heFFTe: Highly Efficient Exascale FFTs Library for Heterogeneous Architectures

Stan Tomov

Innovative Computing Laboratory
Department of Computer Science
University of Tennessee, Knoxville

Miroslav Stoyanov (ORNL), Alan Ayala (AMD), Azzam Haidar (NVIDIA), and Jack Dongarra
Sebastien Cayrols (UTK), Jiali Li (UTK), George Bosilca (UTK),
Veronica Montanaro (ETH), Sonali Mayani (ETH), Andreas Adelmann (ETH),
students and outside collaborators

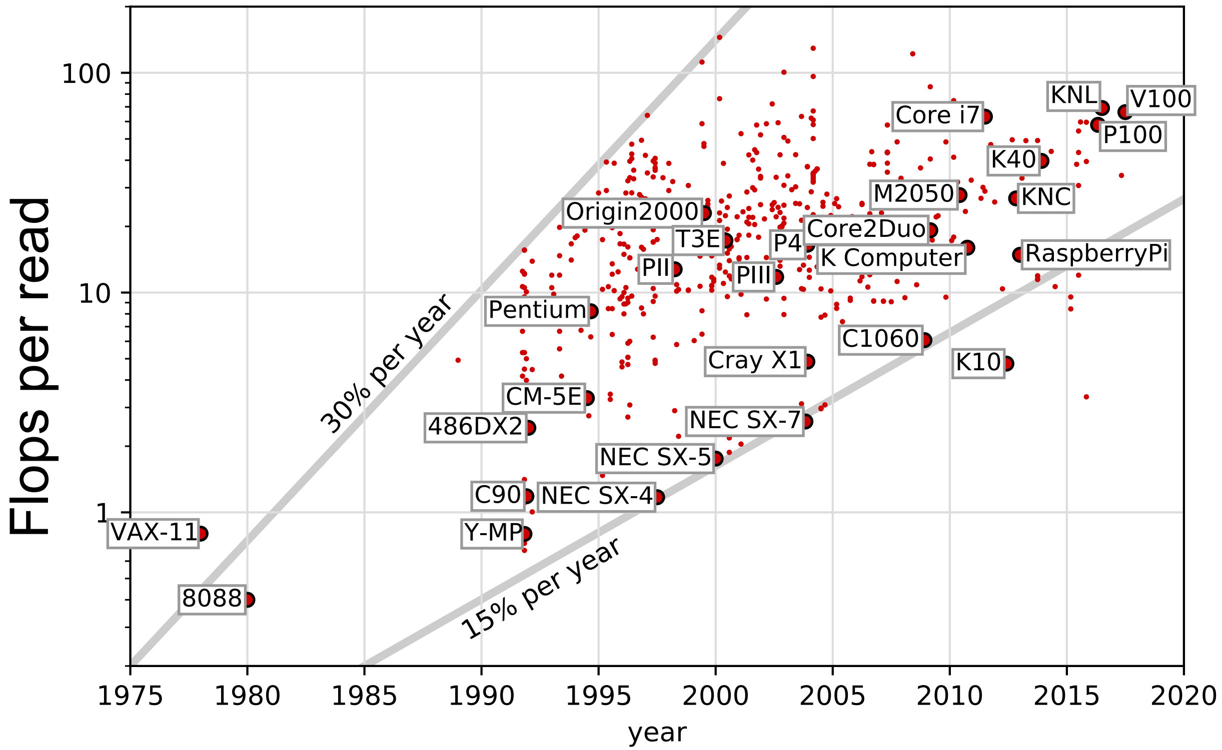
MUG 2023
Columbus, OH
August 23, 2023

ICL INNOVATIVE COMPUTING LABORATORY

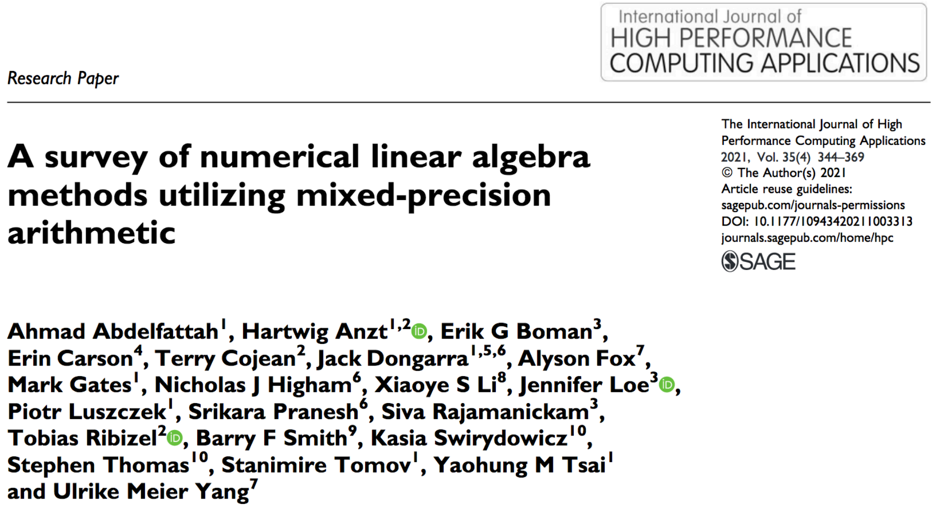
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

Hardware evolution motivates software redesigns

Hardware trends
compute vs. bandwidth peak



Source:


International Journal of
HIGH PERFORMANCE
COMPUTING APPLICATIONS

Research Paper

The International Journal of High Performance Computing Applications
2021, Vol. 35(4) 344–369
© The Author(s) 2021
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/10943420211003313
journals.sagepub.com/home/hpc
SAGE

A survey of numerical linear algebra methods utilizing mixed-precision arithmetic

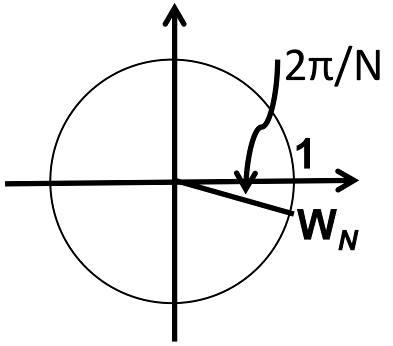
Ahmad Abdelfattah[1], Hartwig Anzt[1,2], Erik G Boman[3], Erin Carson[4], Terry Cojean[2], Jack Dongarra[1,5,6], Alyson Fox[7], Mark Gates[1], Nicholas J Higham[6], Xiaoye S Li[8], Jennifer Loe[3], Piotr Luszczek[1], Srikara Pranesh[6], Siva Rajamanickam[3], Tobias Ribizel[2], Barry F Smith[9], Kasia Swirydowicz[10], Stephen Thomas[10], Stanimire Tomov[1], Yaohung M Tsai[1] and Ulrike Meier Yang[7]

- **Need highly parallel algorithms**
- **Need algorithms with increased data reuse**
  (or reduced communication)
  - Currently, need more than 100x reuse for algorithm to remain compute bound

Fast Fourier Transform (FFT)

- **FFT computes the Discrete Fourier Transform (DFT) of a series:**

  Let $x = x_0, \ldots, x_{N-1}$ are complex numbers. The DFT of x is the sequence $\mathbf{X} = \mathbf{X}_0, \ldots, \mathbf{X}_{N-1}$



$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi kn/N} \qquad k = 0, \ldots, N-1.$$

, i.e., $\mathbf{X} = F_N\, x$, where

$$F_N = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ w_N^{1.1} & w_N^{1.2} & \cdots & w_N^{1.(N-1)} \\ \cdots & & & \\ w_N^{(N-1).1} & w_N^{(N-1).2} & \cdots & w_N^{(N-1).(N-1)} \end{bmatrix}, \quad w_N = e^{-(2\pi i/N)}$$

$= \cos(2\pi/N) - i \sin(2\pi/N)$

is a primitive N^{th} root of unity

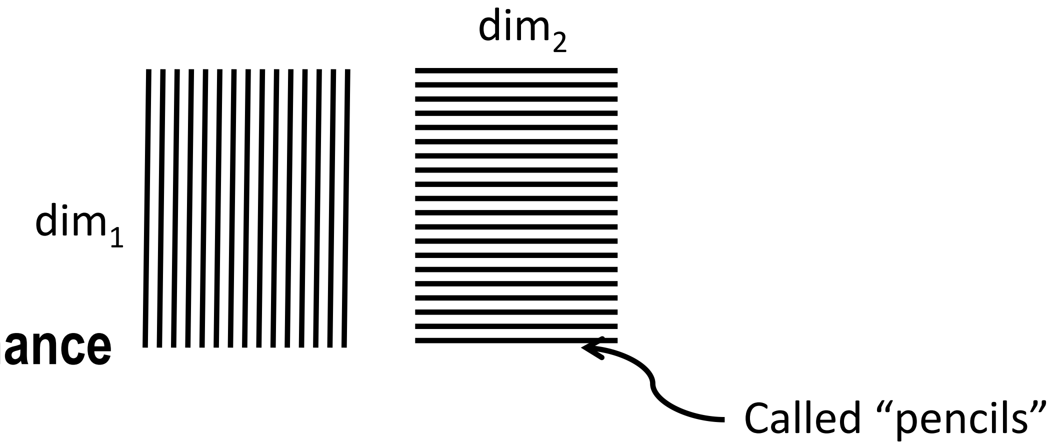
* **DFT can be computed as GEMV in $2N^2$ flops but FFT can do it in $5\, N \log_2 N$ flops!**

- The Inverse Discrete Fourier Transform (IDFT) is similarly defined except that the **e** exponents are taken as $i\, 2\pi\, k\, n\, /\, N$, and elements divided by N

Computing multidimensional FFTs with heFFTe

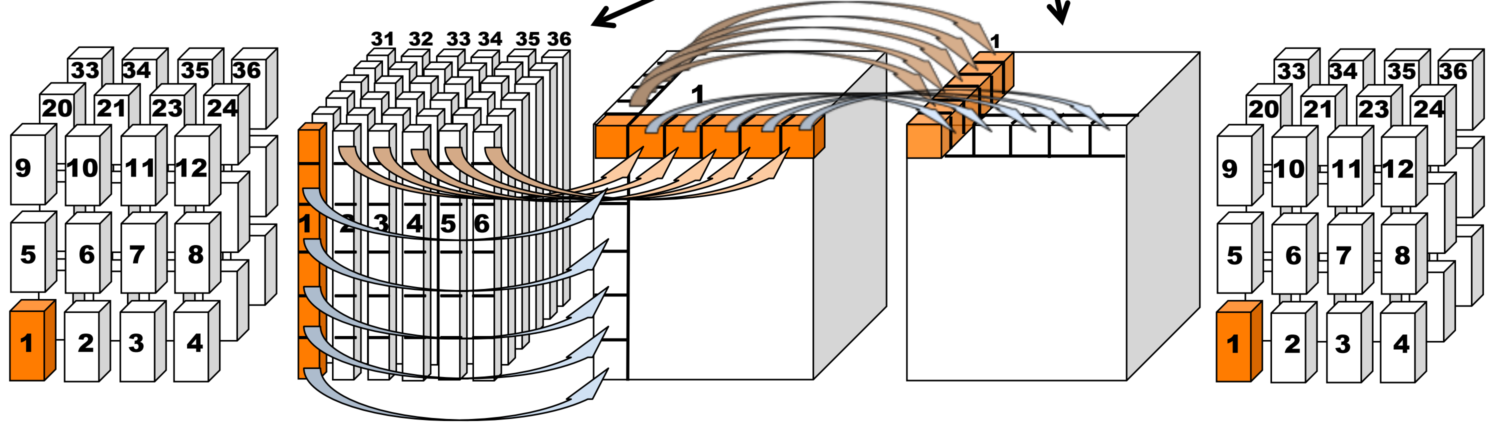
- **D-dimensional FFT algorithm**

```
for i in {dim_j , j = 1 .. D }
        Compute batch of ∏(dim != i) 1D FFTs of size dim_i
```



- **Order could be any, but particular order may impact performance**
- **Main building blocks**
  - **1D FFTs**
  - **Data reshuffles/transpositions**
  - **MPI communications**

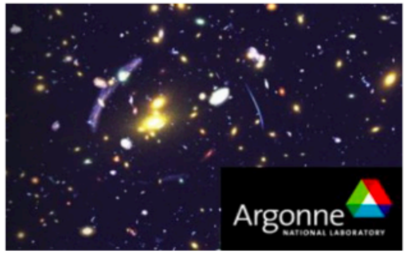
Steps in preparing "pencils" in the different dimensions and computing batches of 1D FFTs locally on each node/GPU



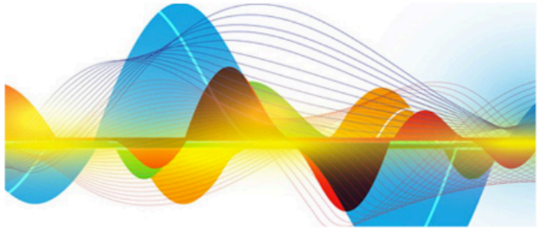
ICL INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

Applications Relying on Parallel FFTs

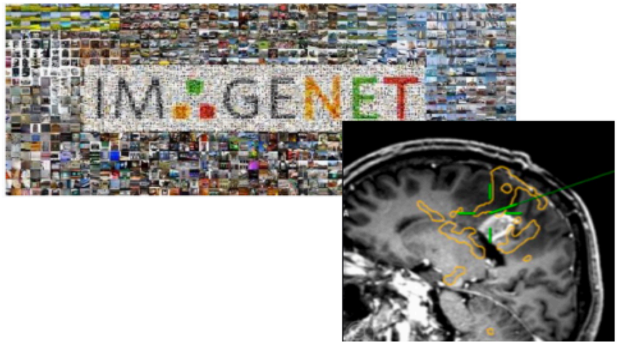
Cosmology
***ECP* ExaSky - HACC**

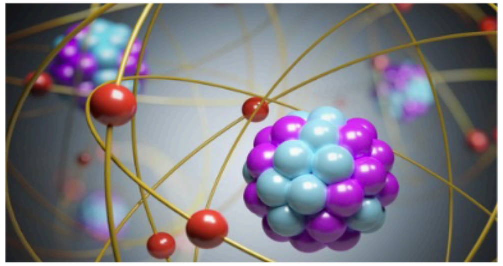


Signal processing,
***ECP* WARPX**

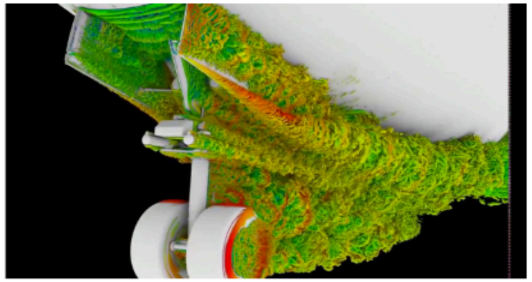


Deep Learning

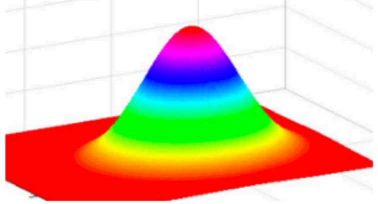




Molecular Dynamics
***ECP* EXAALT**



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



PDE solutions, **MASSIF**

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

ICL UT INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

Examples of FFT use

- **Spectral methods to solve PDEs**

$$\Delta u(x, y) = f(x, y),$$

where f is periodic in x and y, i.e., f(x + 2π, y) = f(x, y + 2π)

Take Fourier transform **F** on both sides, so

$$F \;\Delta u(x, y) = F\, f(x, y)$$

$$=> \; -(j^2+k^2)\,(F\,u)_{j,k} = (F\,f)_{j,k}$$

$$=> \; (F\,u)_{j,k} = -1/(j^2+k^2)\;(F\,f)_{j,k}$$

$$=> \; u = F^{-1}\,(-\,1/(j^2+k^2)\;.*\;F\,f\,)$$

Algorithm 2 Solve $-\nabla u + u = f$ in $\Omega = [0..2\pi]$ using FFTs.

Input : function f, smooth and periodic on the boundary
Output: solution u
1. Sample $f[i] = f(x_i)$ at N grid points $x_i = i * h$, $h = 2\pi/N$ and error tolerance e_{tol}
2. Compute g = FFT (f, e_{tol})
3. Scale g point-wise, $g[i] = g(i)/(1+(ih)^2)$
4. Compute u = IFFT(g, e_{tol})

Particle Simulations
CoPa – Cabana - Cajita

Cosmology
HACC

Molecular Dynamics **Exaalt**

Examples of FFT use

- Compression

```
>> A = imread( 'Fourier' , 'jpeg' );
>> imshow(A);
>> [nx,ny,nz] = size(A)
    512    417    3

>> FA = fft( A );
>> thresh=0.01*max(abs(FA(:))); ind=abs(FA)>thresh; cFA=FA.*ind;
>> count=nx*ny*nz-sum(ind(:)); percent = 100-count/(nx*ny*nz)*100
    percent = 8.59

>> Afilt = ifft( cFA );
>> imshow(uint8(Afilt));
```

Examples of FFT use

- Convolution

Convolutions f * g of images f and filers g can be accelerated through FFT, as shown by the following equality, consequence of the convolution theorem:

$$f * g = FFT^{-1} \left[FFT(f) .* FFT(g) \right],$$

where .* is the Hadamard (component-wise) product, following the '.*' Matlab notation

```
>> m = 100;           n = 50;
>> f = rand(m, 1);    g = rand(n, 1);

>> F = fft(f, m+n-1); G = fft(g, m+n-1);
>> norm( conv(f, g)  -  ifft( F .* G))
   ans =
   5.769457742102946e-14
```

heFFTe

Highly Efficient FFT for Exascale (heFFTe). Scalable, high-performance multidimensional FFTs; Flexible; User-friendly interfaces (C++/C/Fortran/python); Examples & benchmarks; Testing; Modified BSD license.

Capabilities:

- Multidimensional FFTs
- C2C, R2C, C2R
- DCS, DST, and convolution
- Batched FFTs
- Support flexible user data layouts
- Leverage and build on existing **FFT capabilities** through multiple backends

Pre-exascale environment:

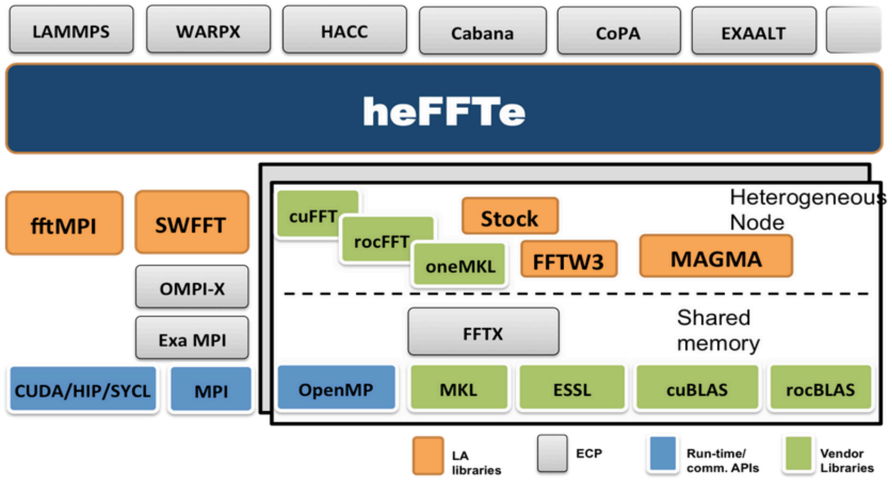
- **Summit @ OLCF (Nvidia GPUs)**
- **Crusher / Frontier (AMD GPUs), and others**
- **Florentia / Aurora (Intel GPU)**

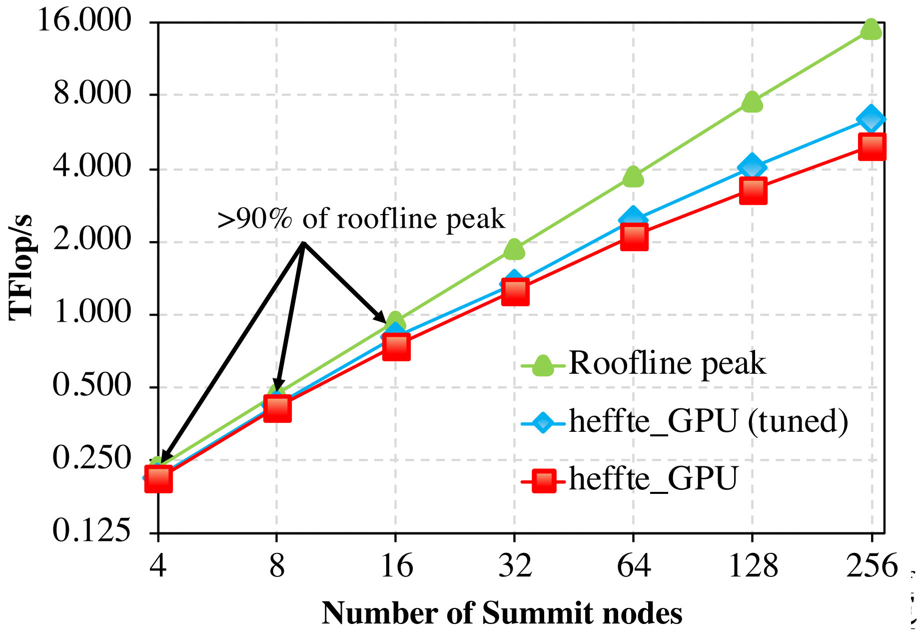
Current status:

- **heFFTe 2.3** with support for CPUs, Nvidia GPUs, AMD GPUs, and Intel GPUs
- Very good strong and weak scaling, reaching up to 90% of roofline peak

Open Source Software

- **spack** installation and integration in xSDK
- Homepage: http://icl.utk.edu/fft/
  Repository: **https://github.com/icl-utk-edu/heffte**





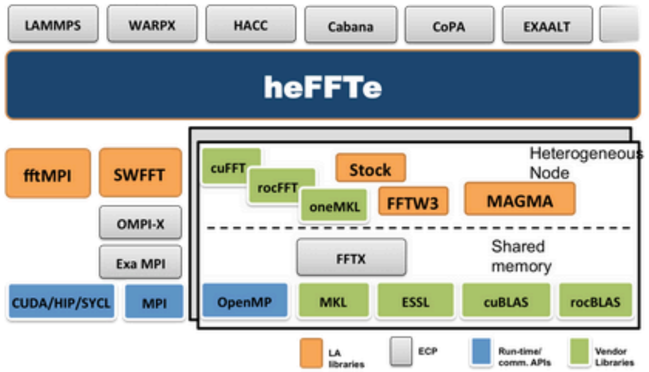
heFFTe

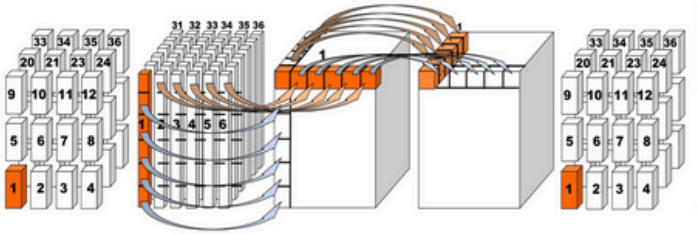
Highly Efficient FFT for Exascale

Stanimire Tomov (UTK)
Miroslav Stoyanov (ORNL)
Alan Ayala (AMD)
Azzam Haidar (NVIDIA)
Jack Dongarra (UTK)

THEN

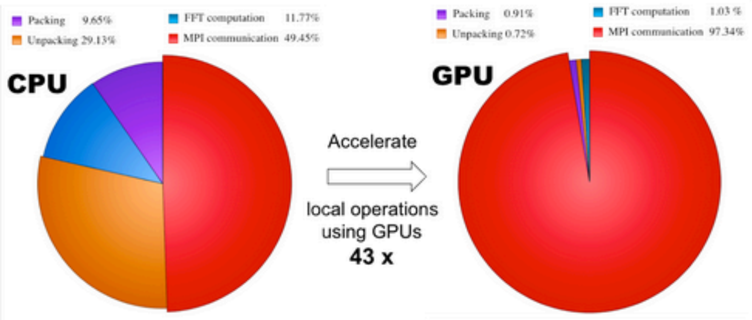
- The fast Fourier transform (FFT) is used in many domain applications - more than a dozen ECP applications use FFTs in their codes;
- State-of-the-art libraries like FFTW were no longer actively developed for emerging platforms;
- No GPU support for distributed multi-dimensional FFTs at the time;
- Some ECP application constructed their own FFTs directly in applications, e.g., fftMPI for LAMMPS and SWFFT for HACC;
- Features and application-specific needs were not supported uniformly;
- The goal was to leverage the existing FFT capabilities and build a sustainable FFT library for Exascale.

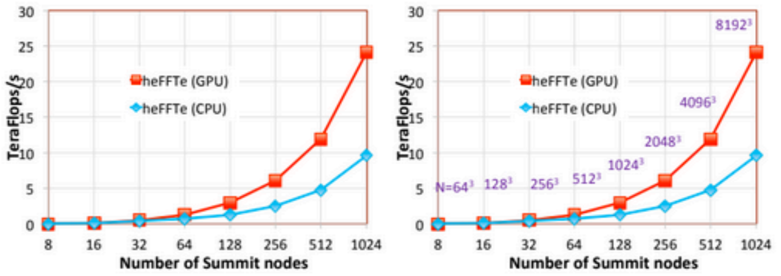




NOW

- GPUs (e.g., V100 on Summit) accelerate local FFT computations more than 40 x
- heFFTe supports multiple backends for Nvidia GPUs, AMD GPUs, Intel GPUs and multicore CPUs;
- Novel features such as Batched 2-D and 3-D FFTs
- Support FFT convolution, sine, and cosine transforms;
- Support for real and complex FFTs, multiple precisions and approximate FFT;
- Very good strong and weak scalability (Figure on right);
- FFT benchmark for MPI collectives and other FFT libraries.





heFFTe

Highly Efficient FFT for Exascale

Stanimire Tomov (UTK)
Miroslav Stoyanov (ORNL)
Alan Ayala (AMD)
Azzam Haidar (NVIDIA)
Jack Dongarra (UTK)

THEN

- **There were many FFT libraries but no GPU support for large-scale distributed systems**
- **HeFFTe did not exist and goal was to add GPU support while leveraging and extending existing capabilities**
  - **Added quickly support for NVIDIA GPUs to cover fftMPI and SWFFT functionalities**
  - **Still explored design choices on language, precisions, versions, how to add other architectures, how to leverage other FFTs, etc.**
  - **Decided to move from LAPACK/MAGMA software engineering and develop in C++ to easily handle data types, parameterizations, architectures, and configurable use of multiple FFT libraries**

NOW

- C++ library with backends for Nvidia GPUs, AMD GPUs, Intel GPUs, and multicore CPUs (with framework to easily add others, if needed)
- Backends are used not just for architectures but also for leveraging 3rd party FFT libraries (e.g., Stock, FFTW3, MKL, oneMKL, cuFFT, rocFFT)
- Support for multiple precisions, real and complex
- Support for many FFT-based functionalities

Experiences preparing for Aurora and Frontier

- Use of abstractions &standards (FFTs) helped with both functional & performance portability
- GPU kernel functional portability was helped by auto-generation tools
- xSDK policies helped the software engineering – heFFTe is xSDK compatible (regarding configuring, installing, testing, MPI usage, portability, contact and version information, open source licensing, namespacing, and repository access)
- Interactions and collaborations with diverse ST developers through xSDK and specialized xSDK PCRs were extremely helpful (xSDK mixed-precision techniques, batched sparse solvers, etc.)
- Interactions with vendors and early access to new and pre-released hardware helped
- To add efficient and sustainable support for many architectures, a large numerical library will inevitably need some auto-tuning capabilities; Libraries are parameterized but more may be needed

CURRENT DEVELOPMENTS

- Amongst the very few parallel FFT libraries that support GPUs, heFFTe provides unique functionalities that cover a large number of features from the state-of-the-art, making it ubiquitous for a wide range of applications

	Library	Pencil Decomp	Brick Decomp	Slab Decomp	Transpose Reshape	Stride Reshape	R2C Transform	Single precision	Mixed precision	Multiple backends	Nonblocking All-to-All
CPU	FFTW3	✓				✓	✓	✓			
CPU	FFTMPI	✓	✓		✓			✓		✓	
CPU	2DECOMP	✓				✓	✓				
CPU	SWFFT		✓		✓						
CPU	PFFT	✓			✓		✓				
CPU	P3DFFT	✓		✓	✓		✓	✓			✓
GPU	AccFFT	✓			✓	✓	✓	✓		✓	
GPU	FFTE	✓		✓	✓		✓	✓			
GPU	heFFTe	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

heFFTe backends

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.

heFFTe backends

Single-Device FFT Comparison

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

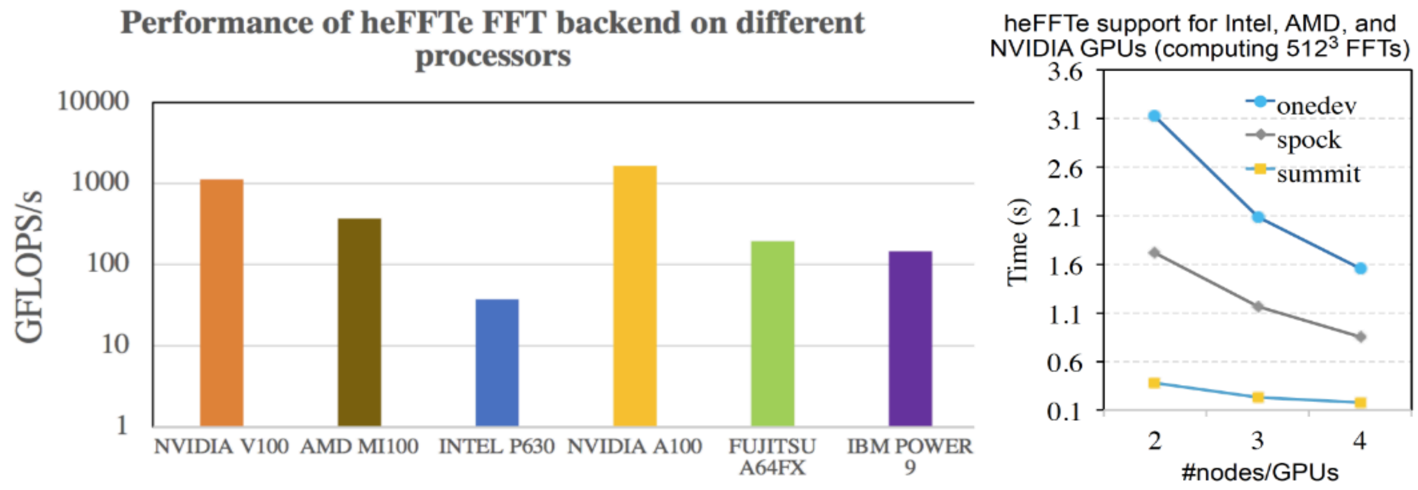


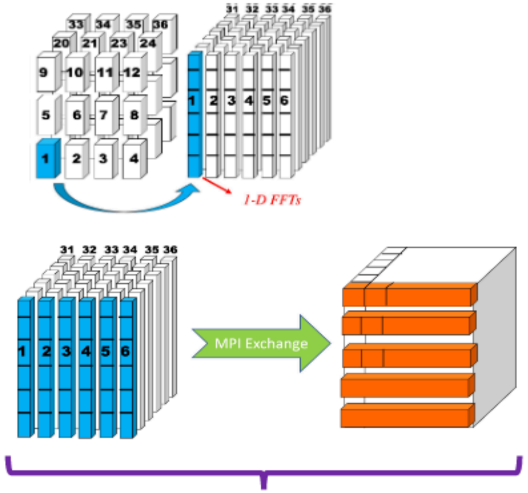
Figure: Comparison of single-device performance for a 512^3 FFT.

heFFTe implementation

Parallel FFT implementation

Algorithm 1 Parallel 3-D FFT computation on GPUs

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

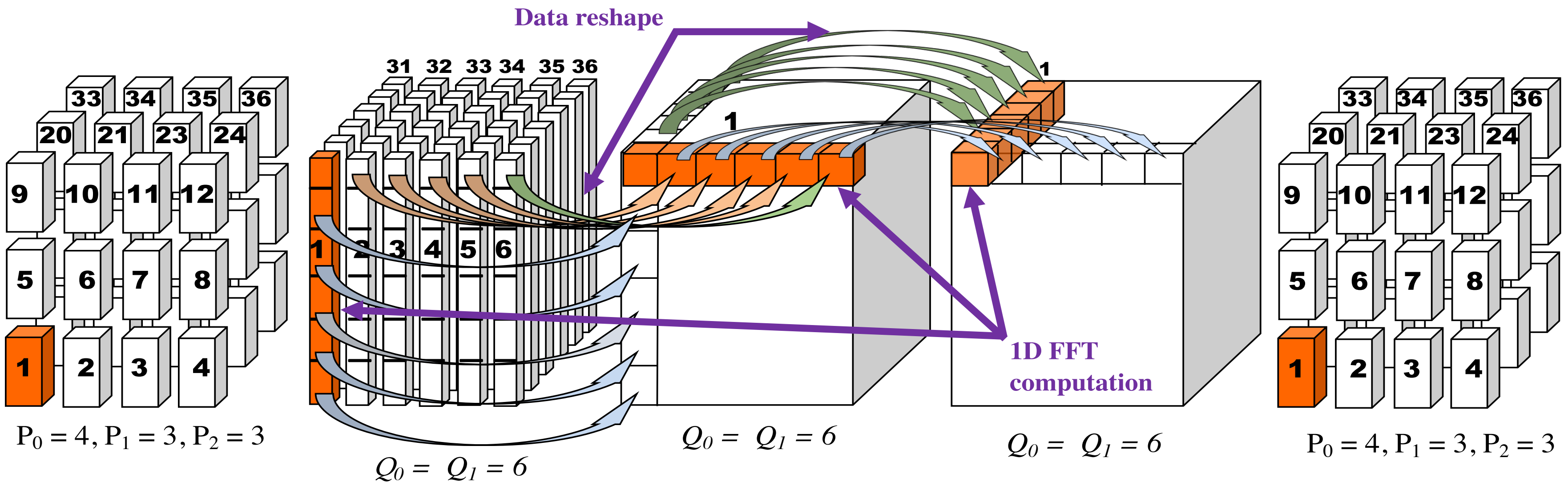


These 3 tasks can be replaced by 1 via MPI_Alltoallw

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

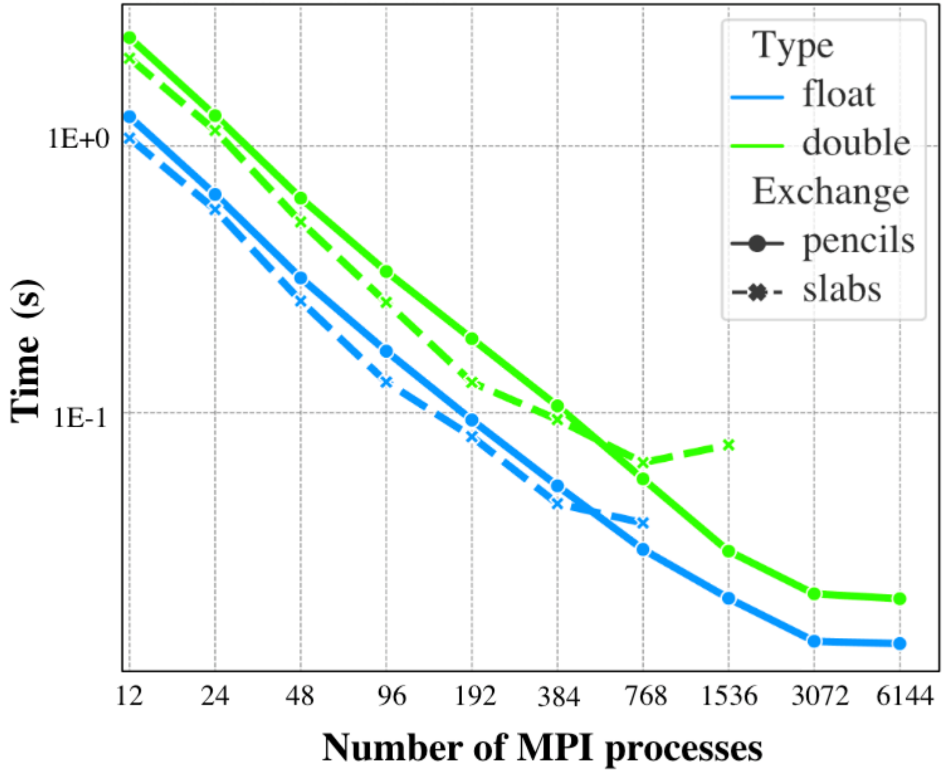
heFFTe Overview

- Support flexible user data layout input/output (pencils/cubes/slabs)



- 2-D and 3-D FFTs
  C2C, R2C, and C2R transformations
  DCS, DST, and convolution
  Batched FFTs
  CPU and GPUs (Nvidia, AMD, and Intel)
  Multi-precision FFTs

heFFTe Strong Scalability – Summit



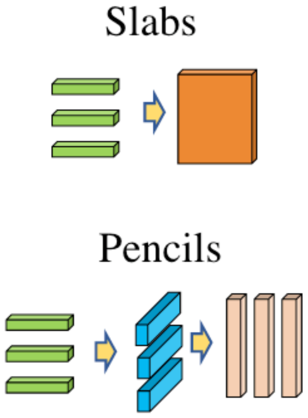
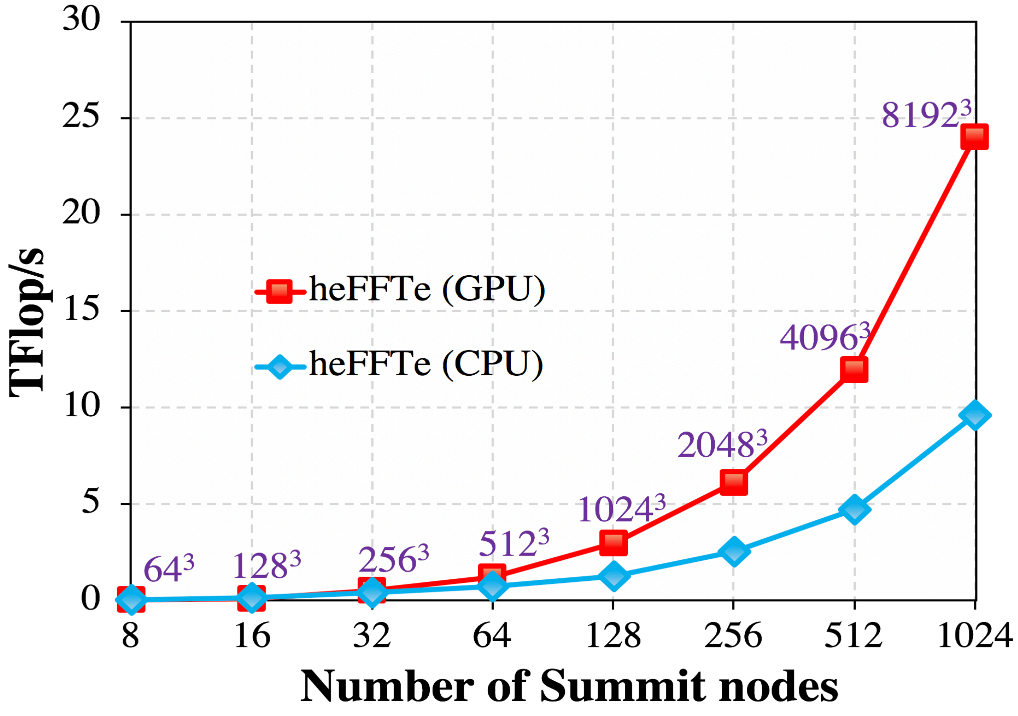
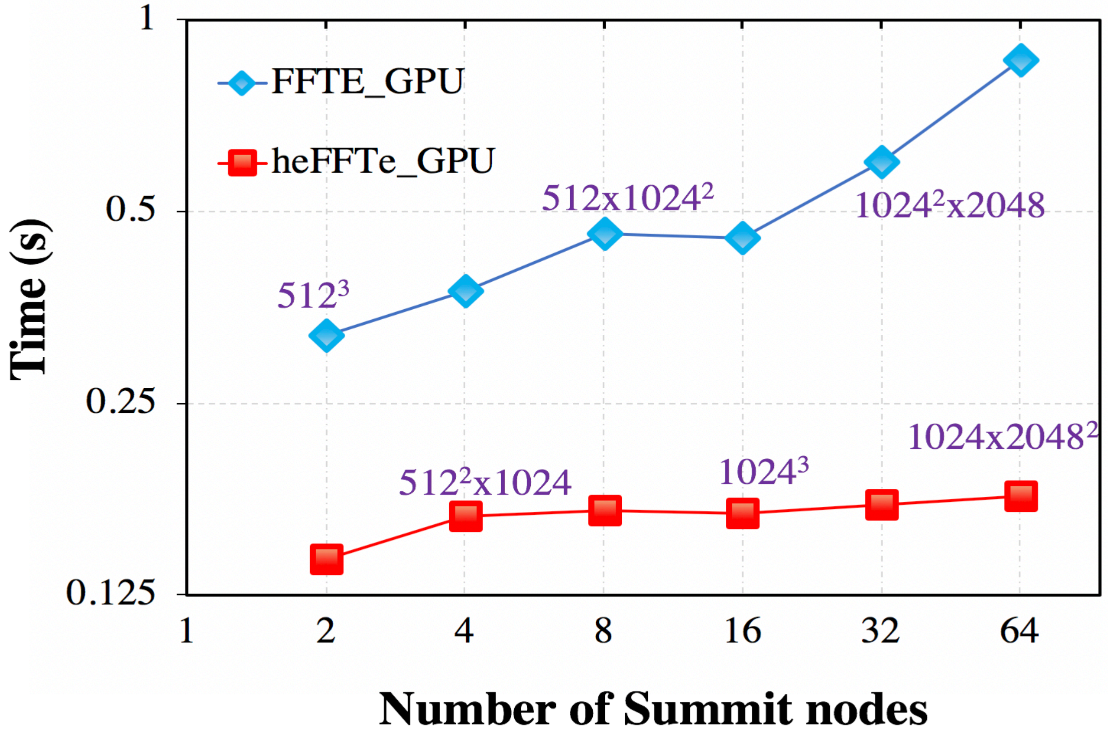


Fig. 6. Comparison of pencil and slab decompositions for strong scaling of a 3-D FFT of size 1024^3. Using *heFFTe* with cuFFT backend, 6 MPI processes (1 MPI processes per GPU-V100) per node, and single-precision complex data.

heFFTe Weak Scalability

- 2x speedup over state-of-the-art CPU libraries, FFTMPI, SWFFT
- 2x speedup over GPU library FFTE.

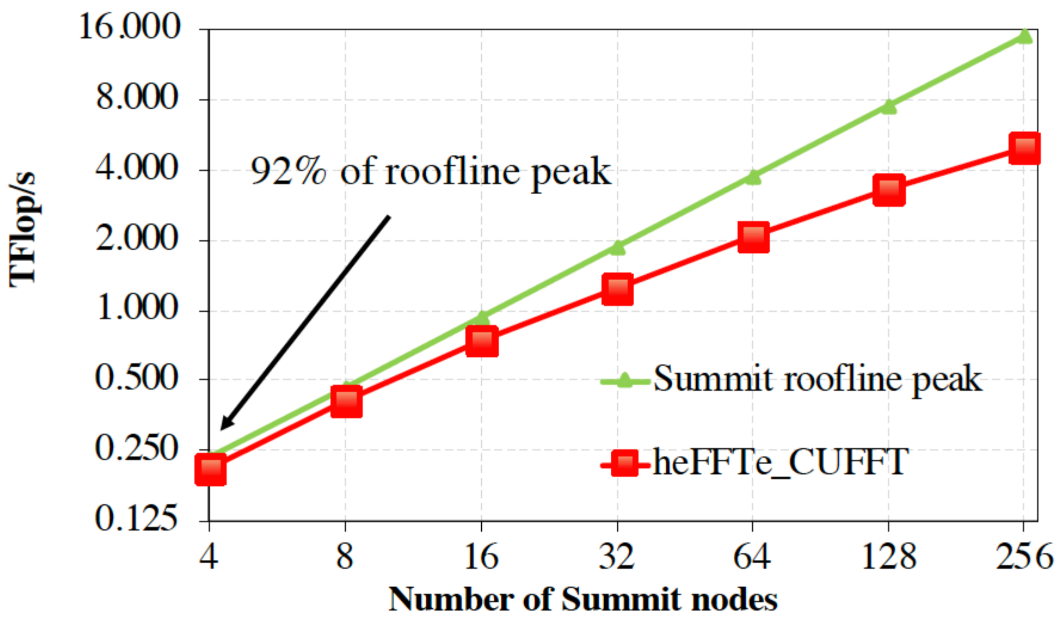


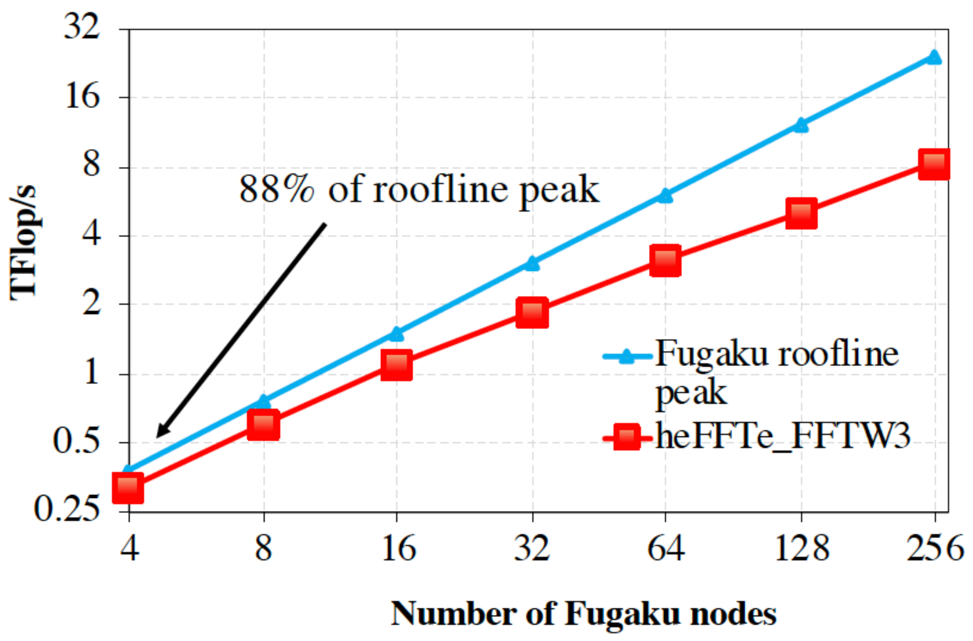


Forward and backward FFT on a Complex 3D array in double precision.

Using 6,144 NVIDA V100 GPUs (6/node) and 40,960 IBM Power 9 cores (40/node).

heFFTe Roofline analysis





Roof-line performance model – heFFTe performance on a 3-D FFT of size 1024^3 using 6 MPI/node, 1 GPU-Volta100 per MPI for Summit, and 48 A64FX per node on Fugaku.

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE
Department of Electrical Engineering and Computer Science

HPFFT benchmark (https://github.com/icl-utk-edu/hpfft)

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	-			✓

HPFFT benchmark (https://github.com/icl-utk-edu/hpfft)

Scaling FFT on top Supercomputers

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

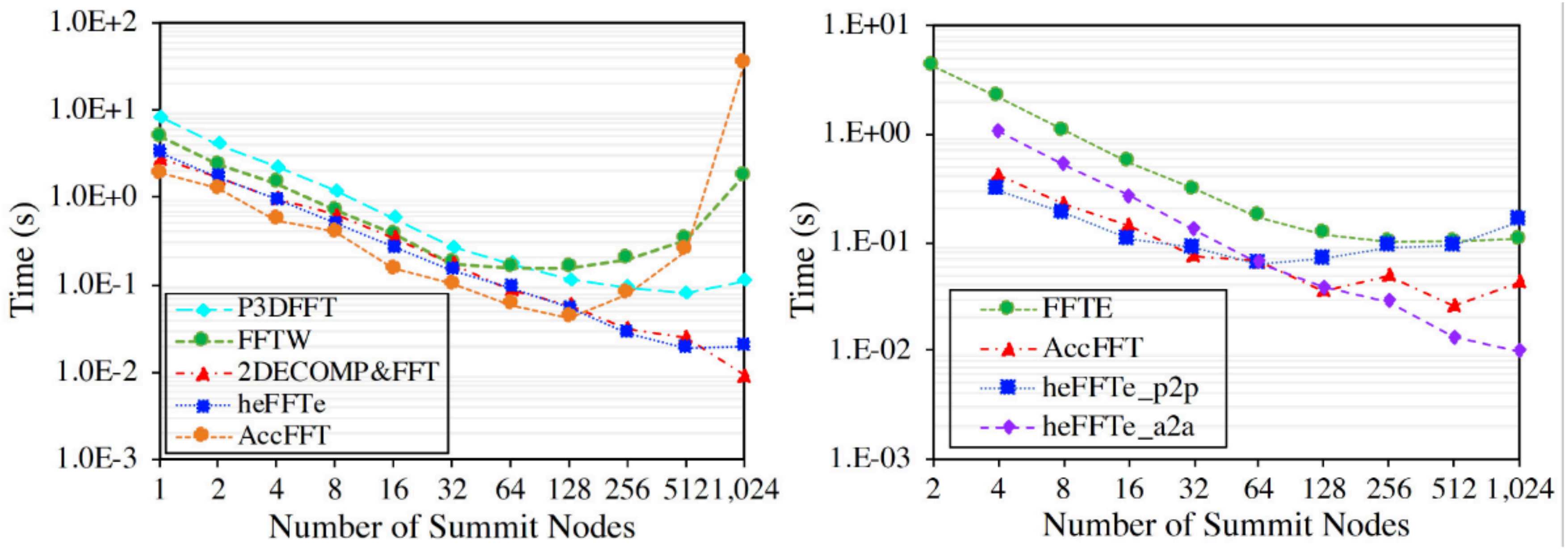
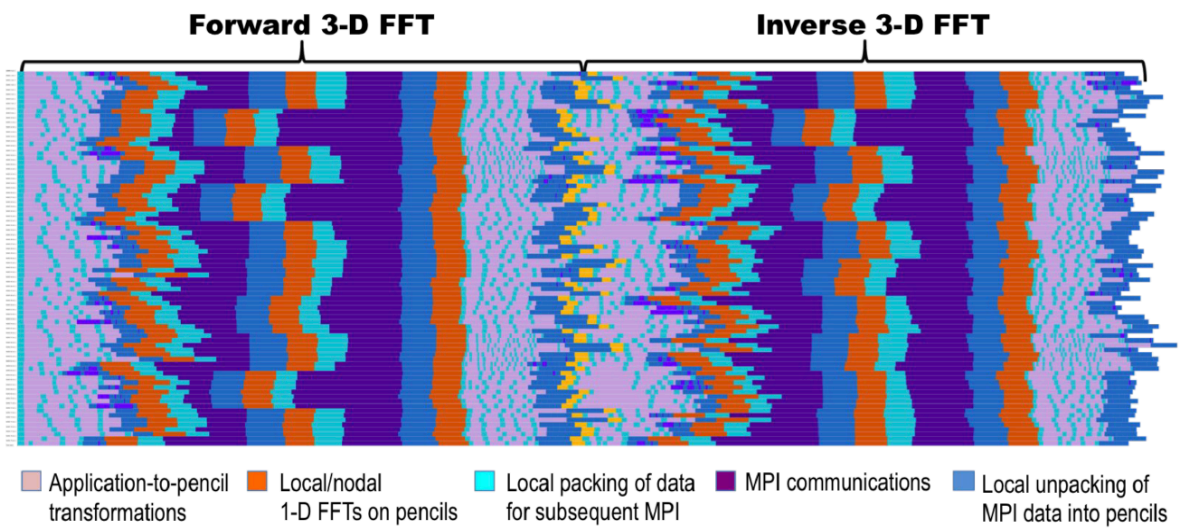


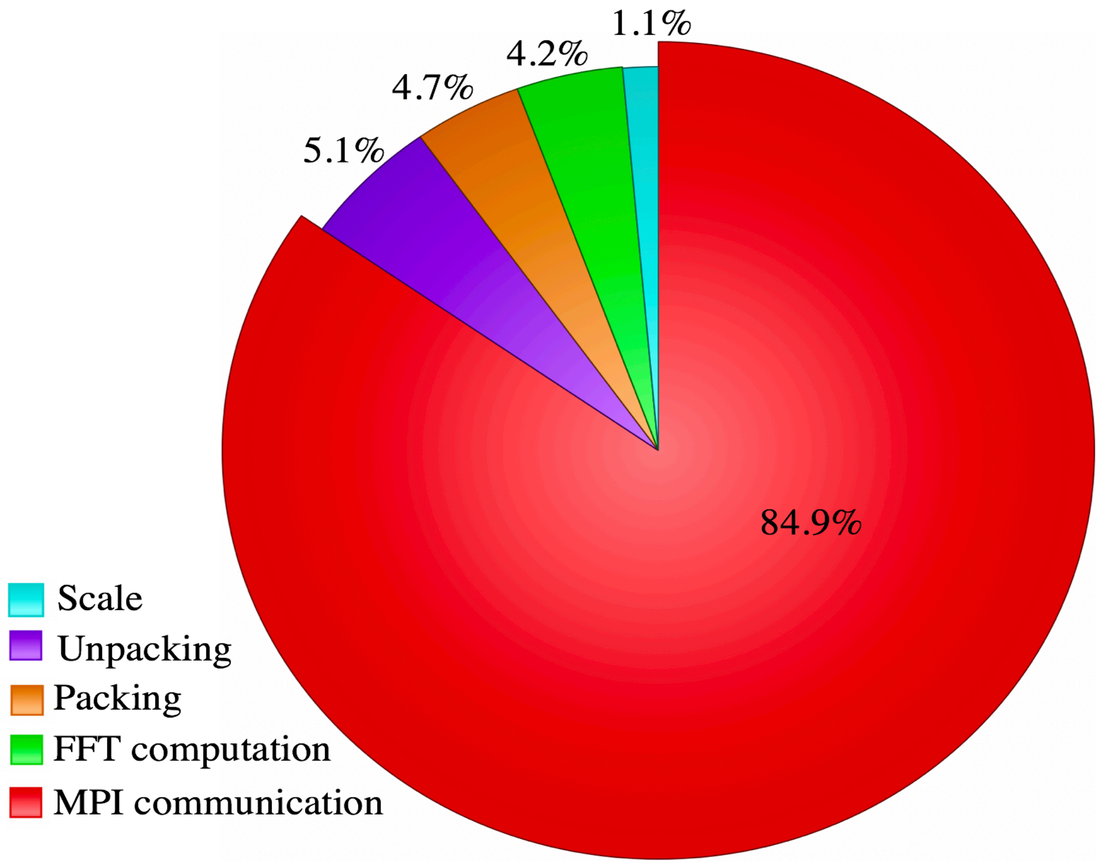
Figure: Strong Scalability on 32K Power9 cores for CPU-based libraries (left), and 4096 V-100 for GPU-based libraries (right).

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

heFFTe tracing tools

- We provide our own tracing function and scripts for direct link with vendor profilers.






```
heffte_tracing("start");
  heffte_execute(fft, work, work, FORWARD);
  heffte_execute(fft, work, work, BACKWARD);
heffte_tracing("stop"));
```

```
mpirun -np 2 ./vampir_trace.sh ./heffte_exec -my_options ...
```

Integration to ECP EXAALT

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×.

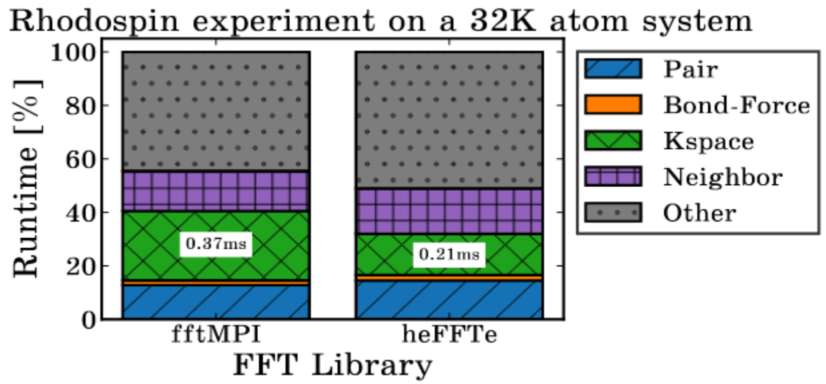


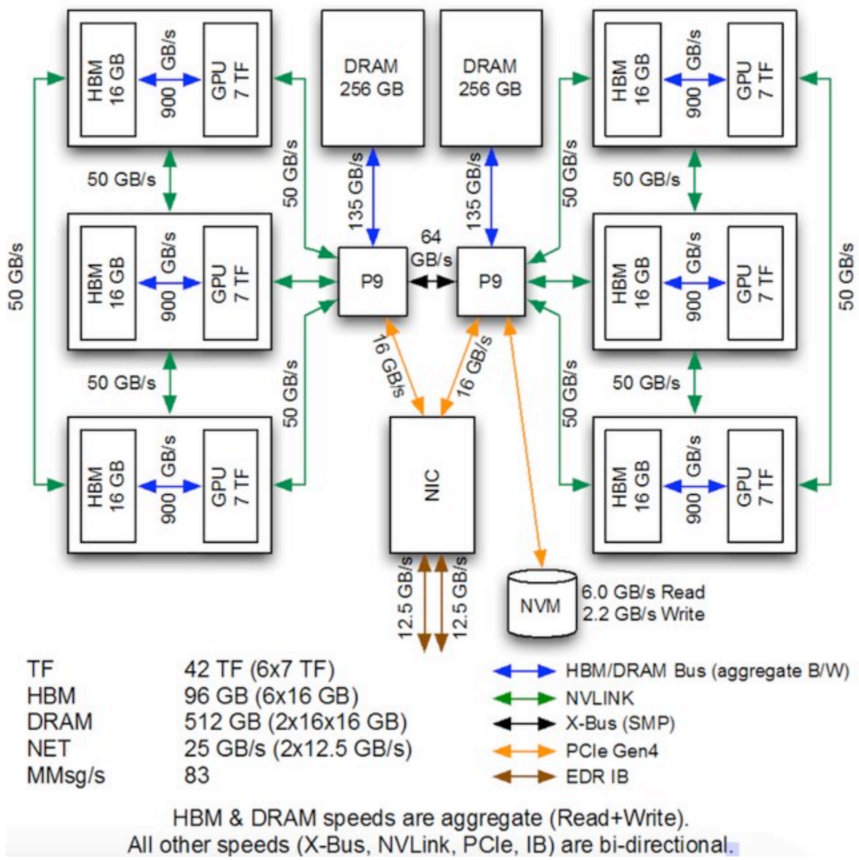
Figure: Breakdown for the LAMMPS Rhodopsin experiment. Using 32 Summit nodes, 6 V-100 GPUs per node, and 1 MPI per GPU.

Ref.: Performance Analysis of Parallel FFT on Large Multi-GPU Systems. *Ayala et al., IEEE IPDPS 2022.*

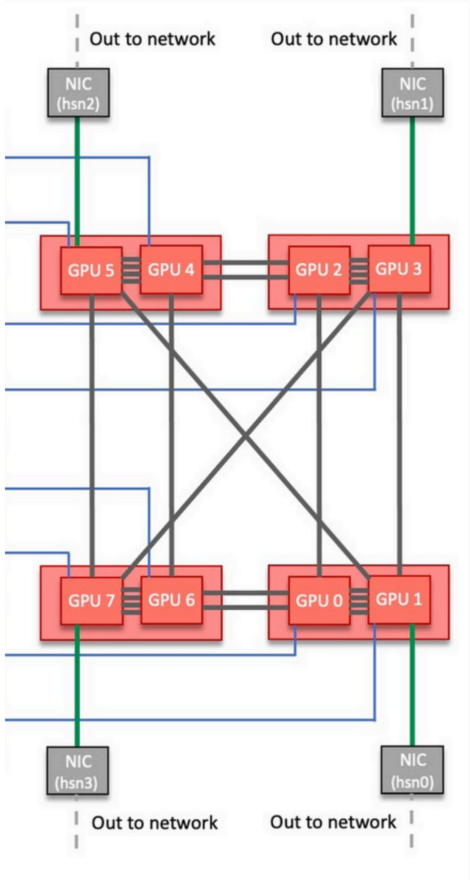
heFFTe on Frontier

- How should performance compare to Summit ?

Summit node



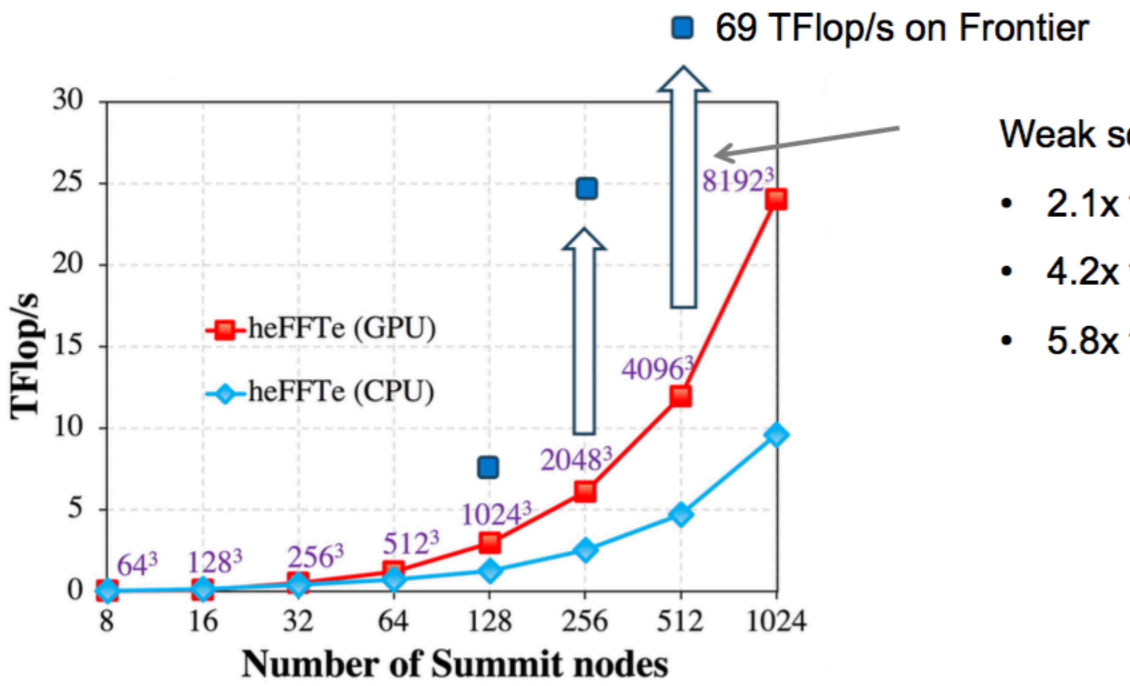
Frontier node



- 4 NICs (25 GBs + 25 GBs) in Frontier vs.
  1 NIC (25 GBs + 25 GBs) in Summit
- Expect to see 4 x speedup on Frontier vs. Summit on communication-bound codes like FFT (asymptotically for the same number of nodes)

heFFTe on Frontier

- **How do we compare to Summit?**

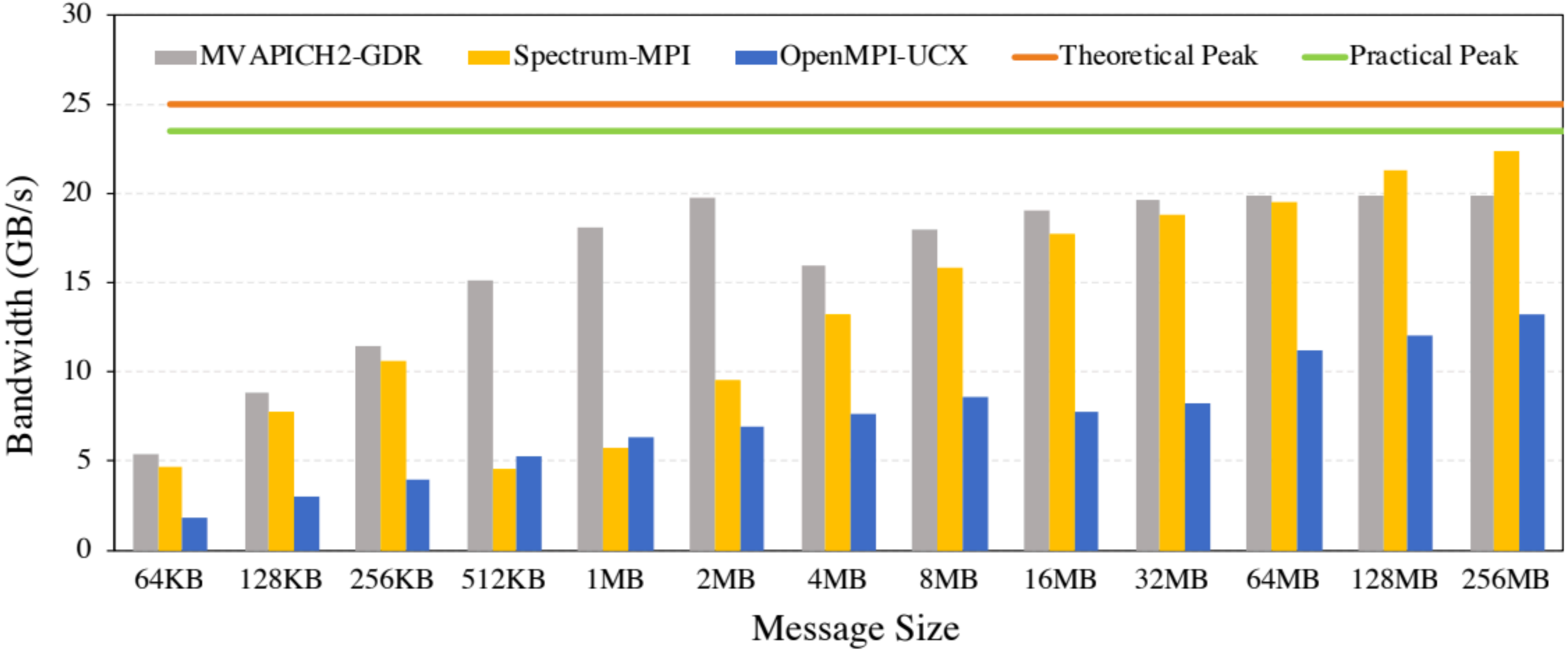


Weak scaling comparison

- 2.1x faster than on summit for 128 nodes
- 4.2x faster than on summit for 256 nodes
- 5.8x faster than on summit for 512 nodes

heFFTe using MVAPICH

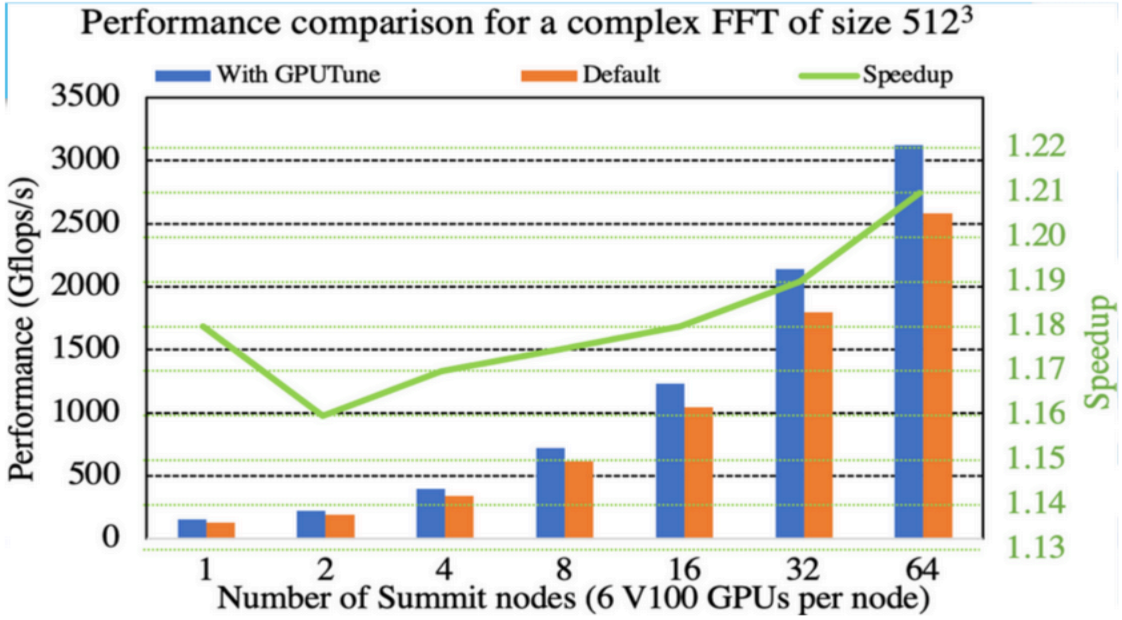
Strong scalability on 3D FFTs of size 1024^3, using 24 MPI processes (1 MPI per Power9 core) per node (blue), and 24 MPI processes (4 MPI per GPU-V100) per node (red)



Tuning heFFTe

Parameterize FFT implementation and expose parameters for tuning (a2a, a2av, a2aw, p2p, blocking/non-blocking, grid sizes, layouts, etc.)

- Auto-tuning heFFTe using GPTune (https://gptune.lbl.gov/), we were able to increase performance by tuning FFT input parameters and communication settings
- Shown is performance improvements and speedup on Summit (~15 - 20%)



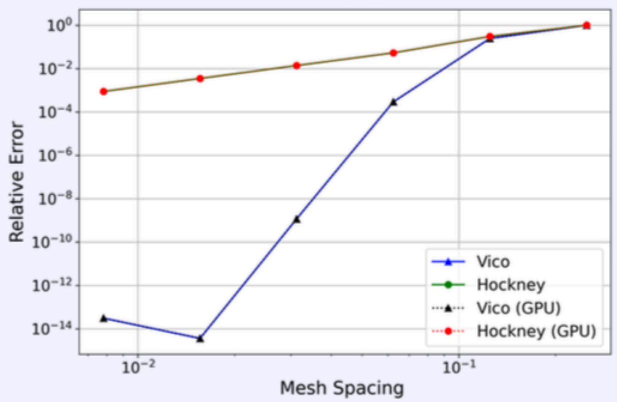
Approximate FFT with run-time lossy data compression

- Some solvers do not require full FP64 accuracy FFTs

Novel Poisson solver: Problem

What do we gain?

Convergence for smooth Gaussian source:



- These solvers are in IPPL and require Discrete Cosine Transform of type 1 Montanaro et al. (ETH)

FFT API for approximate FFTs

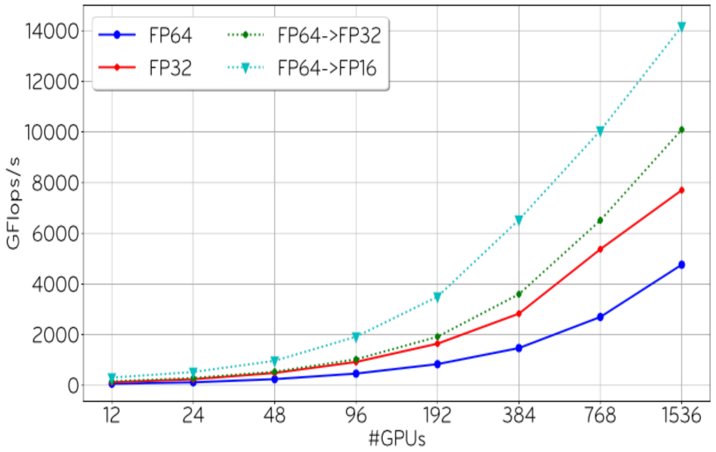
Algorithm 1 Approximate 3D FFT with lossy compression.

Input : 3D data $D_{x,y,z}$ in FP64 precision and error tolerance e_{tol}

Output: Approximate 3D FFT of $D_{x,y,z}$ stored in FP64 within error tolerance e_{tol}

1: **for** $i := x, y, z$ **do**
2: Custom Alltoall (Algorithm 3) combined with data $D_{x,y,z}$ compression/decompression within an error tolerance of e_{tol} in direction i
3: Batched 1D FFTs for direction i in FP64
4: **end for**

Speedup and accuracy of FFT using casting



Accuracy of approximate FFTs in heFFTe for 1024^3 FFTs

#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

TABLE II

COMPARISON OF THE FFT ACCURACY WHEN USING CASTING OPERATION FROM FP64 TO FP32 IN THE COMMUNICATION WITH THE TWO REFERENCES. EACH REFERENCE CORRESPONDS TO AN EXECUTION USING A UNIQUE PRECISION WHICH IS EITHER FP64 OR FP32.

PROJECT OVERVIEW

TITLE
Collaborative Research: Frameworks: Performance Engineering Scientific Applications with MVAPICH and TAU using Emerging Communication Primitives

SPONSOR
NSF CSSI

TEAM MEMBERS
PI: Dhabaleswar K. (DK) Panda,
Lead Institution: The Ohio State University
Aamir Shafi, Hari Subramoni, Mustafa Abduljabbar (**OSU**)
Sameer Shende (**UO**)
Yifeng Cui, Daniel Roten (**SDSC**)
Stan Tomov (**UTK**)

GRAPHICAL REPRESENTATION

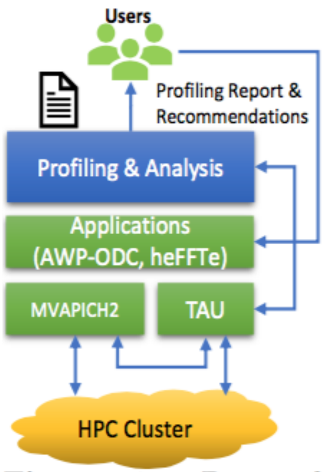
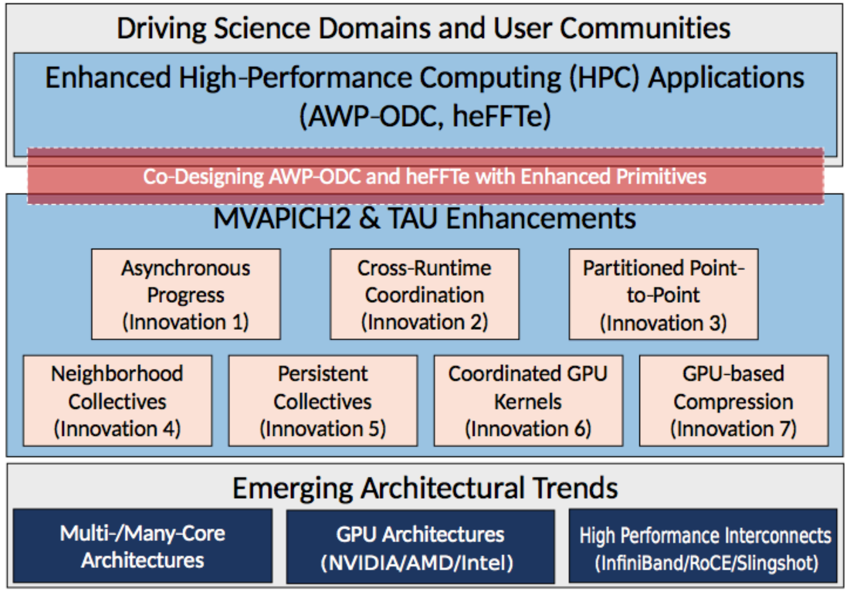


Figure 1: Proposed Profiling Framework



TECHNICAL DETAILS

OVERALL PROJECT OBJECTIVE
Co-design using the MPI_T API—in the MVAPICH2 and TAU libraries with scientific applications. Focus is on two popular HPC applications spanning multiple domains and representing various communication patterns - Anelastic Wave Propagation (AWP-ODC) and heFFTe. AWP-ODC is a highly scalable parallel finite-difference application with point-to-point operations that enables 3D earthquake calculations.

UTK CONTRIBUTION
HeFFTe, dominated by collective operations, is a massively parallel application that provides a scalable and efficient implementation of the widely used Fast Fourier Transform (FFT) operations.

IMPACT AND IMPLEMENTATION

BROADER IMPACT
Impact on driving guidelines for designing, deploying, and utilizing next-generation HPC systems for various application domains.

SCIENTIFIC IMPACT
Develop the next generation innovations by co-designing MVAPICH2 and TAU libraries to scale driving science domains—including AWP-ODC and heFFTe

NEXT STEPS & TIMELINE
3 years project, integrating heFFTe with innovations in MPI and TAU

Collaborators and Support

- 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)