Towards Exascale FFT Computation

A survey, benchmark, and MPI Challenges

A. Ayala[1] S. Tomov[1] M. Stoyanov[2] J. Dongarra[1,2]

[1]Innovative Computing Laboratory
University of Tennessee at Knoxville

[2]Oak Ridge National Laboratory

The 10^{th} Annual MVAPICH User Group Meeting

Section summary

The Fast Fourier Transform (FFT)

- The FFT is an algorithm developed by Cooley-Tukey in 1965
- Considered one of the top 10 algorithms of the 20^{th} century.

Discrete Fourier Transform (DFT)

Let x be an m-dimensional array of size $N := N_1 \times N_2 \times \cdots \times N_m$. Its DFT is defined by $y := DFT(x)$, obtained as:

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

where $\tilde{x} := x(n_1, n_2, \ldots, n_m)$.

- A naive DFT costs $\mathcal{O}(N^2)$
- Using the FFT, the cost can be reduced to $\mathcal{O}(N \log_2 N)$.

The Fast Fourier Transform (FFT)

- The FFT is an algorithm developed by Cooley-Tukey in 1965
- Considered one of the top 10 algorithms of the 20^{th} century.

Discrete Fourier Transform (DFT)

Let x be an m-dimensional array of size $N := N_1 \times N_2 \times \cdots \times N_m$. Its DFT is defined by $y := DFT(x)$, obtained as:

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

where $\tilde{x} := x(n_1, n_2, \ldots, n_m)$.

- A naive DFT costs $\mathcal{O}(N^2)$
- Using the FFT, the cost can be reduced to $\mathcal{O}(N \log_2 N)$.

The Fast Fourier Transform (FFT)

- The FFT is an algorithm developed by Cooley-Tukey in 1965
- Considered one of the top 10 algorithms of the 20^{th} century.

Discrete Fourier Transform (DFT)

Let x be an m-dimensional array of size $N := N_1 \times N_2 \times \cdots \times N_m$. Its DFT is defined by $y := DFT(x)$, obtained as:

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

where $\tilde{x} := x(n_1, n_2, \ldots, n_m)$.

- A naive DFT costs $\mathcal{O}(N^2)$
- Using the FFT, the cost can be reduced to $\mathcal{O}(N \log_2 N)$.

The Fast Fourier Transform (FFT)

- The FFT is an algorithm developed by Cooley-Tukey in 1965
- Considered one of the top 10 algorithms of the 20^{th} century.

Discrete Fourier Transform (DFT)

Let x be an m-dimensional array of size $N := N_1 \times N_2 \times \cdots \times N_m$. Its DFT is defined by $y := DFT(x)$, obtained as:

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

where $\tilde{x} := x(n_1, n_2, \ldots, n_m)$.

- A naive DFT costs $\mathcal{O}(N^2)$
- Using the FFT, the cost can be reduced to $\mathcal{O}(N \log_2 N)$.

The Fast Fourier Transform (FFT)

- The FFT is an algorithm developed by Cooley-Tukey in 1965
- Considered one of the top 10 algorithms of the 20^{th} century.

Discrete Fourier Transform (DFT)

Let x be an m-dimensional array of size $N := N_1 \times N_2 \times \cdots \times N_m$. Its DFT is defined by $y := DFT(x)$, obtained as:

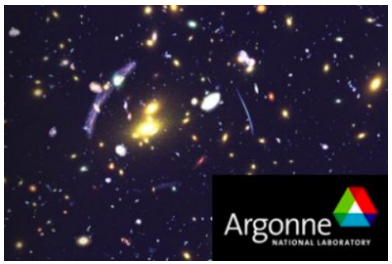
$$y(k_1, k_2, \ldots, k_m) := \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \cdots \sum_{n_m=0}^{N_m-1} \tilde{x} \cdot e^{-2\pi i \left(\frac{k_1 n_1}{N_1} + \frac{k_2 n_2}{N_2} \cdots + \frac{k_m n_m}{N_m} \right)},$$

where $\tilde{x} := x(n_1, n_2, \ldots, n_m)$.

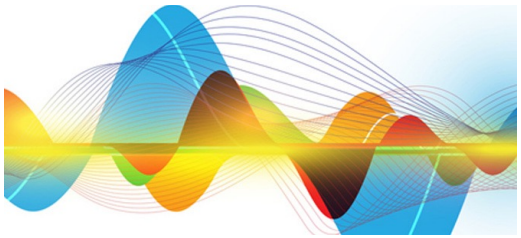
- A naive DFT costs $\mathcal{O}(N^2)$
- Using the FFT, the cost can be reduced to $\mathcal{O}(N \log_2 N)$.

Applications Relying on Parallel FFTs

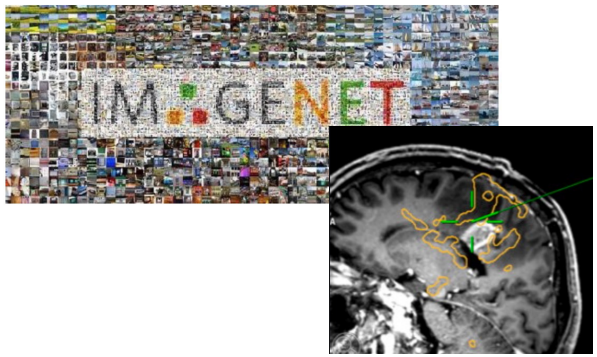
Cosmology
***ECP* ExaSky - HACC**

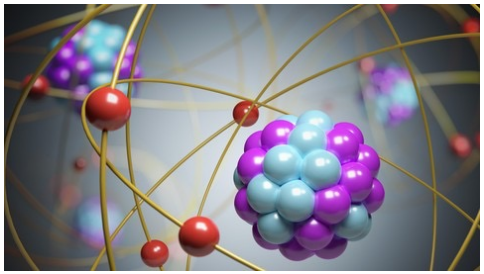


Signal processing,
***ECP* WARPX**

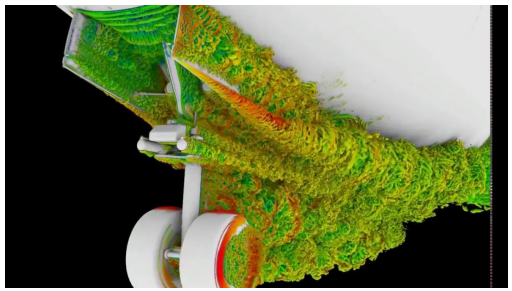


Deep Learning

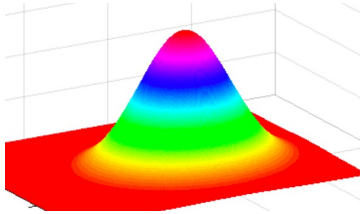




Molecular Dynamics
***ECP* EXAALT**



Particle Simulations
***ECP* CoPa / Cabana**



PDE solutions, **MASSIF**

Figure: Several applications from the U.S. ECP project heavily rely on FFTs.

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	✓	✓	✓	✓

Figure: State-of-the-art of FFT libraries targeting a single-device unit.

Ref.: Interim Report on Benchmarking FFT Libraries on High Performance Systems *Ayala et al., ICL Tech Report 2021.*

Single-Device FFT Comparison

- Useful when input data is small or can be batched.
- heFFTe provides portability to run FFT experiment on different devices.

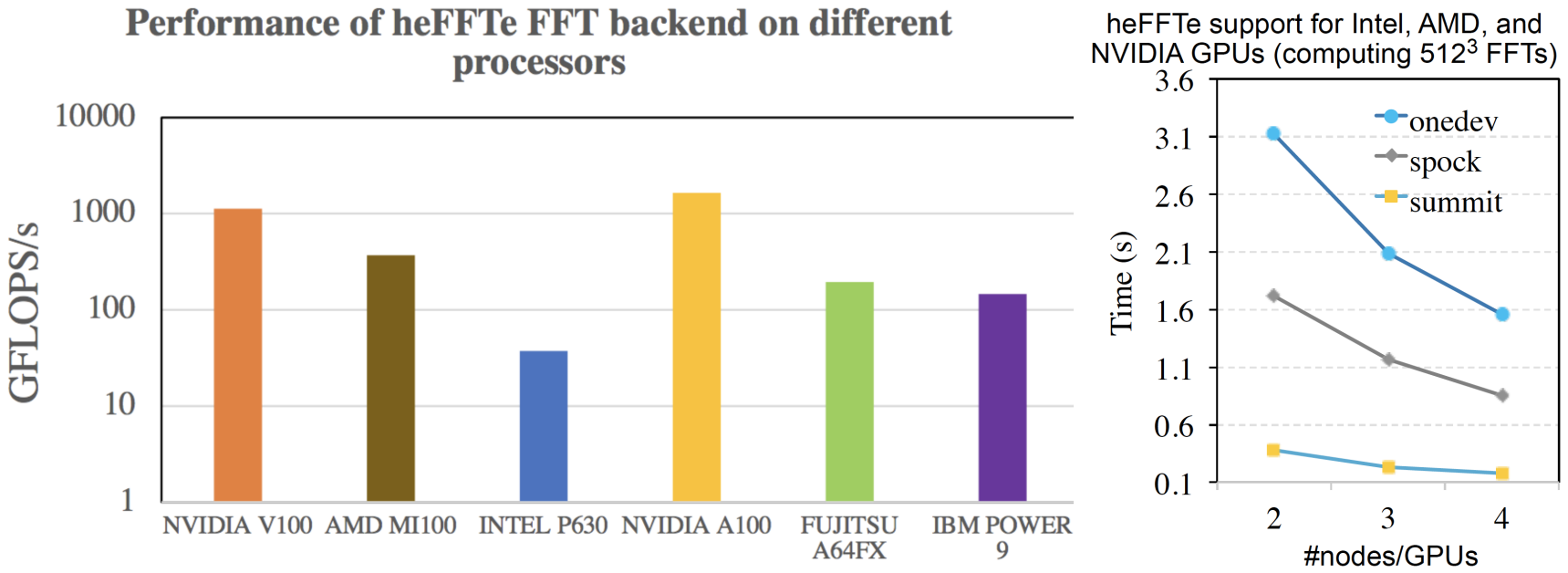


Figure: Comparison of single-device performance for a 512^3 FFT.

Parallel FFT implementation

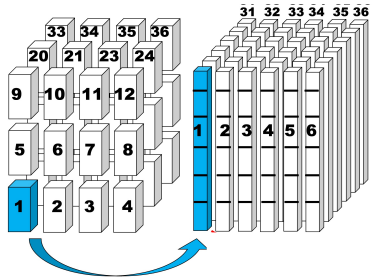
Algorithm 1 Parallel 3-D FFT computation on GPUs

1: **Input:** 3-D array, processor grids: P_{in}, P_{out}
2: Transfer data from P_{in} to a pencil or slab grid
3: Define processor grids (MPI groups) for each direction
4: **for** $r \leftarrow 1, \cdots, n_{\text{exchanges}}$ **do**
5: Compute local 1-D or 2-D FFTs on the GPUs
6: *Pack* data in contiguous memory
7: **for** P on my MPI group **do**
8: *Transfer* computed data to neighbor processes
9: **end for**
10: *Unpack* data in contiguous memory
11: **end for**
12: Transfer data from the pencil or slab grid to P_{out}

Parallel FFT implementation

Algorithm 1 Parallel 3-D FFT computation on GPUs

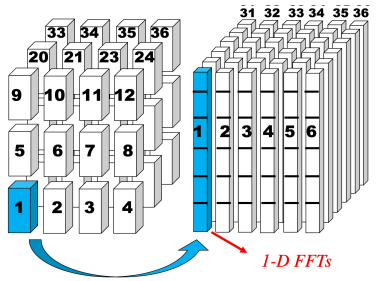
1: **Input:** 3-D array, processor grids: P_{in}, P_{out}
2: Transfer data from P_{in} to a pencil or slab grid
3: Define processor grids (MPI groups) for each direction
4: **for** $r \leftarrow 1, \cdots, n_{\text{exchanges}}$ **do**
5: Compute local 1-D or 2-D FFTs on the GPUs
6: *Pack* data in contiguous memory
7: **for** P on my MPI group **do**
8: *Transfer* computed data to neighbor processes
9: **end for**
10: *Unpack* data in contiguous memory
11: **end for**
12: Transfer data from the pencil or slab grid to P_{out}



Parallel FFT implementation

Algorithm 1 Parallel 3-D FFT computation on GPUs

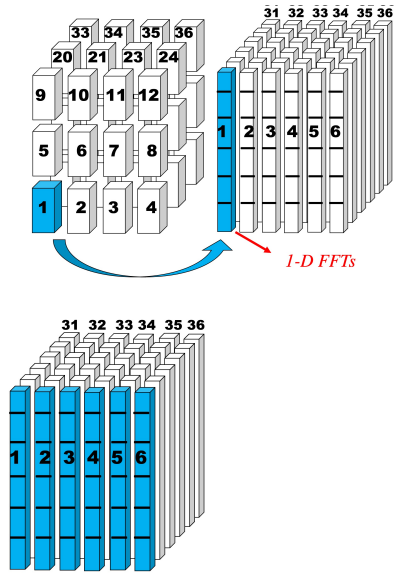
1: **Input:** 3-D array, processor grids: P_{in}, P_{out}
2: Transfer data from P_{in} to a pencil or slab grid
3: Define processor grids (MPI groups) for each direction
4: **for** $r \leftarrow 1, \cdots, n_{\text{exchanges}}$ **do**
5: Compute local 1-D or 2-D FFTs on the GPUs
6: *Pack* data in contiguous memory
7: **for** P on my MPI group **do**
8: *Transfer* computed data to neighbor processes
9: **end for**
10: *Unpack* data in contiguous memory
11: **end for**
12: Transfer data from the pencil or slab grid to P_{out}



Parallel FFT implementation

Algorithm 1 Parallel 3-D FFT computation on GPUs

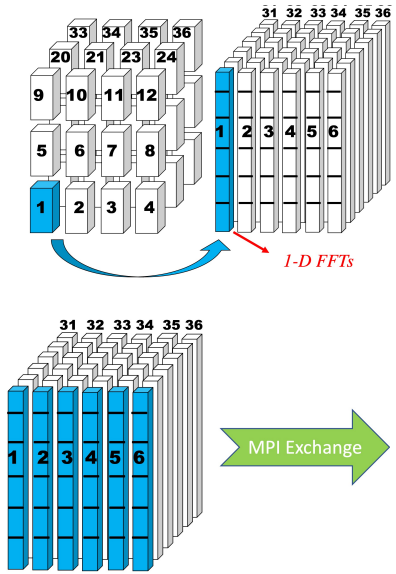
1: **Input:** 3-D array, processor grids: P_{in}, P_{out}
2: Transfer data from P_{in} to a pencil or slab grid
3: Define processor grids (MPI groups) for each direction
4: **for** $r \leftarrow 1, \cdots, n_{\text{exchanges}}$ **do**
5: Compute local 1-D or 2-D FFTs on the GPUs
6: *Pack* data in contiguous memory
7: **for** P on my MPI group **do**
8: *Transfer* computed data to neighbor processes
9: **end for**
10: *Unpack* data in contiguous memory
11: **end for**
12: Transfer data from the pencil or slab grid to P_{out}



Parallel FFT implementation

Algorithm 1 Parallel 3-D FFT computation on GPUs

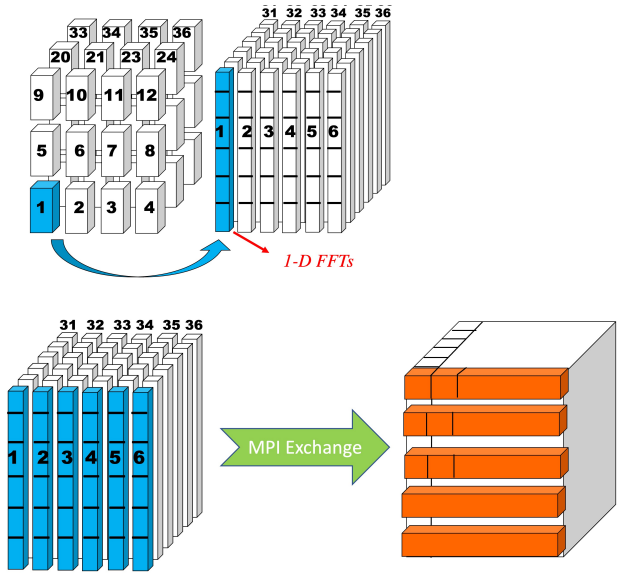
1: **Input:** 3-D array, processor grids: P_{in}, P_{out}
2: Transfer data from P_{in} to a pencil or slab grid
3: Define processor grids (MPI groups) for each direction
4: **for** $r \leftarrow 1, \cdots, n_{\text{exchanges}}$ **do**
5: Compute local 1-D or 2-D FFTs on the GPUs
6: *Pack* data in contiguous memory
7: **for** P on my MPI group **do**
8: *Transfer* computed data to neighbor processes
9: **end for**
10: *Unpack* data in contiguous memory
11: **end for**
12: Transfer data from the pencil or slab grid to P_{out}



Parallel FFT implementation

Algorithm 1 Parallel 3-D FFT computation on GPUs

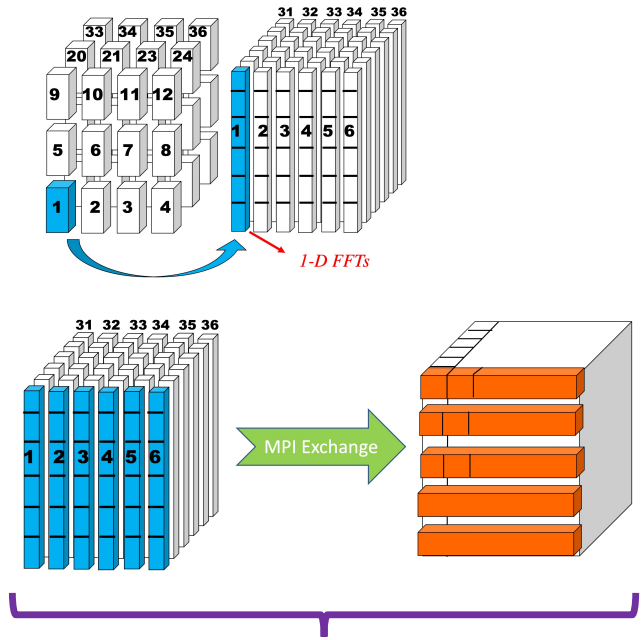
1: **Input:** 3-D array, processor grids: P_{in}, P_{out}
2: Transfer data from P_{in} to a pencil or slab grid
3: Define processor grids (MPI groups) for each direction
4: **for** $r \leftarrow 1, \cdots, n_{\text{exchanges}}$ **do**
5: Compute local 1-D or 2-D FFTs on the GPUs
6: *Pack* data in contiguous memory
7: **for** P on my MPI group **do**
8: *Transfer* computed data to neighbor processes
9: **end for**
10: *Unpack* data in contiguous memory
11: **end for**
12: Transfer data from the pencil or slab grid to P_{out}



Parallel FFT implementation

Algorithm 1 Parallel 3-D FFT computation on GPUs

1: **Input:** 3-D array, processor grids: P_{in}, P_{out}
2: Transfer data from P_{in} to a pencil or slab grid
3: Define processor grids (MPI groups) for each direction
4: **for** $r \leftarrow 1, \cdots, n_{\text{exchanges}}$ **do**
5: Compute local 1-D or 2-D FFTs on the GPUs
6: *Pack* data in contiguous memory
7: **for** P on my MPI group **do**
8: *Transfer* computed data to neighbor processes
9: **end for**
10: *Unpack* data in contiguous memory
11: **end for**
12: Transfer data from the pencil or slab grid to P_{out}



These 3 tasks can be replaced by 1 via MPI_Alltoallw

Communication can be accelerated by enabling Mixed-Precision, c.f., Advances in Mixed Precision Algorithms: 2021 Edition. *Abdelfattah et al., LLNL-TR-825909*

Parallel FFT Libraries

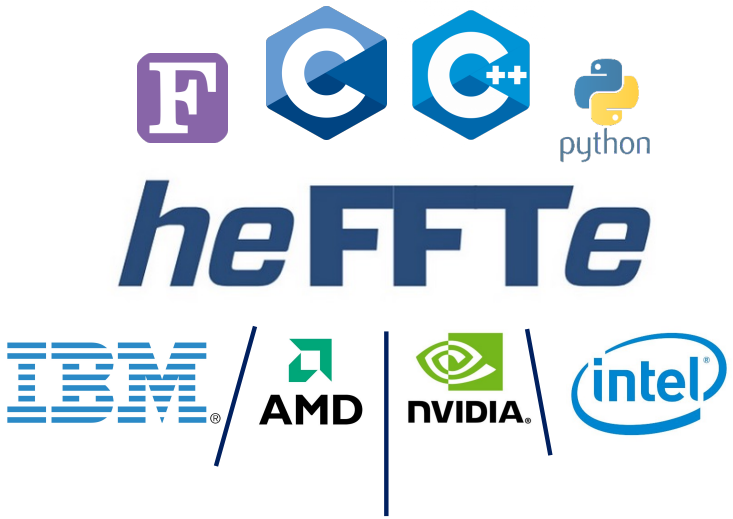
Library	Developer	Language	CPU Backend	GPU Backend	Real-to-Complex	Slab Decomp.	Brick Decomp.
2DECOMP&FFT	NAG	Fortran	FFTW3, ESSL	-	✓	✓	
AccFFT	Georgia Tech	C++	FFTW3	CUFFT	✓		
Cluster FFT	Intel	Fortran	MKL	-			
CRAFFT	Cray	Fortran	FFTW3	-	✓		
cuFFTMp	NVIDIA	C	-	CUFFT	✓		
FFTE	U. Tsukuba / Riken	Fortran	FFTE	CUFFT	✓	✓	
fftMPI	Sandia	C++	FFTW3, MKL, KISS	-			✓
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	-			✓

Figure: State-of-the-art of FFT libraries targeting parallel systems.

Ref.: Interim Report on Benchmarking FFT Libraries on High Performance Systems *Ayala et al., ICL Tech Report 2021.*

The Highly Efficient FFT for Exascale (heFFTe)

- heFFTe is part of the US Exascale Project (ECP).
- Funded by DoE, it aims to provide reliable FFT computation on exascale systems.
- Integrated to ECP projects such as Copa-Cabana.
- Unique features: Batched FFT, Convolution, Sine/Cosine transform, Mixed Precision FFT.



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

Figure: heFFTe interfaces and architecture support.

Moving FFT Kernels to GPUs

- Moving local kernels to GPUs yields high speedups.
- In the following example, we show this effect on Summit.

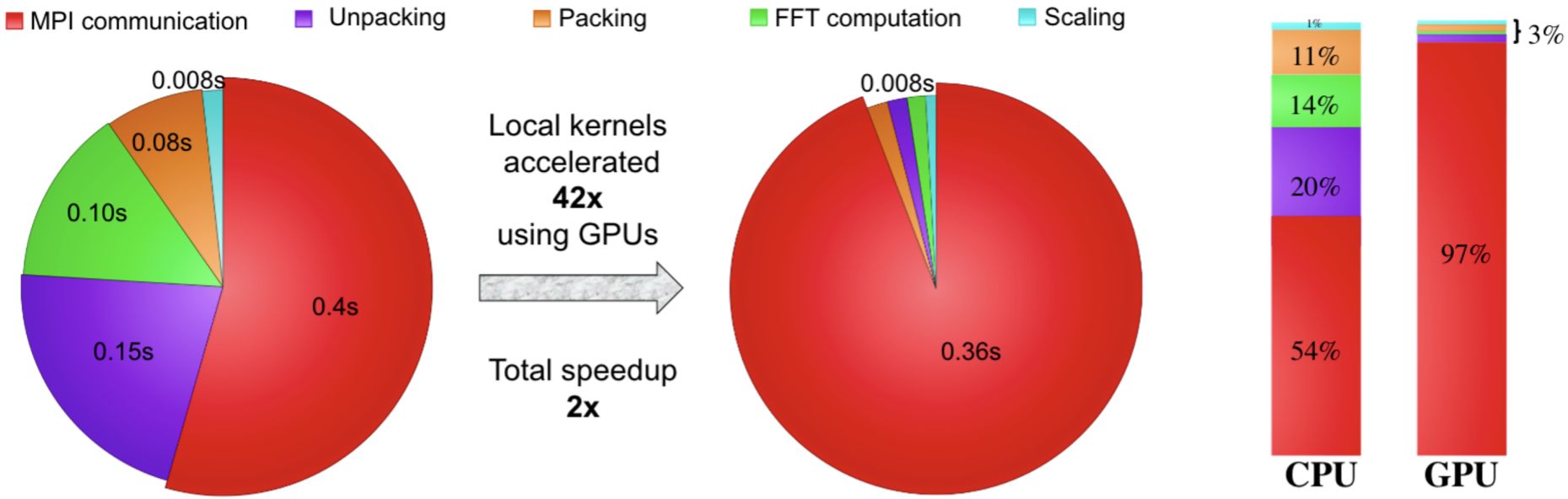


Figure: Performance comparison between fftMPI, using 1280 IBM Power9 Cores, 40 cores per node (left) and heFFTe, using 192 NVIDIA V100-GPUs, 6 per node (right).

Ref.: heFFTe: Highly Efficient FFT for Exascale. *Ayala et al., ICCS 2020.*

LAMMPS Rhodopsin Benchmark using heFFTe

- Molecular dynamics apps heavily rely on FFTs, and often have their own parallel FFT implementation (e.g., fftMPI, SWFFT).

LAMMPS Rhodopsin Benchmark using heFFTe

- Molecular dynamics apps heavily rely on FFTs, and often have their own parallel FFT implementation (e.g., fftMPI, SWFFT).
- Using heFFTe real-to-complex accelerates LAMMPS Kspace kernel around 1.76×.

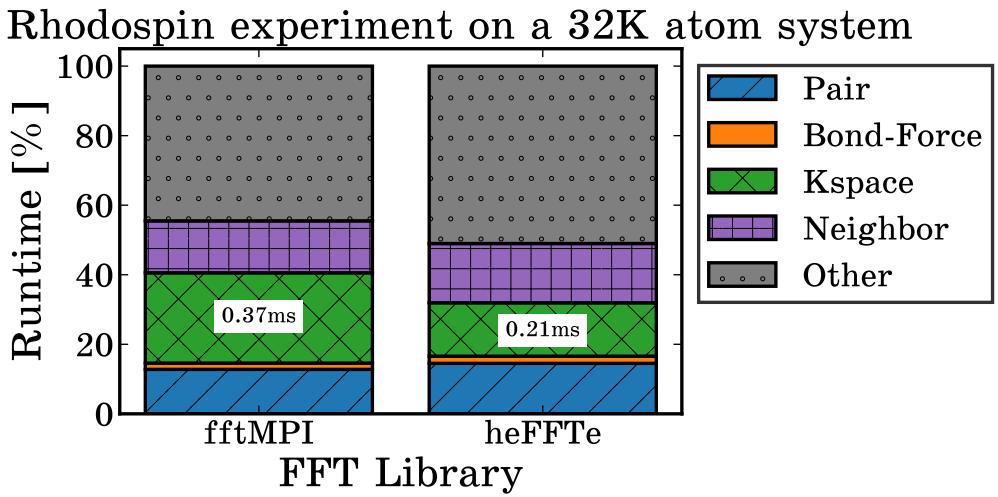


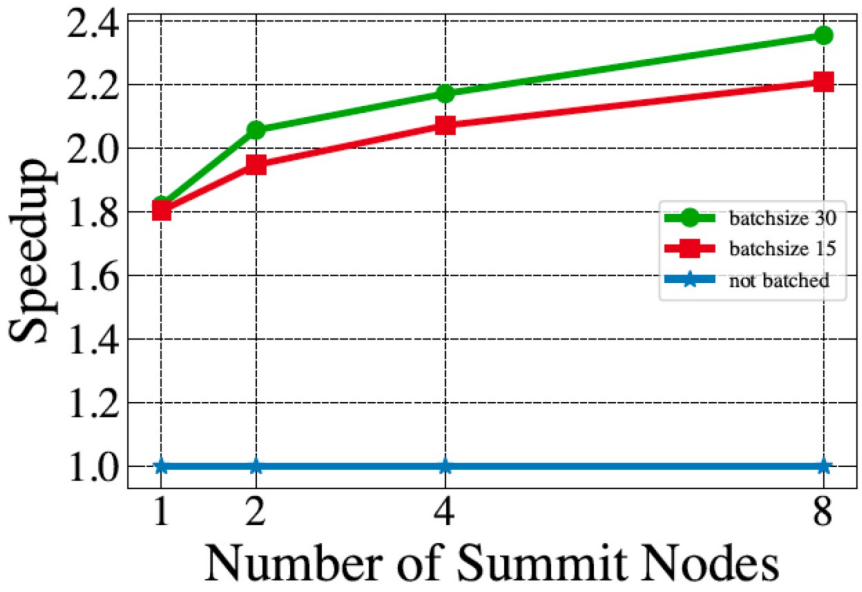
Figure: Breakdown for the LAMMPS Rhodopsin experiment. Using 32 Summit nodes, 6 V-100 GPUs per node, and 1 MPI per GPU.

Ref.: Performance Analysis of Parallel FFT on Large Multi-GPU Systems.

Ayala et al., IEEE IPDPS 2022.

3-D Batched FFT experiments

- Batched FFTs are needed in applications such image processing, filtering, particle energy computations, among others.



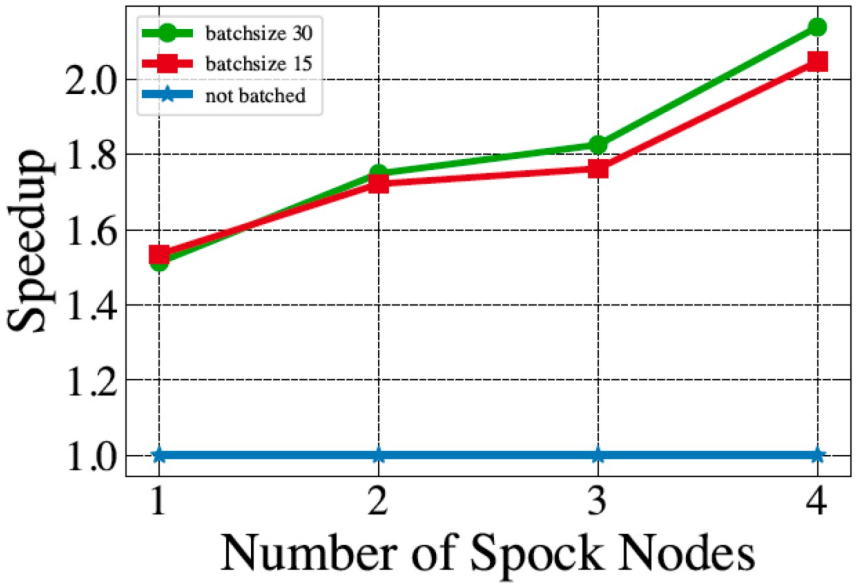
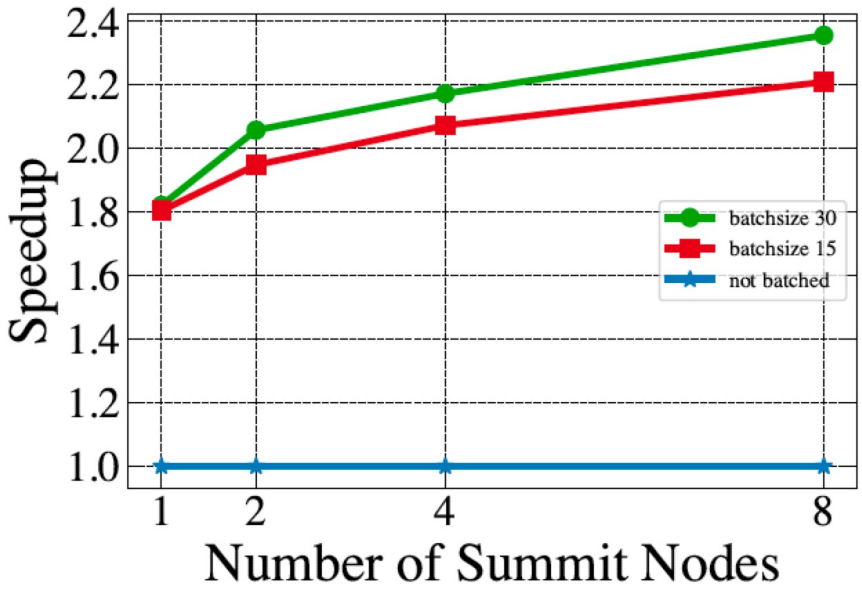


Figure: Batch of 3-D FFT of size 64^3 on NVIDIA and AMD GPUs, 1 MPI per GPU. Speedups of over 2× with respect to the not batched version.

3-D Batched FFT experiments

- Batched FFTs are needed in applications such image processing, filtering, particle energy computations, among others.



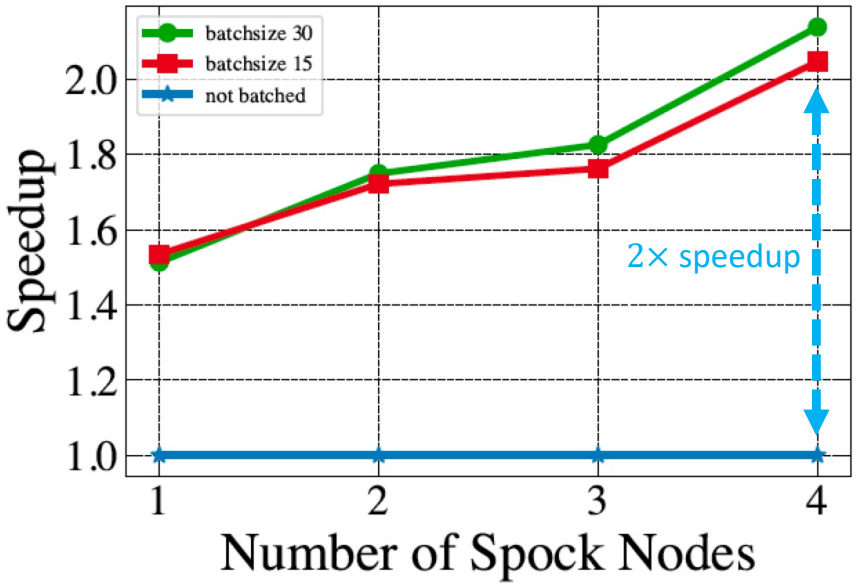
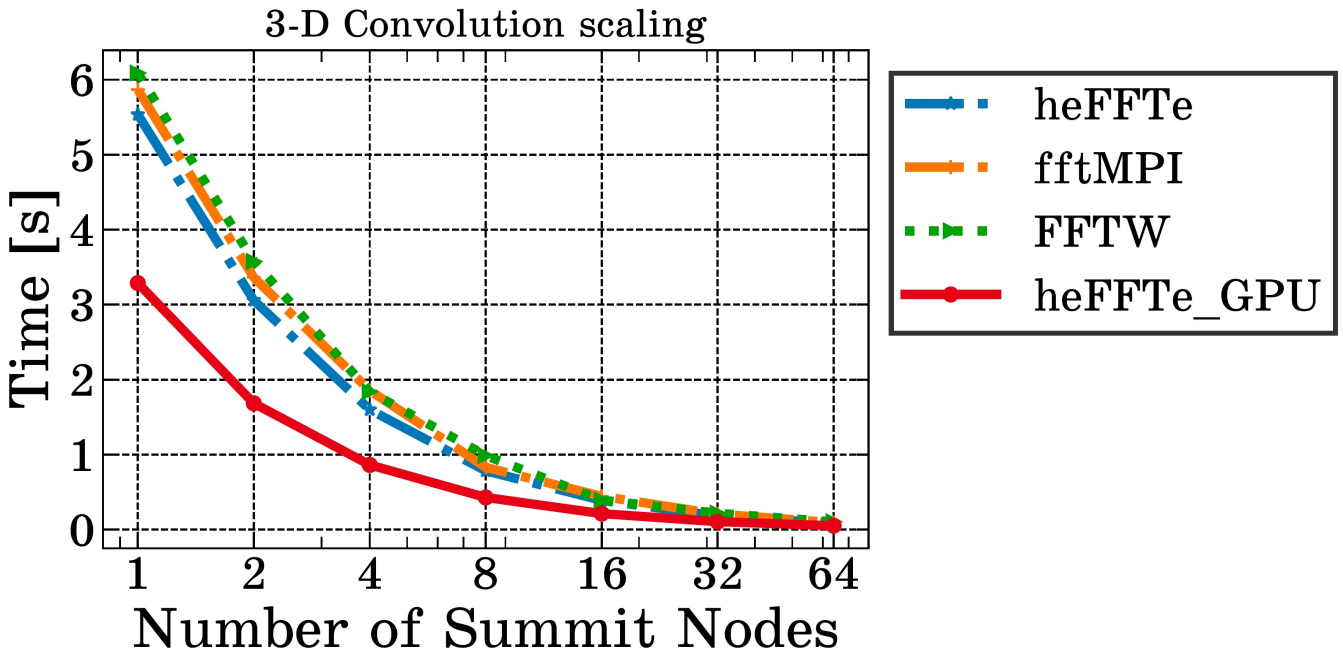
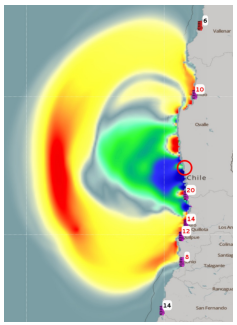


Figure: Batch of 3-D FFT of size 64^3 on NVIDIA and AMD GPUs, 1 MPI per GPU. Speedups of over 2× with respect to the not batched version.

3-D Convolution

- Convolutions are widely used for solving PDEs, e.g., in Earthquake simulation.
- They are also useful for convolutional networks.





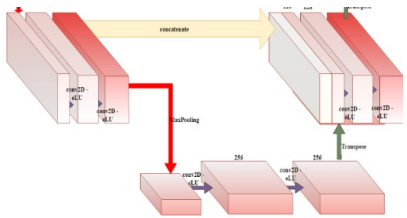


Figure: Convolution of a 3-D FFT of size 64^3, using 40 Power9 cores and 6 V-100 GPUs per node.

Enabling Mixed Precision in heFFTe

- Can be achieved using third party lossless and lossy compression libraries (NVOMP, ZFP).
- Currently, heFFTe uses casting compression + our own implementation of MPI_OSC_Alltoall

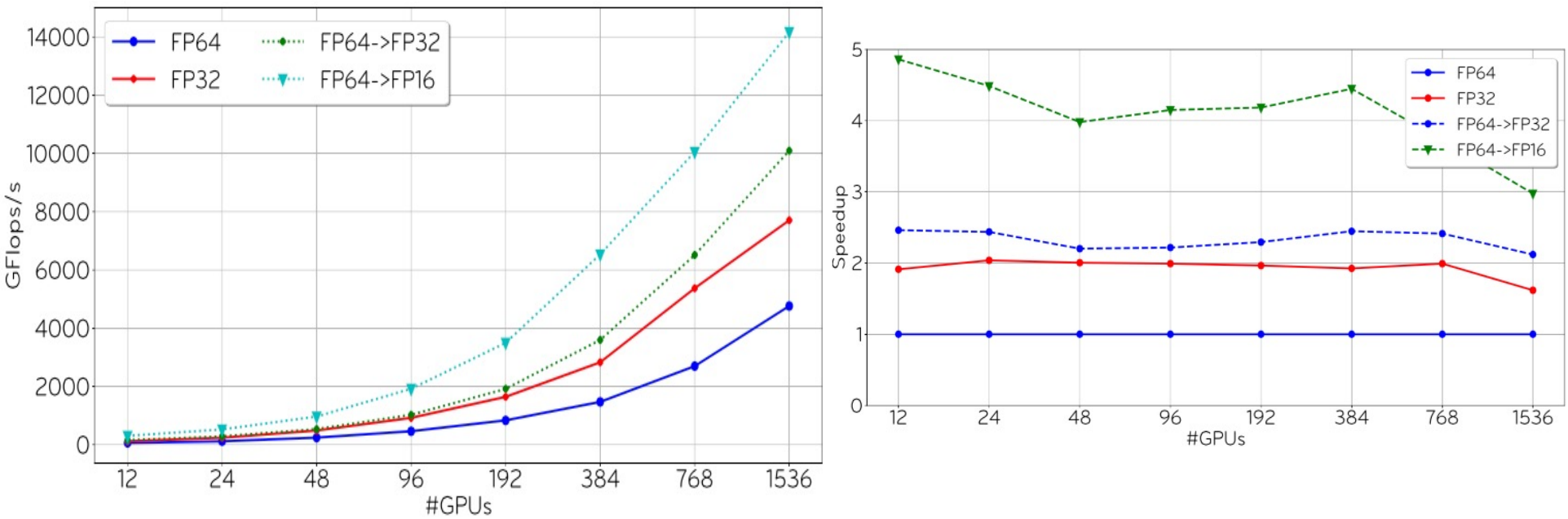
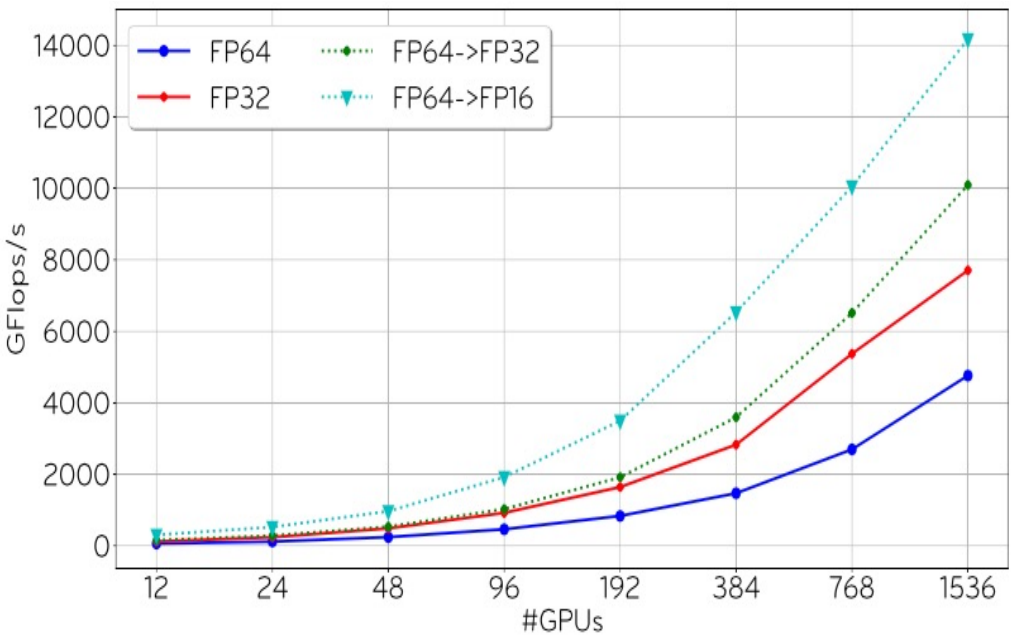


Figure: Speedup obtained reducing Floating Point (FP).

Ref.: Mixed precision and approximate 3D FFTs. *Cayrols et al., IEEE Cluster 2022.*

Enabling Mixed Precision in heFFTe

- Can be achieved using third party lossless and lossy compression libraries (NVOMP, ZFP).
- Currently, heFFTe uses casting compression + our own implementation of MPI_OSC_Alltoall



#GPU	FP64	FP32	FP64 → FP32
12	6.00e-15	4.96e-06	1.94e-07
24	6.17e-15	4.91e-06	2.20e-07
48	5.92e-15	4.49e-06	3.01e-07
96	6.00e-15	3.47e-06	3.90e-07
192	5.11e-15	3.54e-06	3.99e-07
384	5.25e-15	4.44e-06	5.09e-07
768	5.29e-15	3.13e-06	5.44e-07
1536	5.38e-15	3.06e-06	5.57e-07

Figure: Accuracy obtained reducing Floating Point (FP).

Ref.: Mixed precision and approximate 3D FFTs. *Cayrols et al., IEEE Cluster 2022.*

Scaling FFT on top Supercomputers

- Parallel FFT scales at the same rate of the underlying MPI framework, until a breakdown point.

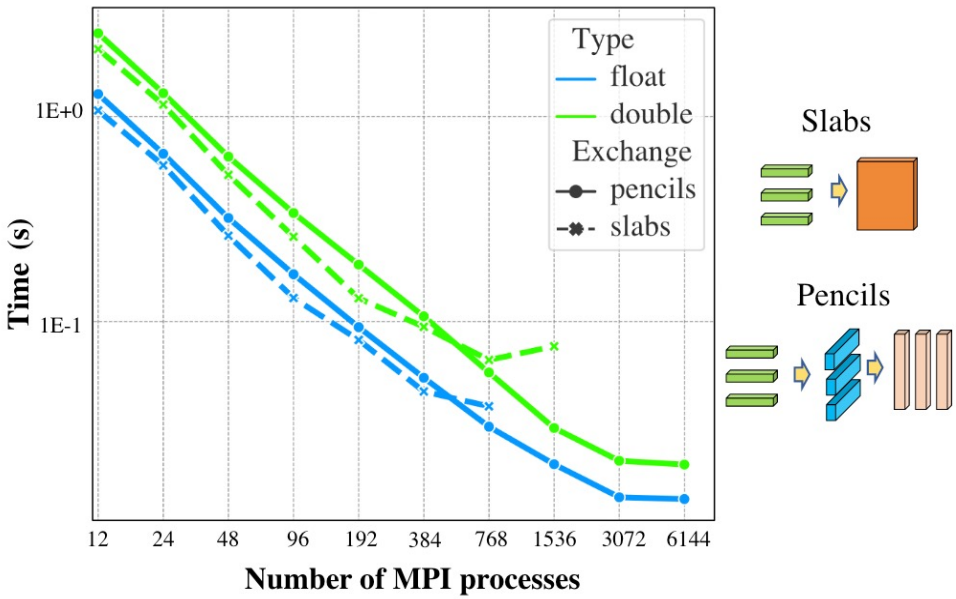
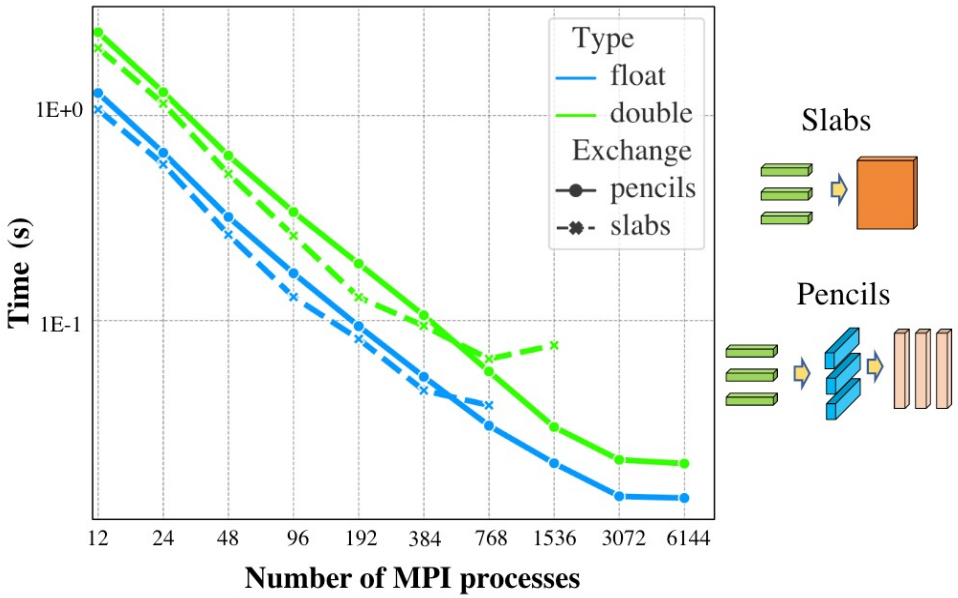


Figure: Strong Scalability on up to 6144 V-100 GPUs on Summit (left), and Weak Scalability on up to $\sim$ 1.2M Fugaku core, 48 cores per node (right). Using heFFTe.

Ref.: Scalability Issues in FFT Computation. *Ayala et al., ACM PACT 2021.*

Scaling FFT on top Supercomputers

- Parallel FFT scales at the same rate of the underlying MPI framework, until a breakdown point.



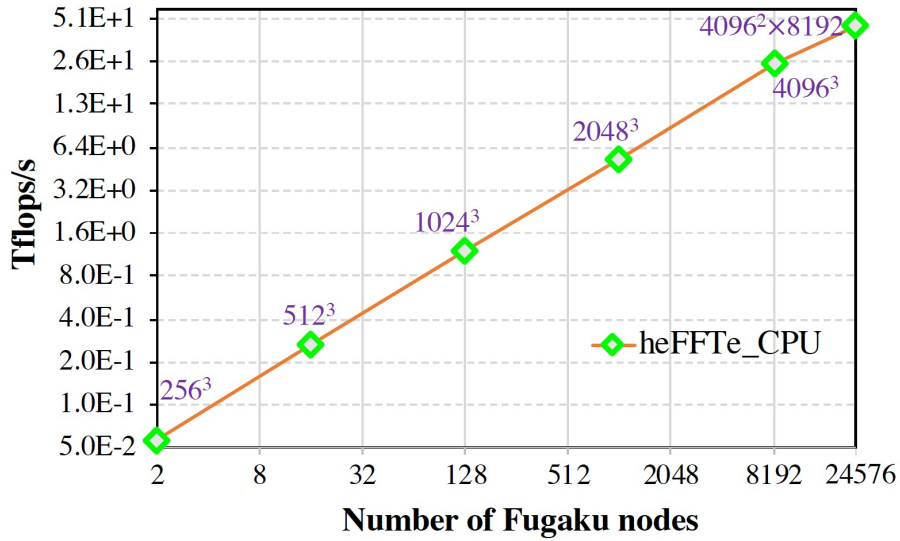


Figure: Strong Scalability on up to 6144 V-100 GPUs on Summit (left), and Weak Scalability on up to $\sim$ 1.2M Fugaku core, 48 cores per node (right). Using heFFTe.

Ref.: Scalability Issues in FFT Computation. *Ayala et al., ACM PACT 2021.*

Scaling FFT on top Supercomputers

- Similar behavior is observed for state-of-the-art FFT libraries.

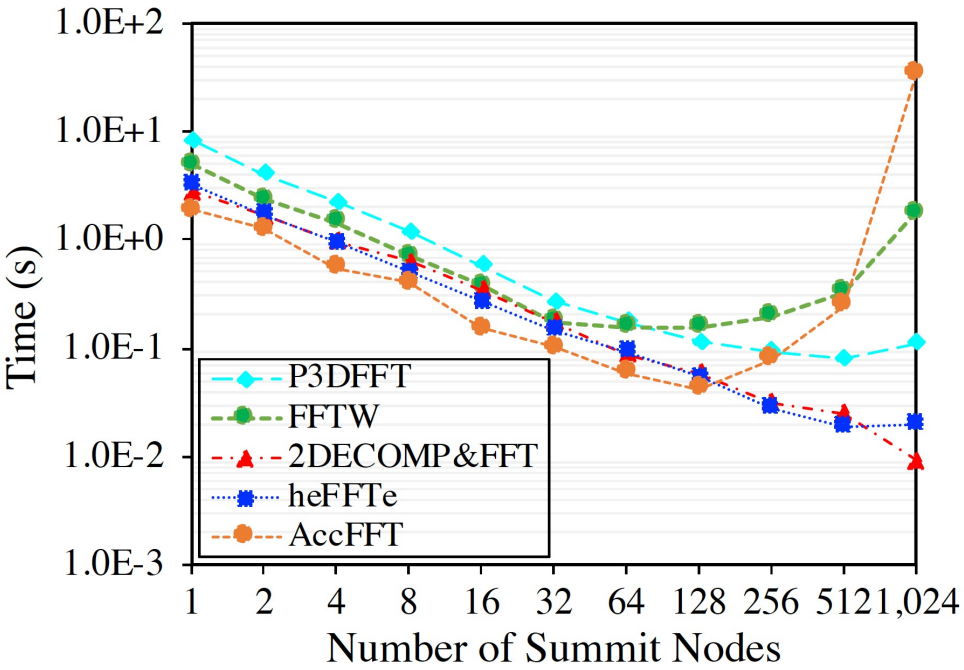


Figure: Strong Scalability on 32K Power9 cores for CPU-based libraries (left), and 4096 V-100 for GPU-based libraries (right).

Ref.: FFT Benchmark Performance Experiments on Systems Targeting Exascale.
Ayala et al., ICL Tech Report 2022.

Scaling FFT on top Supercomputers

- Similar behavior is observed for state-of-the-art FFT libraries.

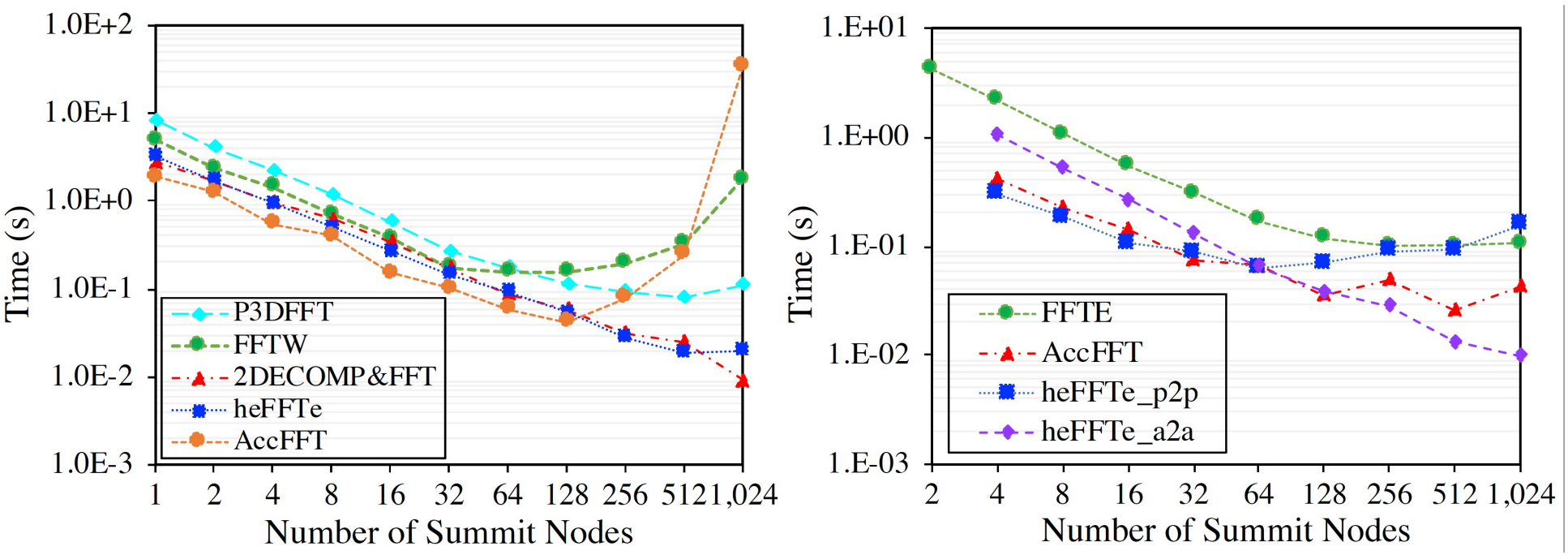


Figure: Strong Scalability on 32K Power9 cores for CPU-based libraries (left), and 4096 V-100 for GPU-based libraries (right).

Ref.: FFT Benchmark Performance Experiments on Systems Targeting Exascale.
Ayala et al., ICL Tech Report 2022.

3-D FFT with All-to-All Communication

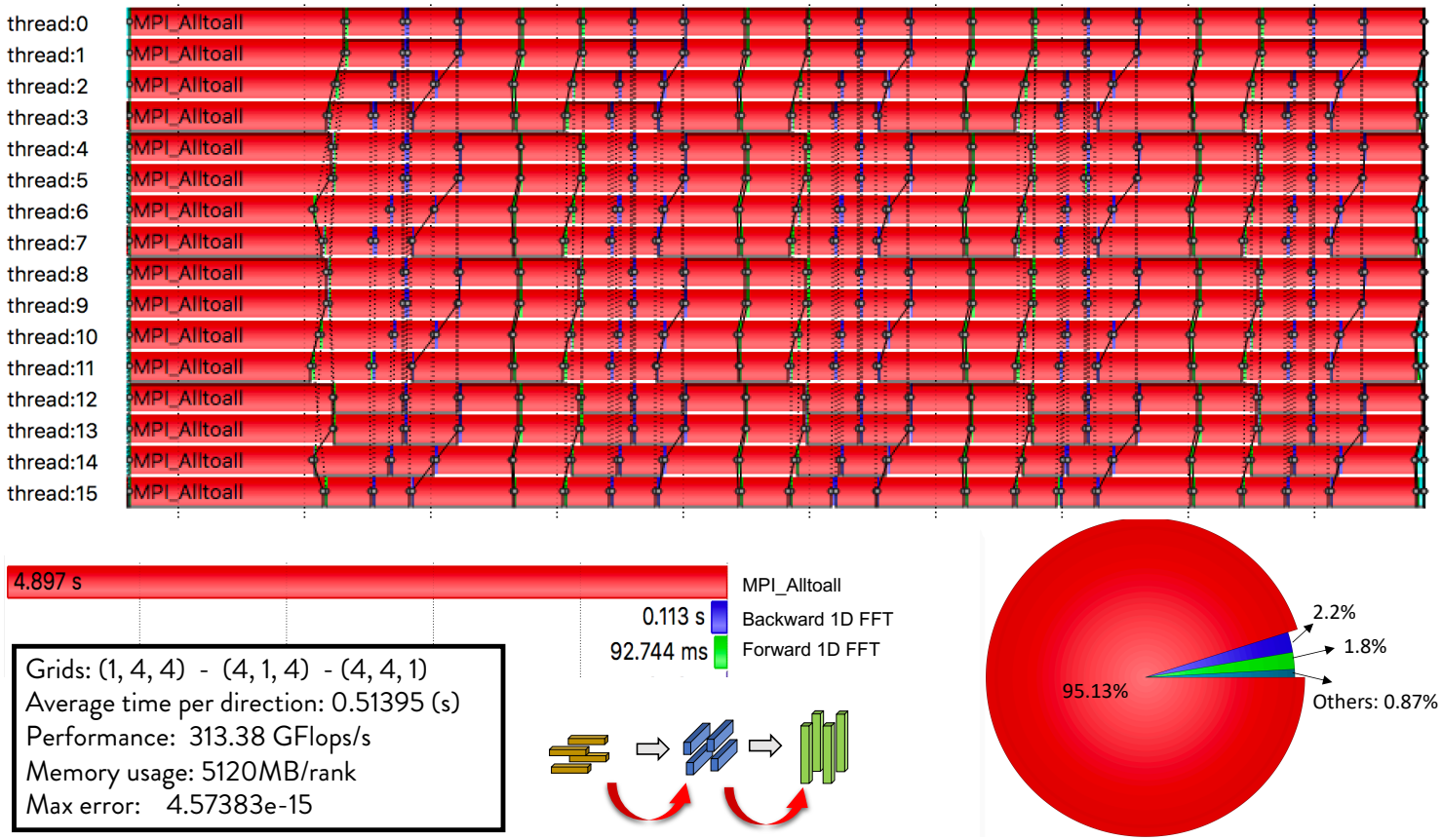


Figure: Vampir trace of back-to-back 3-D FFTs of size 1024^3 (5 forward + 5 backward), using 4 Summit nodes with 16 NVIDIA GPUs, 4 MPIs per node.

3-D FFT with Point-to-Point Communication

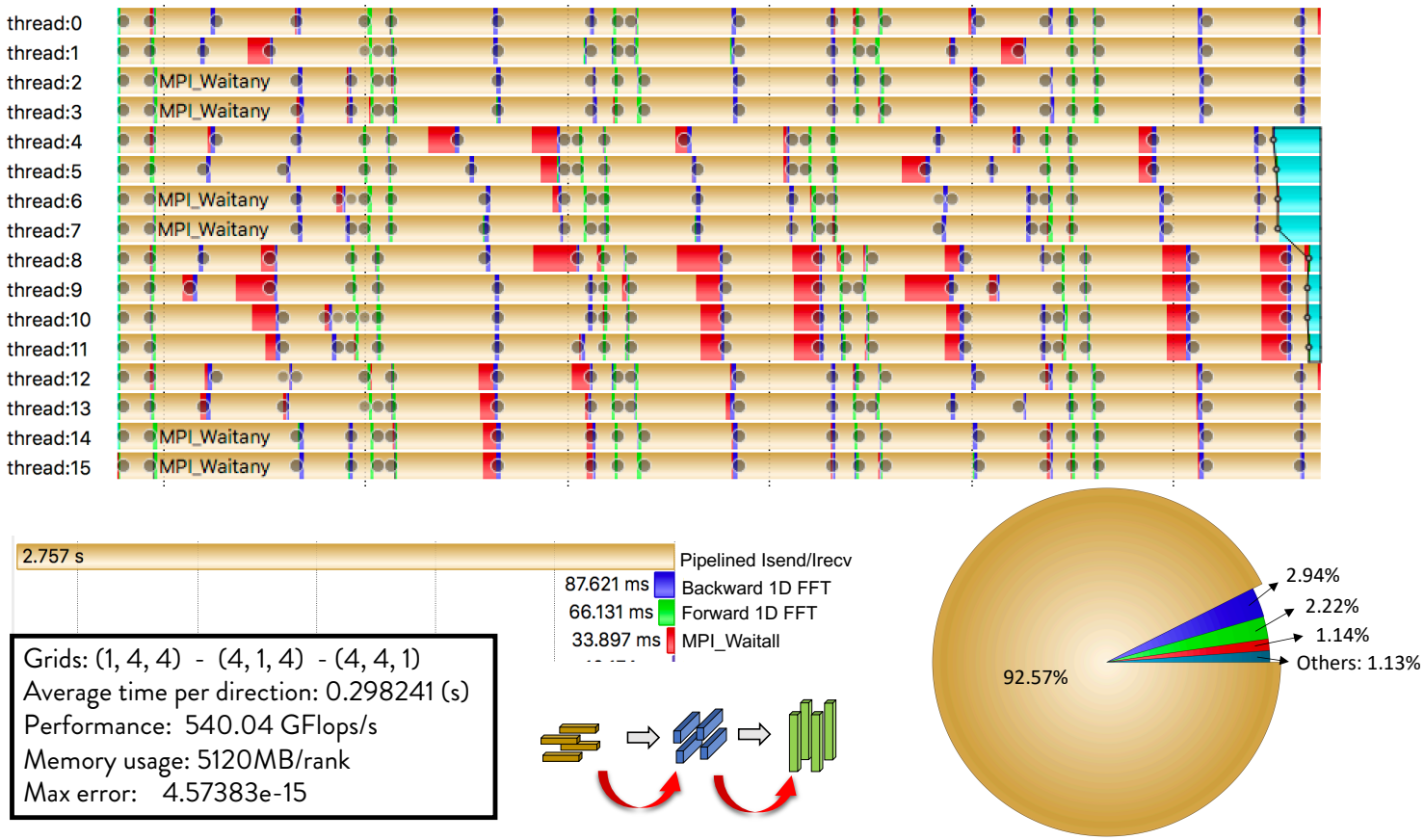


Figure: Vampir trace of back-to-back 3-D FFTs of size 1024^3 (5 forward + 5 backward), using 4 Summit nodes with 16 NVIDIA GPUs, 4 MPIs per node.

Scaling Communication

- While messages become smaller, latency effects become significant.
- Next figure shows variability of time spent on MPI_Alltoall.

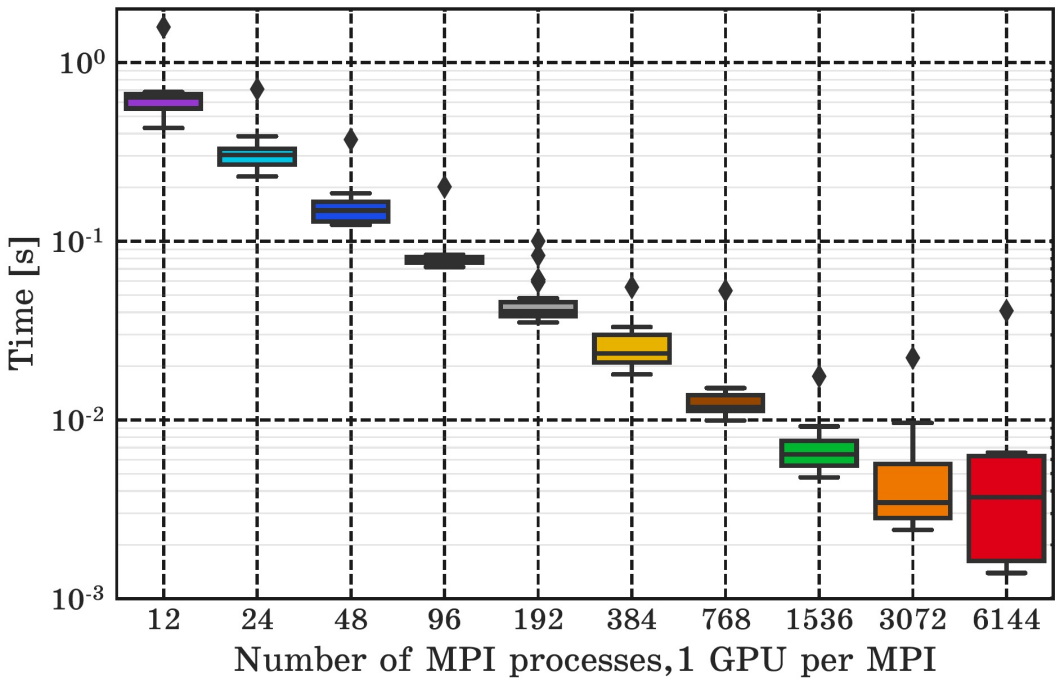


Figure: Strong Scalability on up to 6144 V-100 GPUs on Summit (left), and Weak Scalability on up to $\sim$ 1.2M Fugaku core, 48 cores per node (right). Using heFFTe.

Ref.: Accelerating MultProcess Communication for Parallel 3-D FFT. *Ayala et al., ExaMPI SC 2021.*

Tuning Processor grid

- At every transposition step a processor grid $P \times Q$ is defined
- The selection of this grid highly impacts scalability, in the next figure we set $P = 5$ and let Q vary.

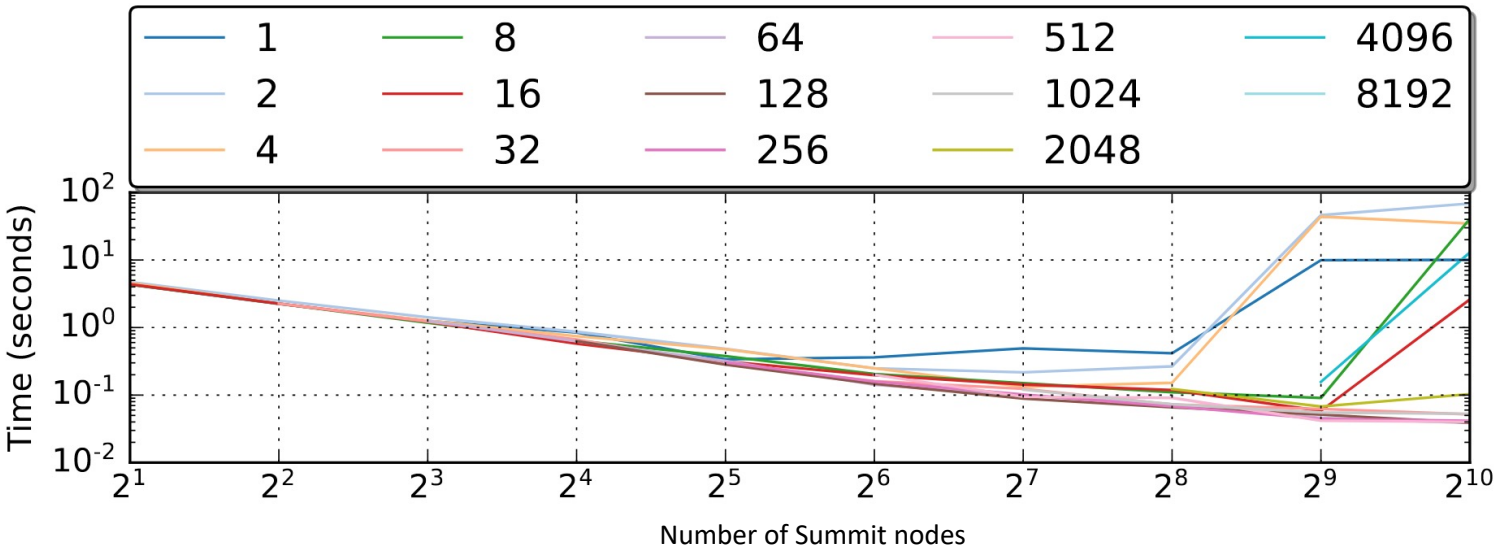


Figure: Strong Scalability on 40960 Power9 cores setting different grids $5 \times Q$. Using P3DFFT. *Auto-tuning also available with XTune.*

Ref.: Interim Report on Benchmarking FFT Libraries on High Performance Systems *Ayala et al., ICL Tech Report 2021.*

Tuning Processor grid

- At every transposition step a processor grid $P \times Q$ is defined
- This grid can be found using autotuning, e.g., using GPU-Tune

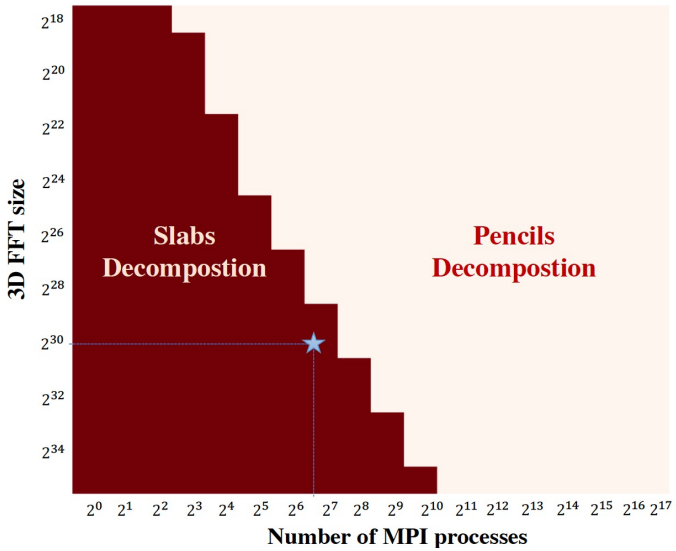
```
# Task parameters
dimx      = Integer     (64, 128, transform="normalize", name="dimx")
dimy      = Integer     (64, 128, transform="normalize", name="dimy")
dimz      = Integer     (64, 128, transform="normalize", name="dimz")

# Input/tuning parameters
#### note that the px_i, py_i, px_o, py_o are in log scale
px_i    = Integer     (0, np.log2(nodes*npernode), transform="normalize", name="px_i")
py_i    = Integer     (0, np.log2(nodes*npernode), transform="normalize", name="py_i")
px_o    = Integer     (0, np.log2(nodes*npernode), transform="normalize", name="px_o")
py_o    = Integer     (0, np.log2(nodes*npernode), transform="normalize", name="py_o")
comm_type    = Categoricalnorm (['a2a', 'p2p'], transform="onehot", name="comm_type")

# Tuning Objective
time    = Real        (float("-Inf") , float("Inf"), name="time")
```

Figure: GPU-Tune for autotuning heFFTe processor grids.

Ref.: Autotuning heFFTe with GPU-Tune. *Sherry Li's Group at LBNL.*

Tuning Algorithm

- Choosing between Pencil or Slab can lead to considerable speedups, $\sim 30\%$.
- The choice of Binary or Collective MPI is important at large-scale.



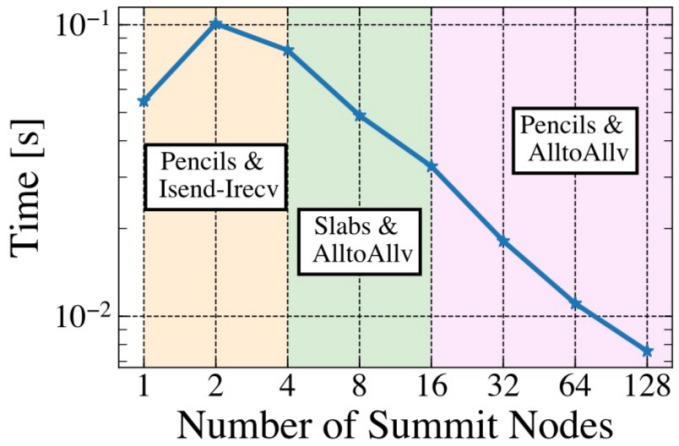


Figure: Phase Diagram (left) and best settings for a 512^3 FFT with 6 GPUs per node and manual tuning (right). Using heFFTe on Summit.

Ref.: Impacts of Multi-GPU MPI collective communications on large FFT computation *Ayala et al., ExaMPI SC 2019.*

Profiling Communication of Distributed FFTs

- We have used the following tools:
  - NVVP, rocProf
  - Score-P, Vampir, Cube
- Available ongoing developments include: OSU-INAM
- Challenges of the state-of-the-art:
  - Limited support for modern C++
  - Lack of support for all MPI distributions
  - Some do not support GPUs yet
  - If any, support for GPUs is vendor-specific
  - Hard to perform user-specific requests
  - In general, output does not reflect FFT workloads
  - Not mapping to the architecture and network
  - Developing a profiler tool requires lots of effort

Building a Simple and Unorthodox FFT profiler

- heFFTe tracing provides a detailed timing for tasks

Building a Simple and Unorthodox FFT profiler

- heFFTe tracing provides a detailed timing for tasks
- We need an adaptable visualization tool

Building a Simple and Unorthodox FFT profiler

- heFFTe tracing provides a detailed timing for tasks
- We need an adaptable visualization tool
- We can take advantage of the following analogies:
  - Network Traffic $\longleftrightarrow$ Vehicular Traffic
  - Memory unit $\longleftrightarrow$ Vehicle
  - Practical Bandwidth $\longleftrightarrow$ Average Velocity
  - Peak Bandwidth $\longleftrightarrow$ Road Capacity

Building a Simple and Unorthodox FFT profiler

- heFFTe tracing provides a detailed timing for tasks
- We need an adaptable visualization tool
- We can take advantage of the following analogies:
  - Network Traffic $\longleftrightarrow$ Vehicular Traffic
  - Memory unit $\longleftrightarrow$ Vehicle
  - Practical Bandwidth $\longleftrightarrow$ Average Velocity
  - Peak Bandwidth $\longleftrightarrow$ Road Capacity
- Using the power of heFFTe tracing + Python + TransCAD, we can build an architecture-aware visualization of network congestion.

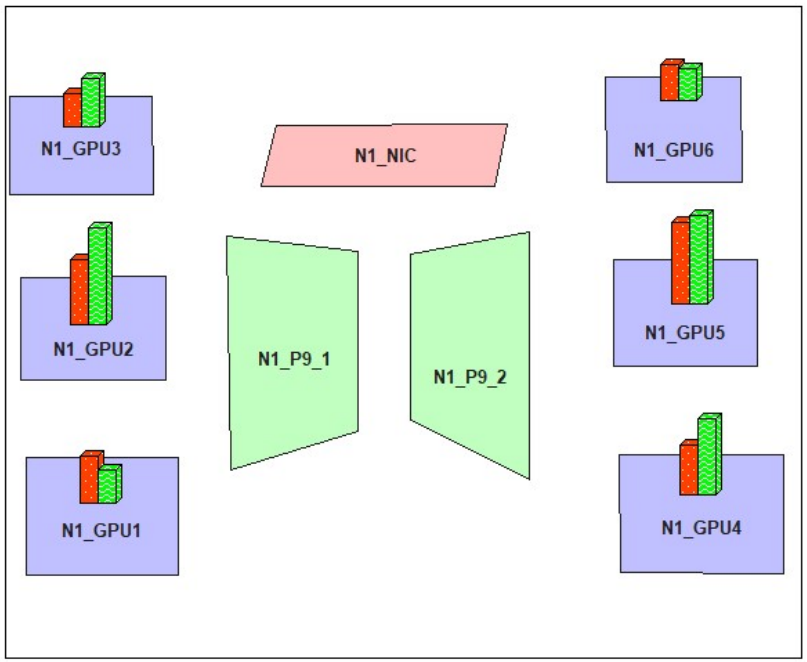
Building a Simple and Unorthodox FFT profiler

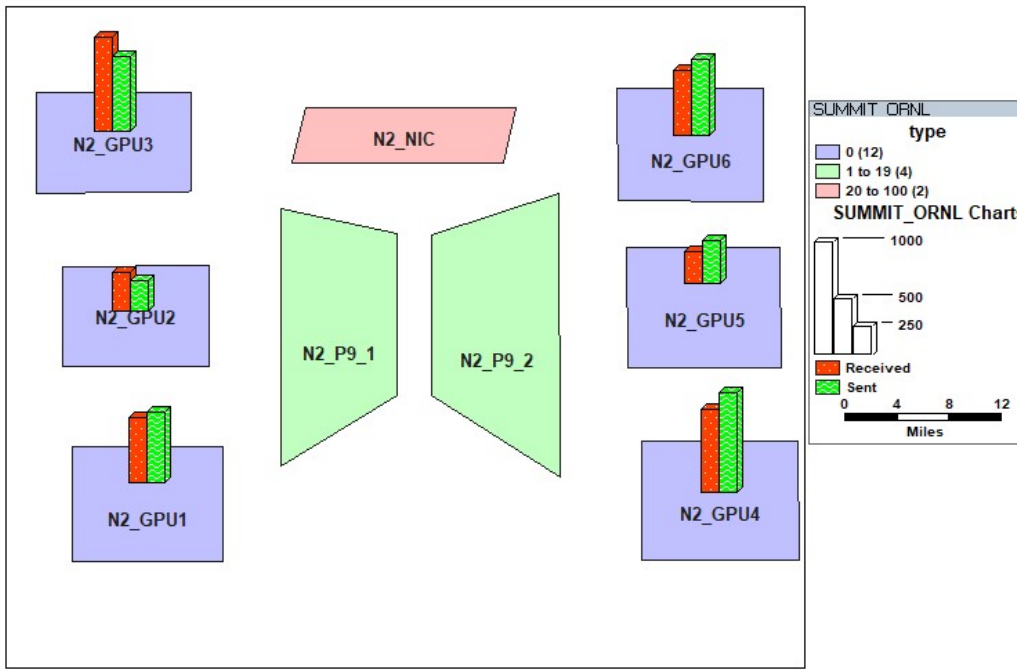
- heFFTe tracing provides a detailed timing for tasks
- We need an adaptable visualization tool
- We can take advantage of the following analogies:
  - Network Traffic $\longleftrightarrow$ Vehicular Traffic
  - Memory unit $\longleftrightarrow$ Vehicle
  - Practical Bandwidth $\longleftrightarrow$ Average Velocity
  - Peak Bandwidth $\longleftrightarrow$ Road Capacity
- Using the power of heFFTe tracing + Python + TransCAD, we can build an architecture-aware visualization of network congestion.
- Defining network peak values, TransCAD can also provide optimized paths

Ref.: heFFTe profiler *Ayala, 2022.*

Profiling Communication with heFFTe

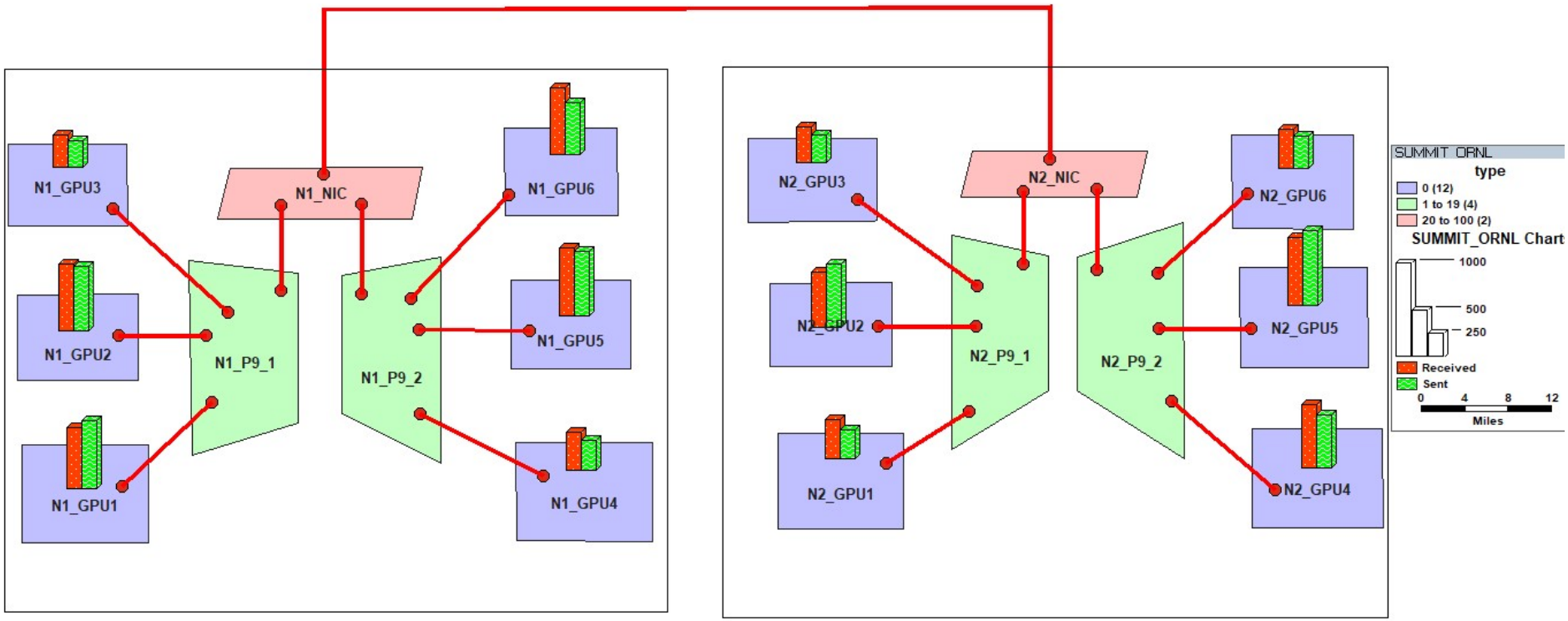
- We can obtain a video of data-volume exchange over time.





Profiling with heFFTe

- We can obtain a video of network traffic over time.



Acknowledgments

- heFFTe is funded by the Department of Energy (DoE) Exascale Project WBS 2.3.3.13.
- Collaborators:
  - A. Haidar (NVIDIA)
  - ICL OpenMPI Team (UTK)
  - ICL FIBER Team (UTK)
  - Network-Based Computing Research (DK. Panda's group, OSU)
  - ECP X-Tune (Sherry Li's group, LBNL)
  - D. Takahashi (U. Tsukuba)
  - D. Pekurovsky (SDSC)