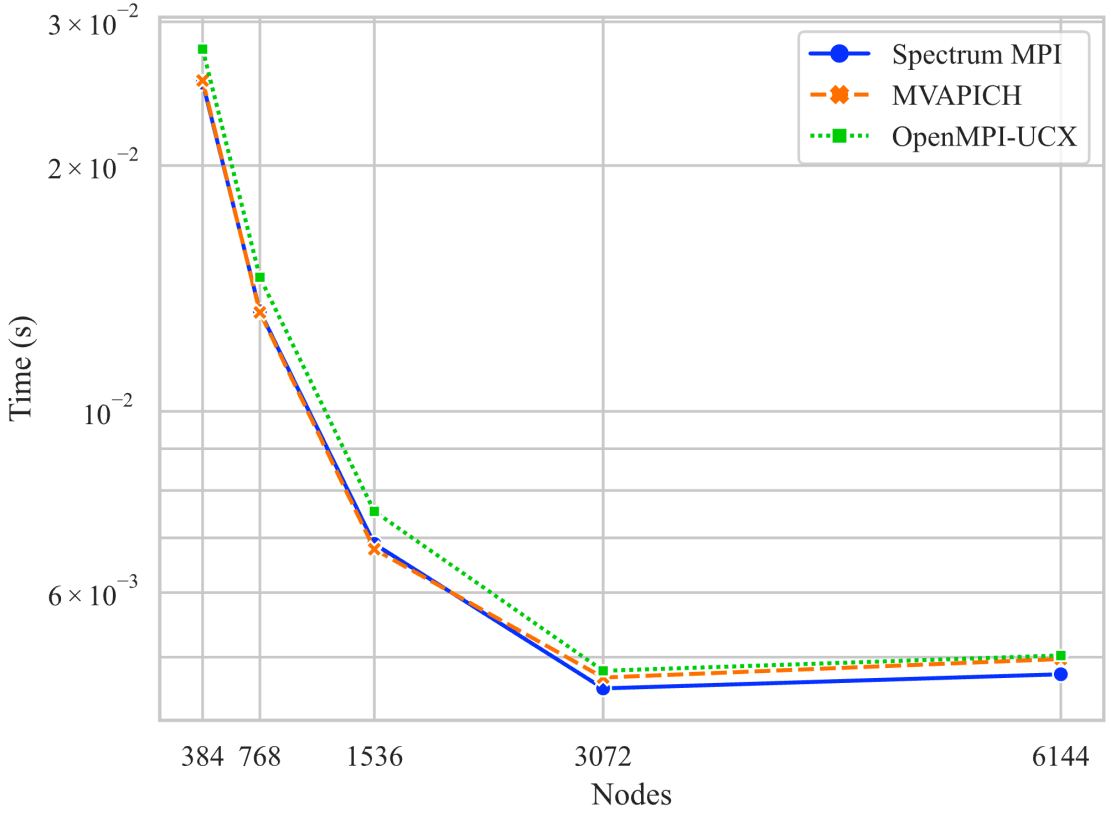
3.2 Impact of MPI on scaling

MVAPICH is around 10-20% faster than SpectrumMPI 10.3



Up to 64 nodes (192 GPUs)

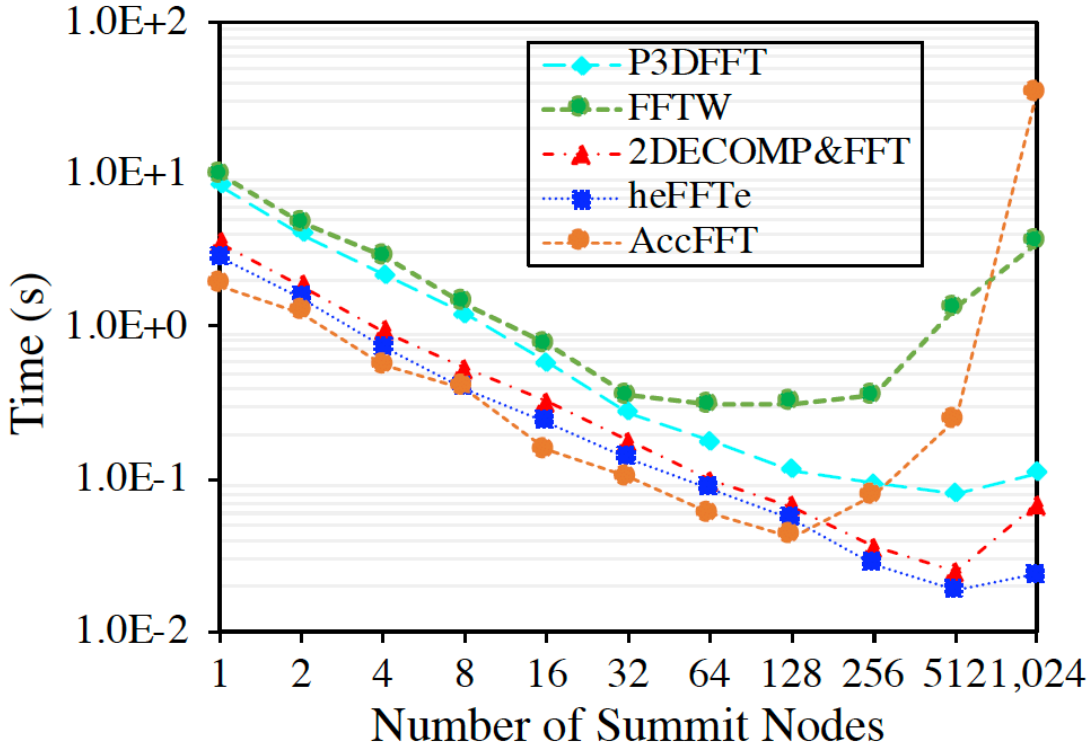
No much difference for large number of nodes



From 128 to 1024 nodes (6144 GPUs)

3.3 Performance analysis

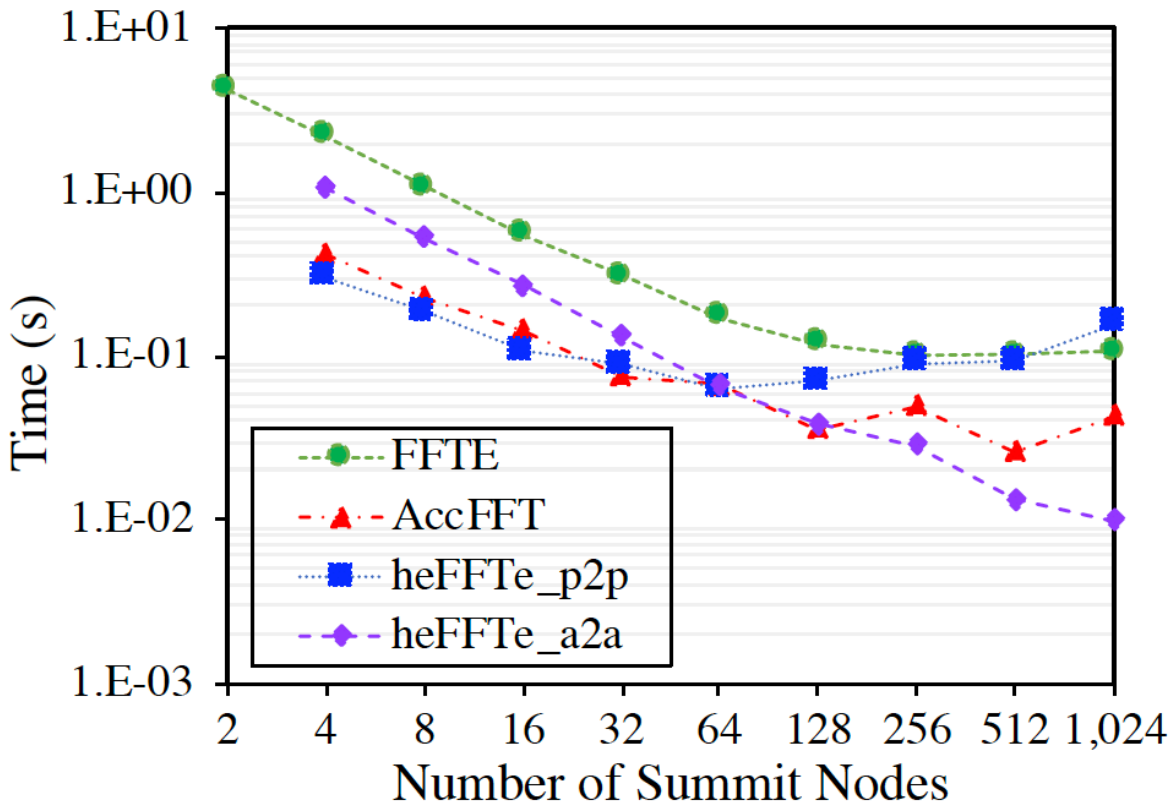
In the following figure we compare the runtime to compute a 3-D transform of size 1024^3, using double-precision complex random input.



Strong scalability for a 3-D FFT of size 1024^3 using 40 MPI processes per node, 1 per IBM Power9 core, 20 per socket. Using 2 reshapes (transpositions) per FFT direction.

3.3 Performance analysis

In the following figure we compare the runtime to compute a 3-D transform of size 1024^3, using double-precision complex random input.



Strong scalability for GPU libraries using 4 NVIDIA V100 GPUs per node, 2 per socket.

Conclusions

- We presented an analysis of performance and scalability for 3-D parallel FFT libraries and analyzed the computation on upcoming large scale systems.

- Scalability of FFTs is highly impacted by how libraries handle communication for a large number of process units.

- Tuning can help to further accelerate the performance of current FFT software. The choice of the MPI distribution can help to further speedup; however, this is architecture dependent.

- Scalability towards Exascale would require (auto) tuning of algorithmic parameters and bandwidth management.