heFFTe Tutorial

Stan Tomov, Miroslav Stoyanov (ORNL),
Alan Ayala (AMD), Azzam Haidar (NVIDIA), and Jack Dongarra

students and outside collaborators

Innovative Computing Laboratory
University of Tennessee, Knoxville

Exascale Computing Project
2023 ECP Tutorials
February 9, 2023

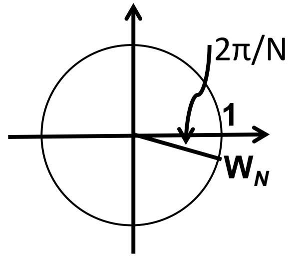
The Fast Fourier Transform (FFT)

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

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 20th century
  - **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

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

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

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

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 20th 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:

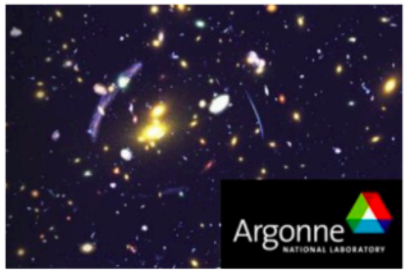
$$y(k_1, k_2, \ldots, k_m) := \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \cdots \sum_{n_m=0}^{N_m-1} \tilde{x} \cdot e^{-2\pi i \left(\frac{k_1 n_1}{N_1} + \frac{k_2 n_2}{N_2} \cdots + \frac{k_m n_m}{N_m} \right)},$$

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

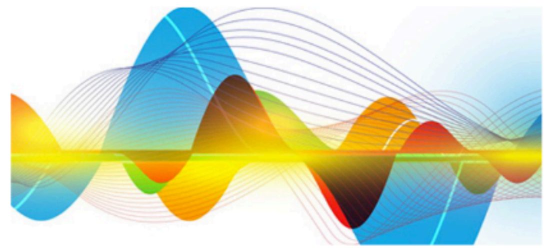
- A naive DFT costs $\mathcal{O}(N^2)$
- Using the FFT, the cost can be reduced to $\mathcal{O}(N \log_2 N)$.

Applications Relying on Parallel FFTs

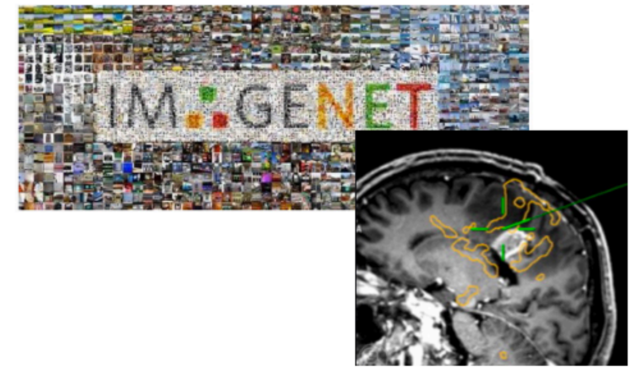
Cosmology
ECP **ExaSky - HACC**

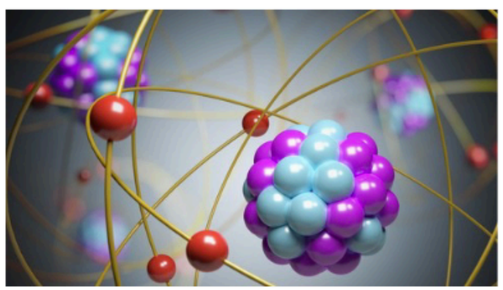


Signal processing,
***ECP* WARPX**

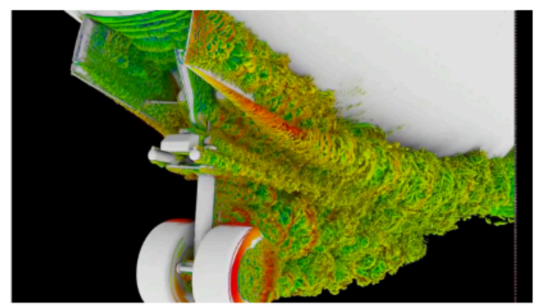


Deep Learning

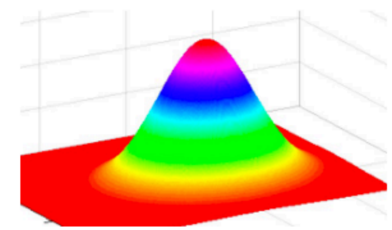




Molecular Dynamics
***ECP* EXAALT**



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



PDE solutions, **MASSIF**

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

Examples of FFT use

- **Spectral methods to solve PDEs**

Δ 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 Δ 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)

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

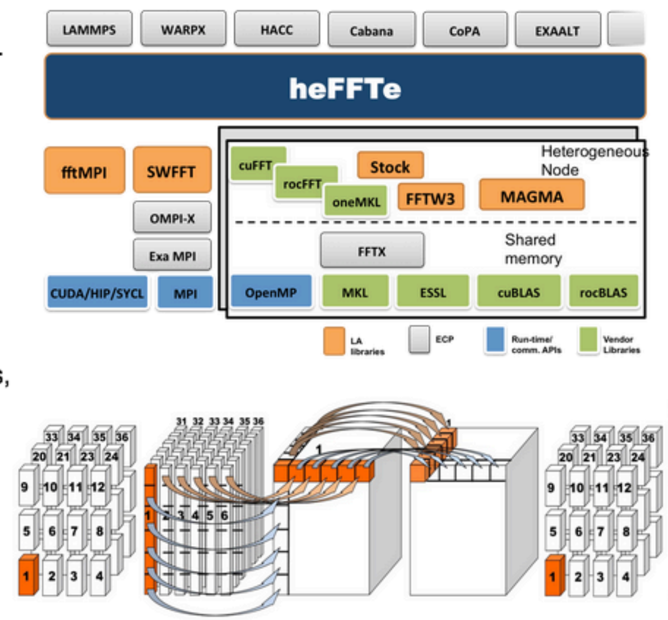
```
>> m = 100;            n = 50;
>> f = rand(m, 1);     g = rand(n, 1);

>> 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 FFTs for Exascale

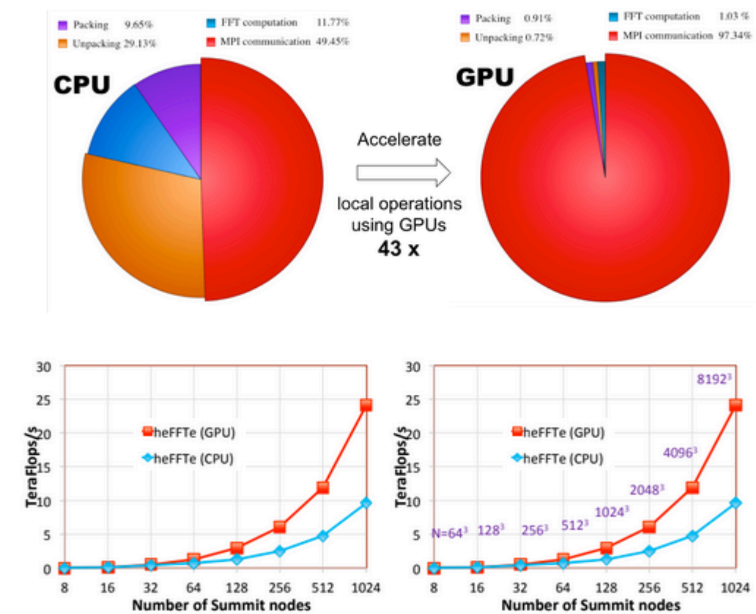
THEN

- 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.



ICL THE UNIVERSITY OF TENNESSEE KNOXVILLE

heFFTe
Highly Efficient FFTs for Exascale

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

heFFTe

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

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:

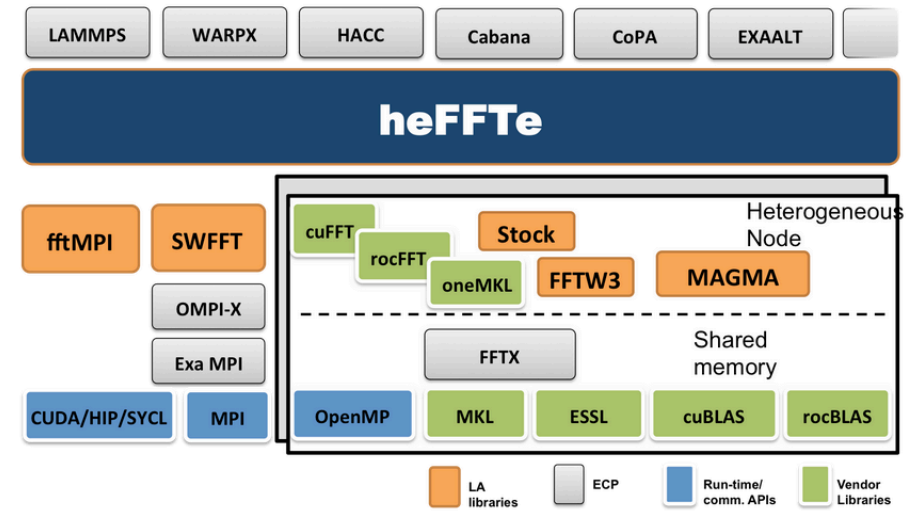
- **Summit @ OLCF (Nvidia GPUs)**
- **Crusher / Frontier (AMD GPUs), and others**
- **Florentia / Aurora (Intel GPU)**

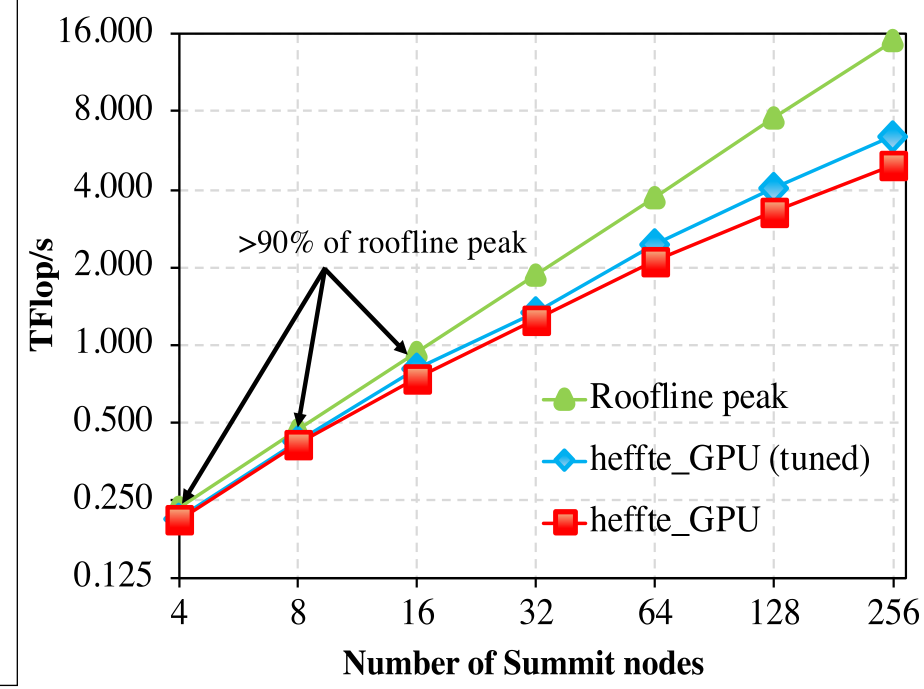
Current status:

- **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://bitbucket.org/icl/heffte/





heFFTe Availability

Availability

- http://icl.utk.edu/fft/ download, documentation
- https://bitbucket.org/icl/heffte Git repo
- Latest release is heFFTe 2.3

Support

- Linux
- CPU, Nvidia GPUs, AMD GPUs, Intel GPUs
- CUDA >= 7.0; recommend latest CUDA
- CUDA architecture >= 2.0 (Fermi, Kepler, Maxwell, Pascal, Volta, Ampere, Hopper)
- 1D FFTs: Stock, cuFFT, FFTW, oneMKL, IBM ESSL, rocFFT, ...

Installation options

1. Cmake
   - `heffte> mkdir build && cd build`

   To install GPU-enabled heFFTe, e.g., for NVIDIA GPUs:
   - `heffte/build> cmake –DHeffte_ENABLE_CUDA=ON ..`
   - `heffte/build> make –j && make install`
2. Spack
   - `Part of xSDK and E4S`
   - `spack install heffte`
3. Dependencies are 1D FFTs (open source or vendor) libraries

   `Stock, CUFFT (NVIDIA), oneMKL (Intel), FFTW3, ROCFFT (AMD)`

For further detail see: https://bitbucket.org/icl/heffte/src/master/doxygen/

Installation options ...

Typical CMake build command:

```
cmake \
    -D CMAKE_BUILD_TYPE=Release \
    -D BUILD_SHARED_LIBS=ON      \
    -D CMAKE_INSTALL_PREFIX=<path-for-installation> \
    -D Heffte_ENABLE_AVX=ON \
    -D Heffte_ENABLE_FFTW=ON \
    -D FFTW_ROOT=<path-to-fftw3-installation> \
    -D Heffte_ENABLE_CUDA=ON \
    -D CUDA_TOOLKIT_ROOT_DIR=<path-to-cuda-installation> \
    <path-to-heffte-source-code>
```

Additional heFFTe options:

```
Heffte_ENABLE_ROCM=<ON/OFF>       (enable the rocFFT backend)
Heffte_ENABLE_ONEAPI=<ON/OFF>     (enable the oneMKL backend)
Heffte_ENABLE_MKL=<ON/OFF>        (enable the MKL backend)
Heffte_ENABLE_AVX512=<ON/OFF>     (enable AVX512 instructions in the stock backend)
MKL_ROOT=<path>                   (path to the MKL folder)
Heffte_ENABLE_DOXYGEN=<ON/OFF>    (build the documentation)
Heffte_ENABLE_TRACING=<ON/OFF>    (enable the even logging engine)
```

Additional language interfaces and helper methods:

```
-D Heffte_ENABLE_PYTHON=<ON/OFF>   (configure the Python module)
-D Heffte_ENABLE_FORTRAN=<ON/OFF>  (build the Fortran modules)
-D Heffte_ENABLE_SWIG=<ON/OFF>     (generate new Fortrans source files)
-D Heffte_ENABLE_MAGMA=<ON/OFF>    (link to MAGMA for helper methods)
```

GPU-Aware MPI

Different implementations of MPI can provide GPU-Aware capabilities, where data can be send/received directly in GPU memory. OpenMPI provided CUDA aware capabilities if compiled with the corresponding options, e.g., see CUDA-Aware OpenMPI. Both CUDA and ROCm support such API; however, the specific implementation available to the user may not be available for various reasons, e.g., insufficient hardware support. HeFFTe can be compiled without GPU-Aware capabilities with the CMake option:

```
-D Heffte_DISABLE_GPU_AWARE_MPI=ON
```

Note: Only one of the GPU backends can be enabled (CUDA, ROCM, or ONEAPI) since the three backends operate with arrays allocated in GPU device memory (or alternatively shared/managed memory). By default when using either GPU backend, heFFTe assumes that the MPI implementation is GPU-Aware, see the next section.

```
-D Heffte_ENABLE_CUDA=ON
-D CUDA_TOOLKIT_ROOT_DIR=<path-to-cuda-installation>
```

```
-D CMAKE_CXX_COMPILER=hipcc
-D Heffte_ENABLE_ROCM=ON
```

```
-D CMAKE_CXX_COMPILER=dpcpp
-D Heffte_ENABLE_ONEAPI=ON
-D Heffte_ONEMKL_ROOT=<path-to-onemkl-installation>
```

For further detail see: https://bitbucket.org/icl/heffte/src/master/doxygen/

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	✓	✓	✓	✓

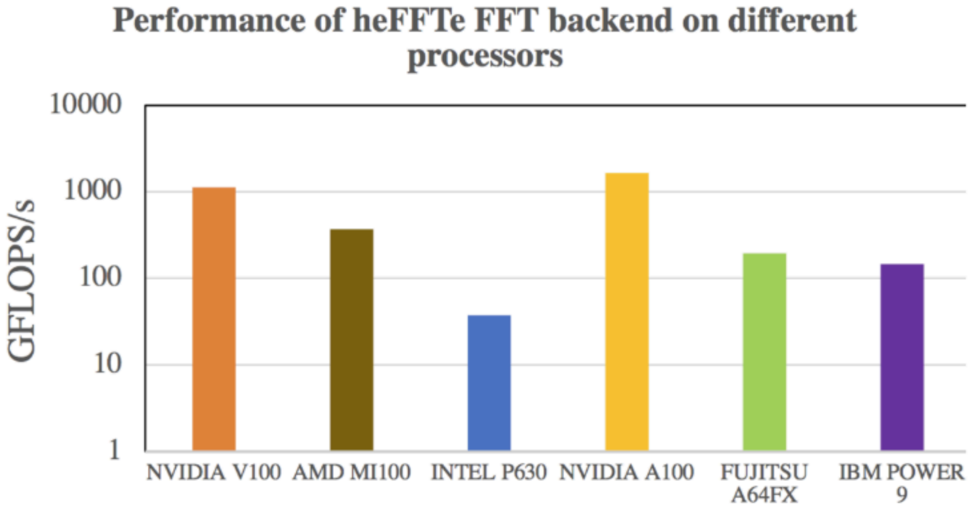
Figure: State-of-the-art of FFT libraries targeting a single-device unit.

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.



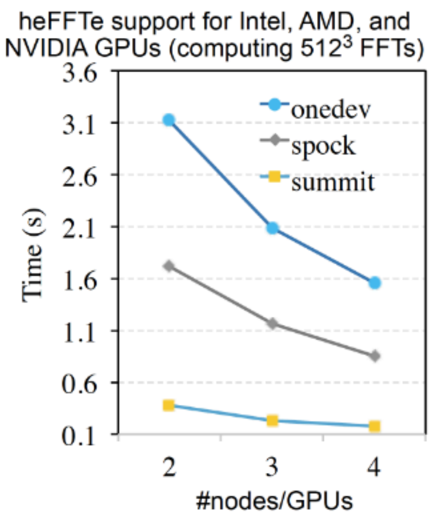


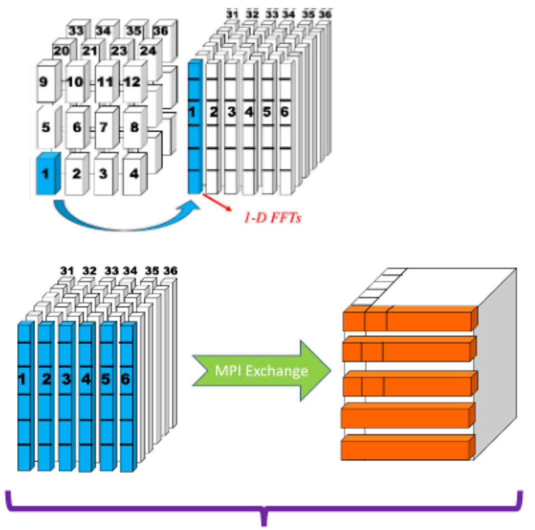
Figure: Comparison of single-device performance for a 512^3 FFT.

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}

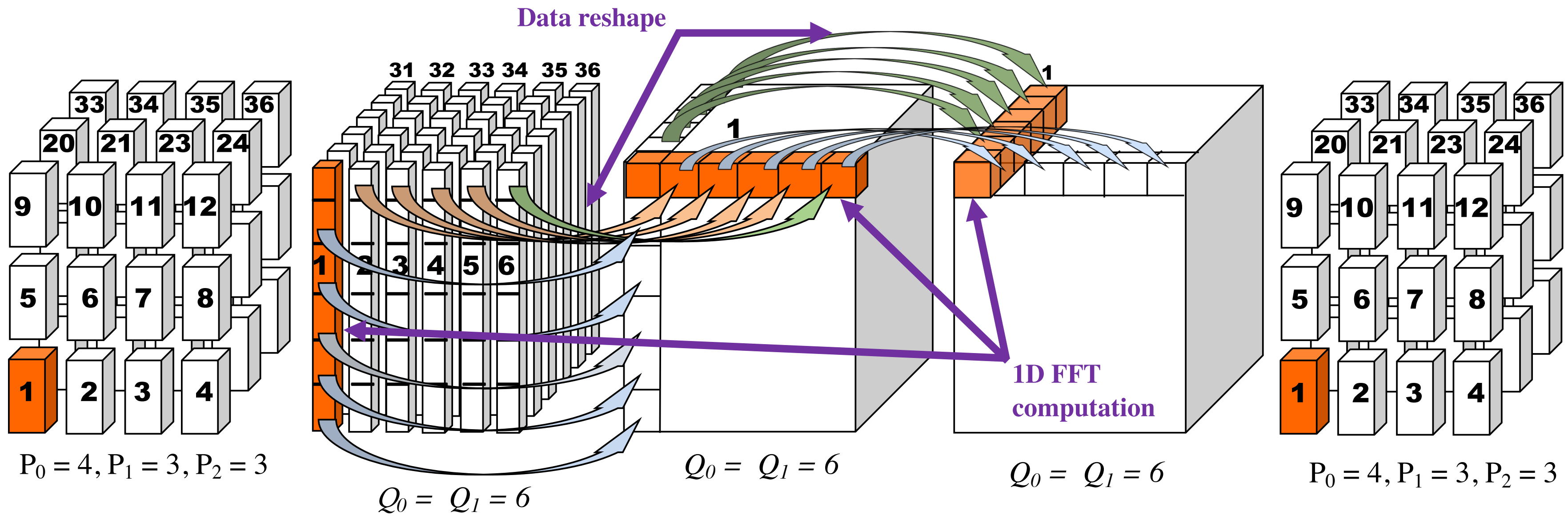


These 3 tasks can be replaced by 1 via MPI_Alltoallw

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

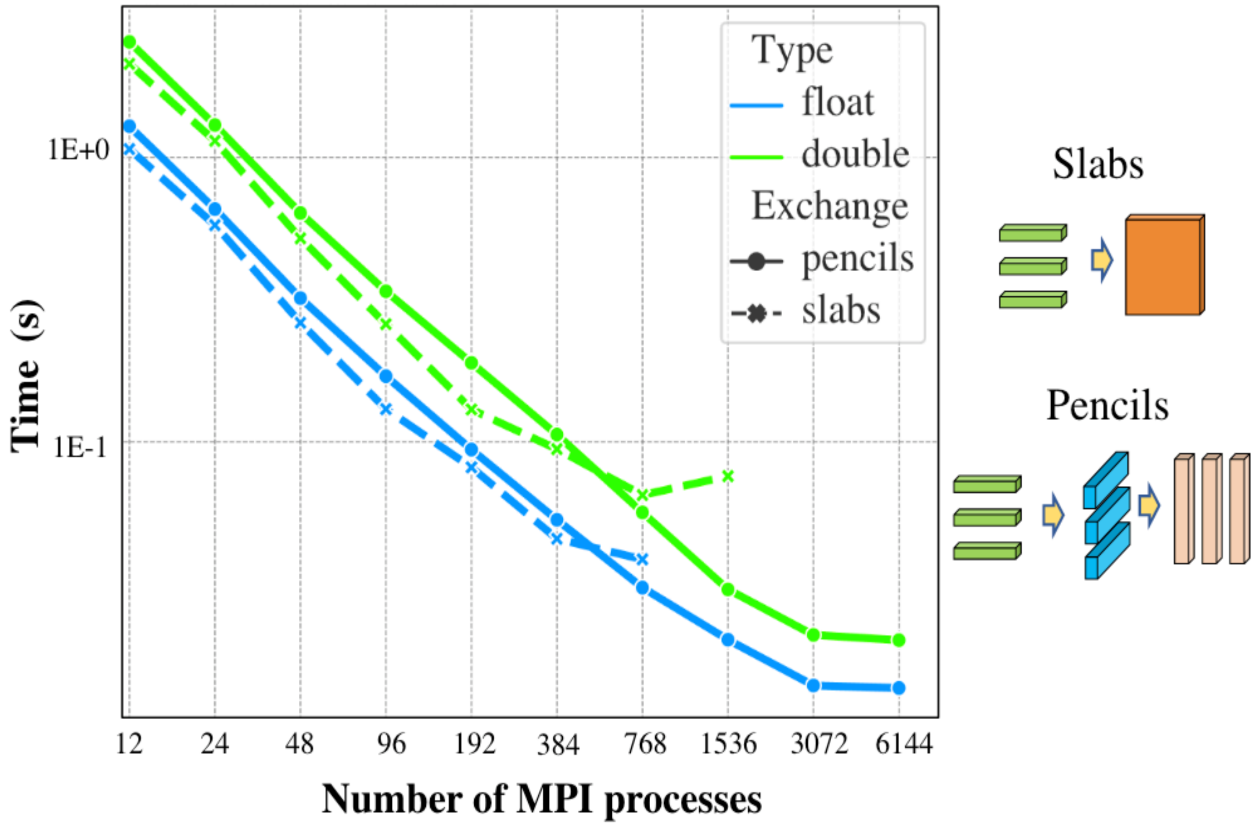
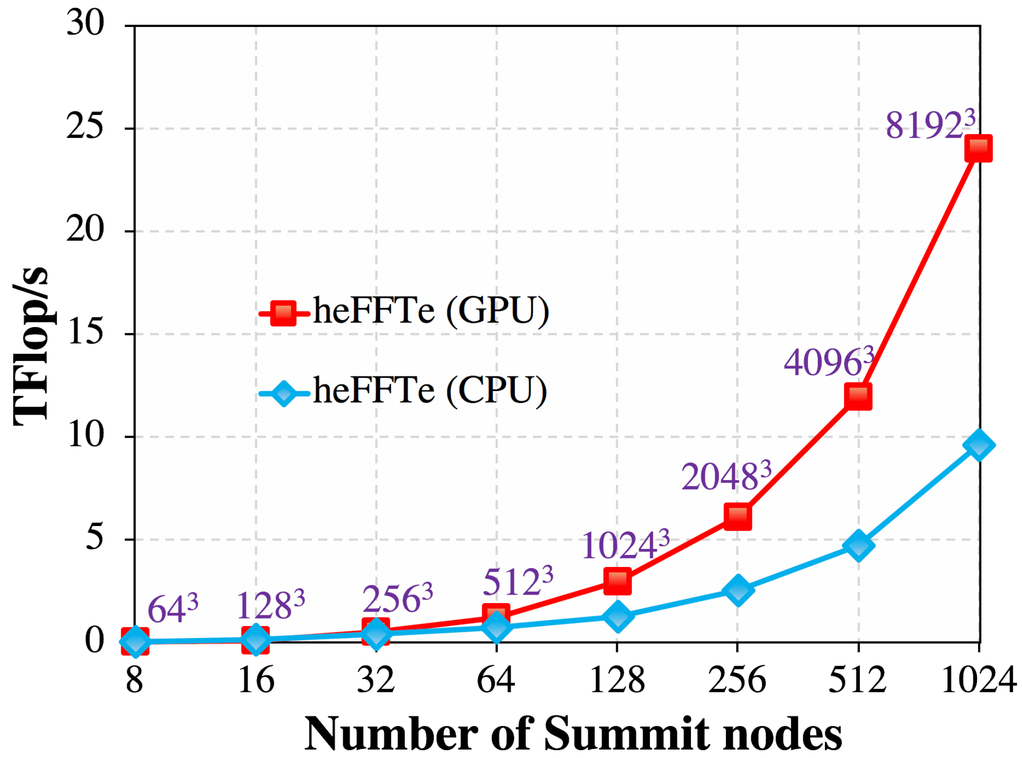
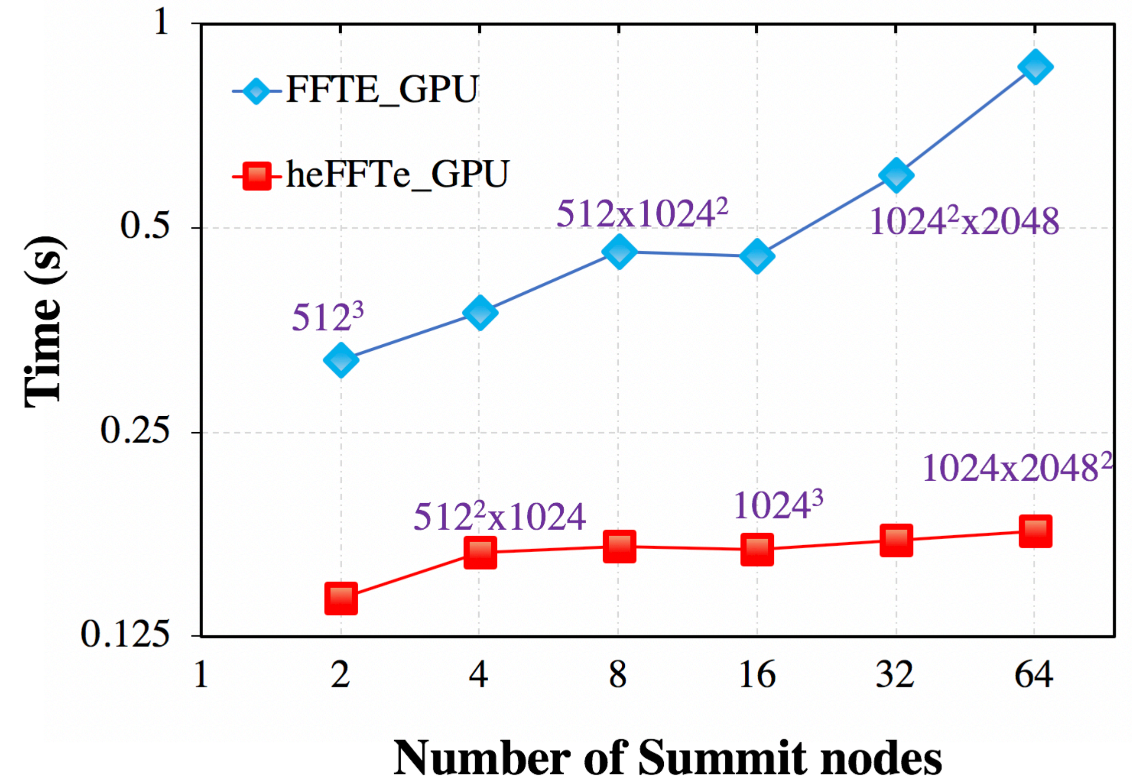


Fig. 6. Comparison of pencil and slab decompositions for strong scaling of a 3-D FFT of size 1024^3. Using *heFFTe* with cuFFT backend, 6 MPI processes (1 MPI processes per GPU-V100) per node, and single-precision complex data.

heFFTe Weak Scalability

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

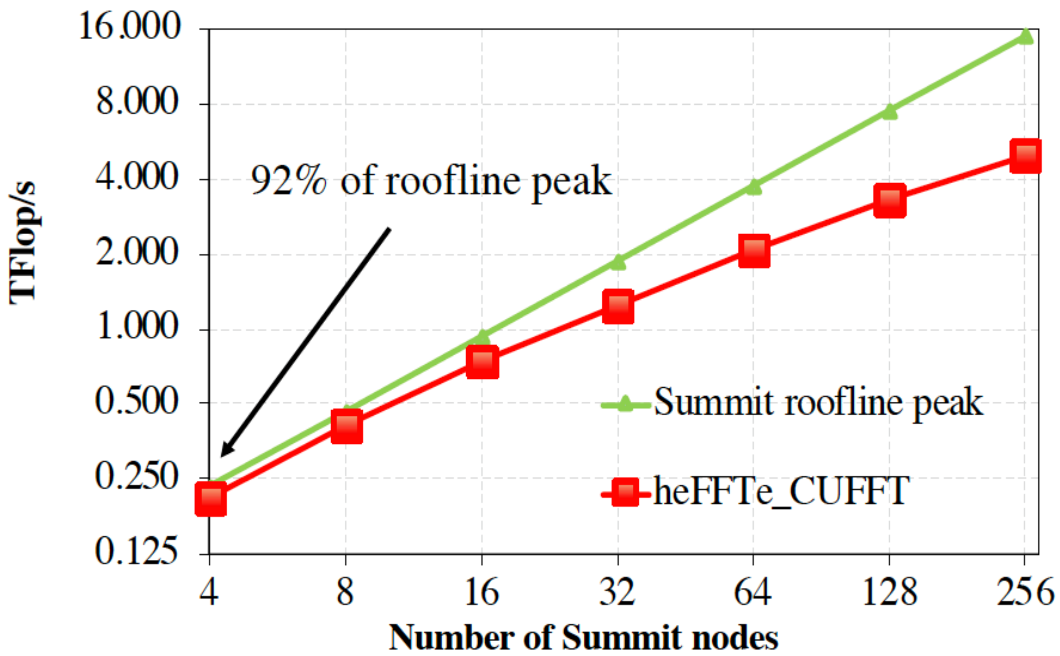


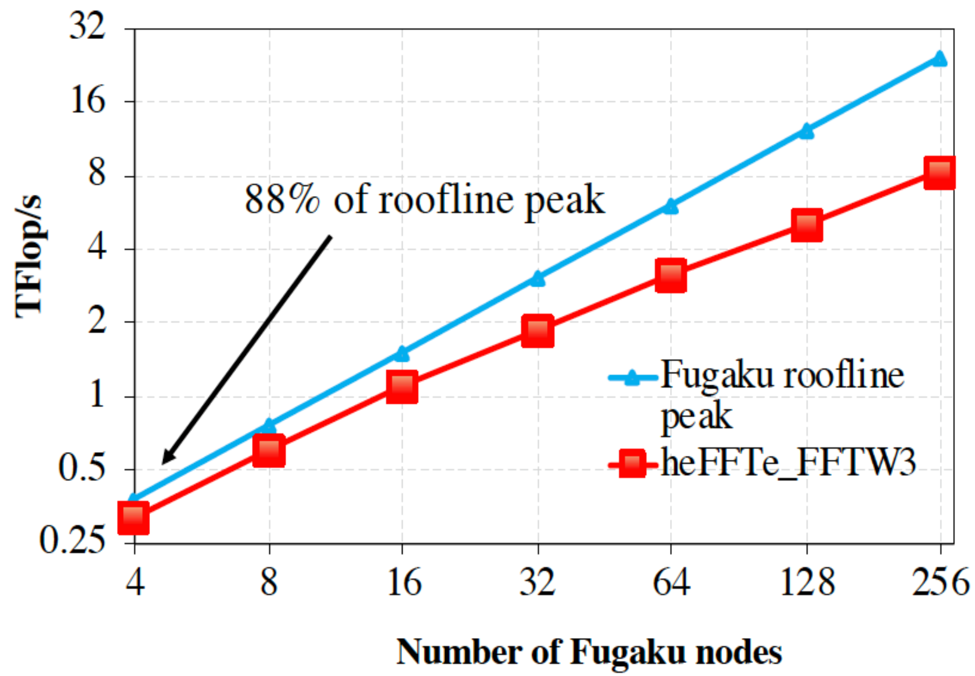


Forward and backward FFT on a Complex 3D array in double precision.

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.

FIBER FFT benchmark (https://github.com/icl-utk-edu/fiber)

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.

ICL THE UNIVERSITY OF TENNESSEE KNOXVILLE

FIBER FFT benchmark (https://github.com/icl-utk-edu/fiber)

Scaling FFT on top Supercomputers

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

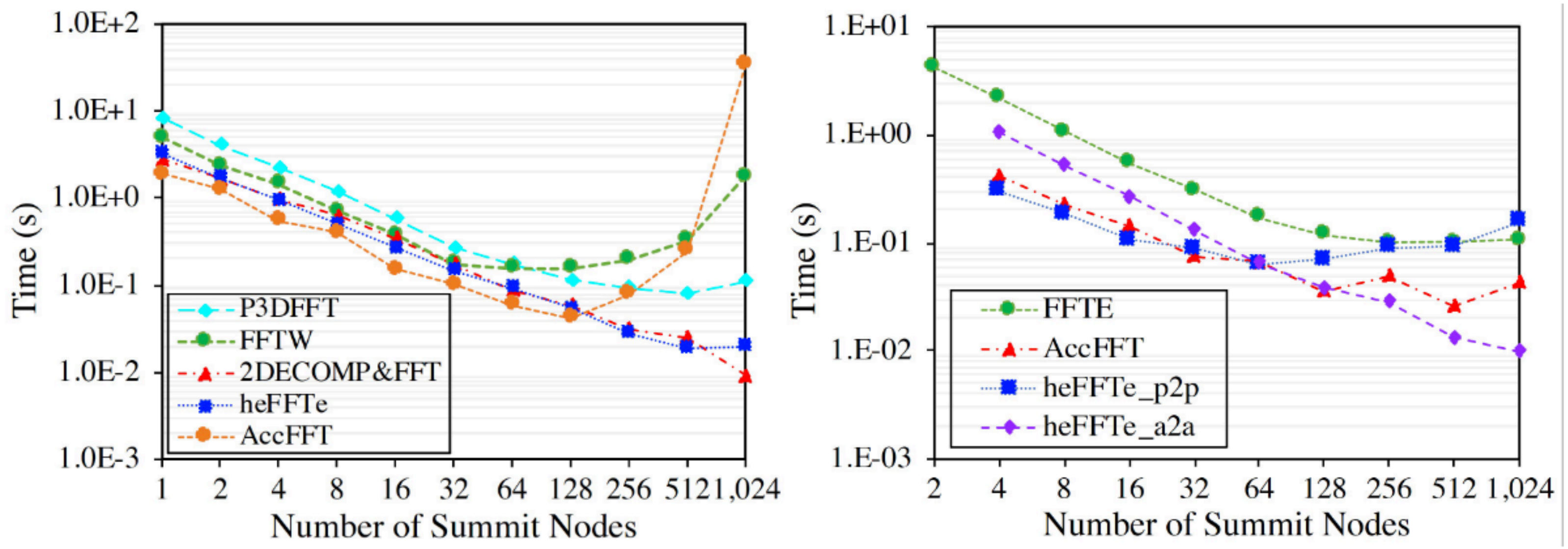
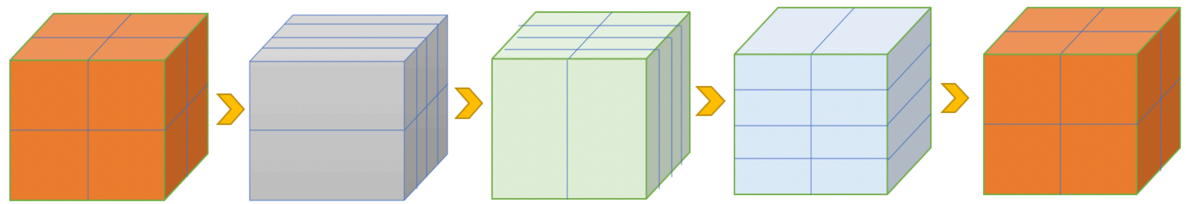


Figure: Strong Scalability on 32K Power9 cores for CPU-based libraries (left), and 4096 V-100 for GPU-based libraries (right).

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

heFFTe API

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



Input processor grid

```
std::array<int,3>  proc_i
```

Output processor grid

```
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:

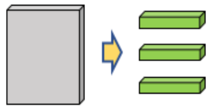
```
heffte ::proc_setup_min_surface(my_mpi_comm, nprocs);
```

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);
```

3. Select type of FFT intermediate reshape, via one of the following flags:

--slabs



--pencils

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.

```
heffte::add_trace(“Initiating tracing”);

    ---Code to be traced ---

heffte::add_trace(“Ending tracing”);
```

heFFTe API

Test drivers / examples / benchmarks

Directory benchmarks provides speed3d_c2c, speed3d_r2c, speed3d_r2r, and convolution benchmarks

```
Usage:
mpirun -np x <bench_executable> <backend> <precision> <size-x> <size-y> <size-z> <args>

       backend is the 1-D FFT library
       precision is either float or double
         use float-long or double-long to enable 64-bit indexing
       size-x/y/z are the 3D array dimensions
       args is a set of optional arguments that define algorithmic tweaks and variations
        -reorder: reorder the elements of the arrays so that each 1-D FFT will use contiguous data
        -no-reorder: some of the 1-D will be strided (non contiguous)
        -a2a: use MPI_Alltoall() communication method
        -a2av: use MPI_Alltoallv() communication method (default)
        -p2p: use MPI_Send() and MPI_Irecv() communication methods
        -p2p_pl: use MPI_Isend() and MPI_Irecv() communication methods
        -no-gpu-aware: move the data to the cpu before doing gpu operations (gpu backends only)
        -pencils: use pencil reshape logic
        -slabs: use slab reshape logic
        -io_pencils: if input and output proc grids are pencils, useful for comparison with other libraries
        -ingrid x y z: specifies the processor grid to use in the input, x y z must be integers
        -outgrid x y z: specifies the processor grid to use in the output, x y z must be integers
        -subcomm num_ranks: specifies the number of ranks to use in intermediate reshapes
        -batch batch_size: specifies the size of the batch to use in the benchmark
        -r2c_dir dir: specifies the r2c direction for the r2c tests, dir must be 0 1 or 2
        -mps: for the cufft backend and multiple gpus, associate the mpi ranks with different cuda devices
        -nX: number of times to repeat the run, accepted variants are -n5 (default), -n10, -n50

Examples:
   mpirun -np  4  speed3d_r2r  fftw-cos  double 128 128 128 -p2p
   mpirun -np  8  speed3d_r2r  cufft-cos float  256 256 256
   mpirun -np 12  speed3d_r2r  fftw-sin  double 512 512 512 -slabs

   mpirun -np  4  speed3d_c2c  fftw  double 128 128 128 -no-reorder
   mpirun -np  8  speed3d_c2c  cufft float  256 256 256
   mpirun -np 12  speed3d_r2c  fftw  double 512 512 512 -p2p -slabs
```

heFFTe API

Test drivers / examples / benchmarks

/* mpirun */
mpirun -np 2 speed3d_c2c fftw float 4 2 2 –a2a

```
-----------------------------------------------------------------------------------------------------------------
                                              Testing HEFFTE library
-----------------------------------------------------------------------------------------------------------------
Test summary:
-------------
        Computation of 3D FFT
        1 forward and 1 backward 3D-FFTs on 2 procs on a 4x2x2 grid
        1D FFT library       : FFTW3
        Precision            : SINGLE
        Comunication type    : ALL2ALL
        Scaling after forward: YES
Memory comsuption:
------------------
        Memory usage (per-proc) for FFT grid   = 6.1e-05 MB
        Memory usage (per-proc) by FFT library = 0.00076 MB
        Total memory comsuption                = 0.0015 MB
Processor grids for FFT stages:
-------------------------------
        Initial grid    1st-direction   2nd-direction   3rd-direction
            2 1 1           1 1 2          2 1 1           2 1 1
                                                        (Final grid)

-----------------------------------------------------------------------------------------------------------------
 ID      np        nx       ny       nz        Gflops/s       One FFT (s)    Initialisation (s)       Max Error
_3D_      2         4        2        2        0.06804         3.26e-06          0.001659          6.917283e-08
-----------------------------------------------------------------------------------------------------------------
```

heFFTe API

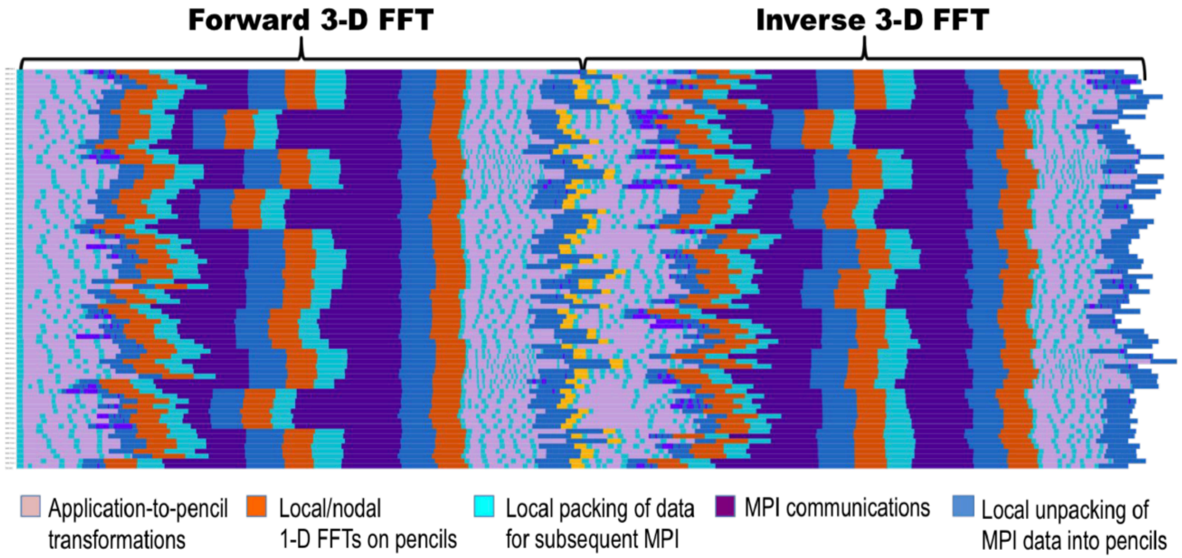
Test drivers / examples / benchmarks

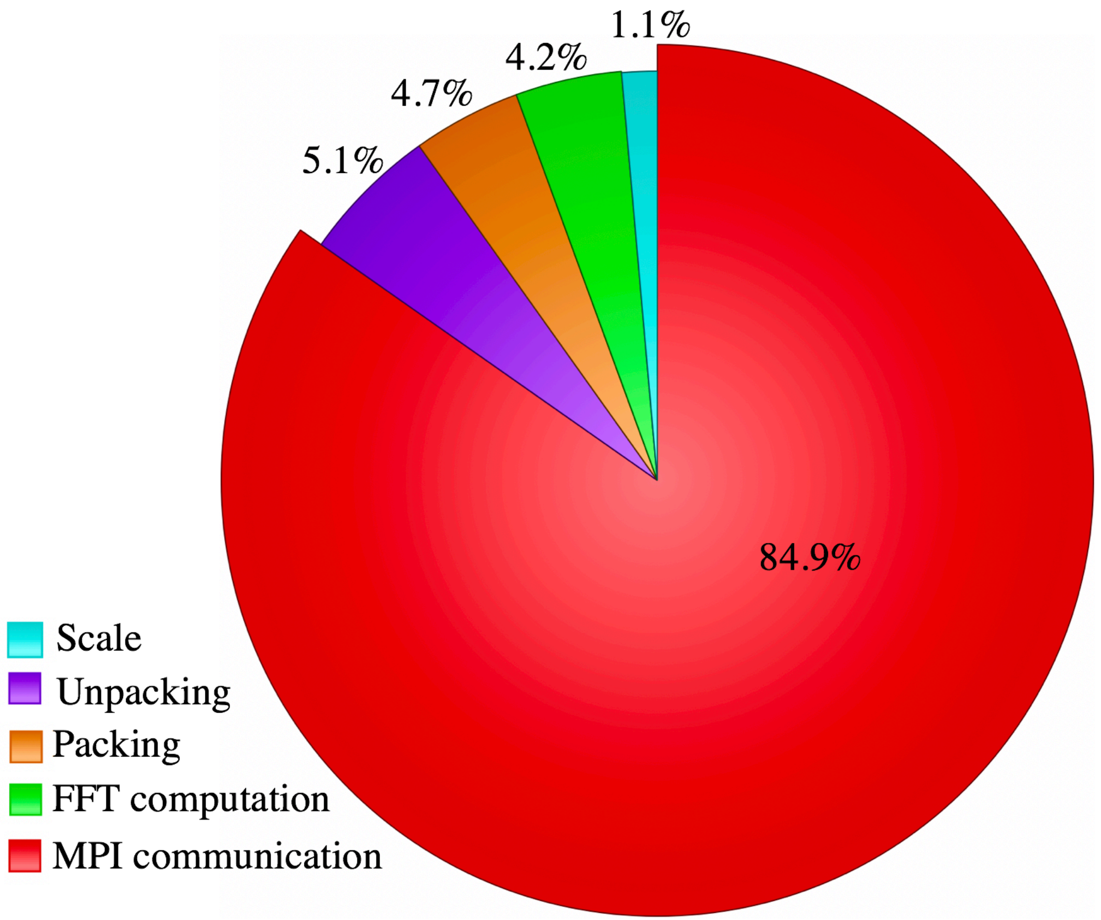
Should you have questions about the use of flags, please refer to `flags.md` for detailed information. For systems, such as Summit supercomputer, which support execution with `jsrun` by default, follow the examples:

```
jsrun  -n1280 -a1 -c1 -r40 ./speed3d_r2c fftw double 1024 256 512 -pencils
jsrun --smpiargs="-gpu" -n192 -a1 -c1 -g1 -r6 ./speed3d_c2c cufft double 1024 1024 1024 -p2p -reorder
```

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×.

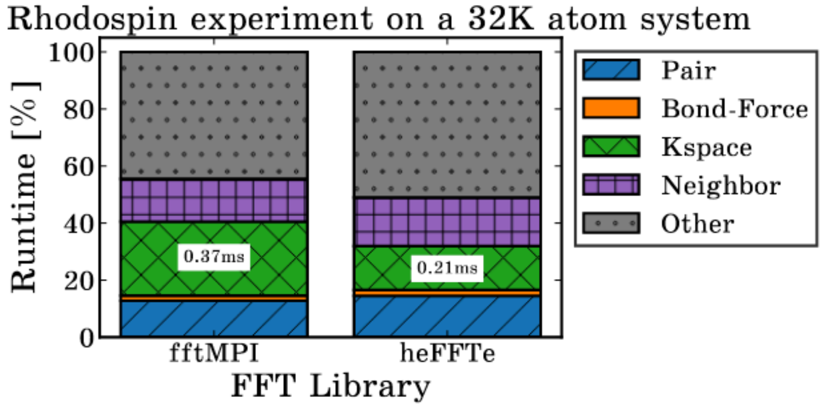


Figure: Breakdown for the LAMMPS Rhodopsin experiment. Using 32 Summit nodes, 6 V-100 GPUs per node, and 1 MPI per GPU.

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

Integration to ECP EXAALT

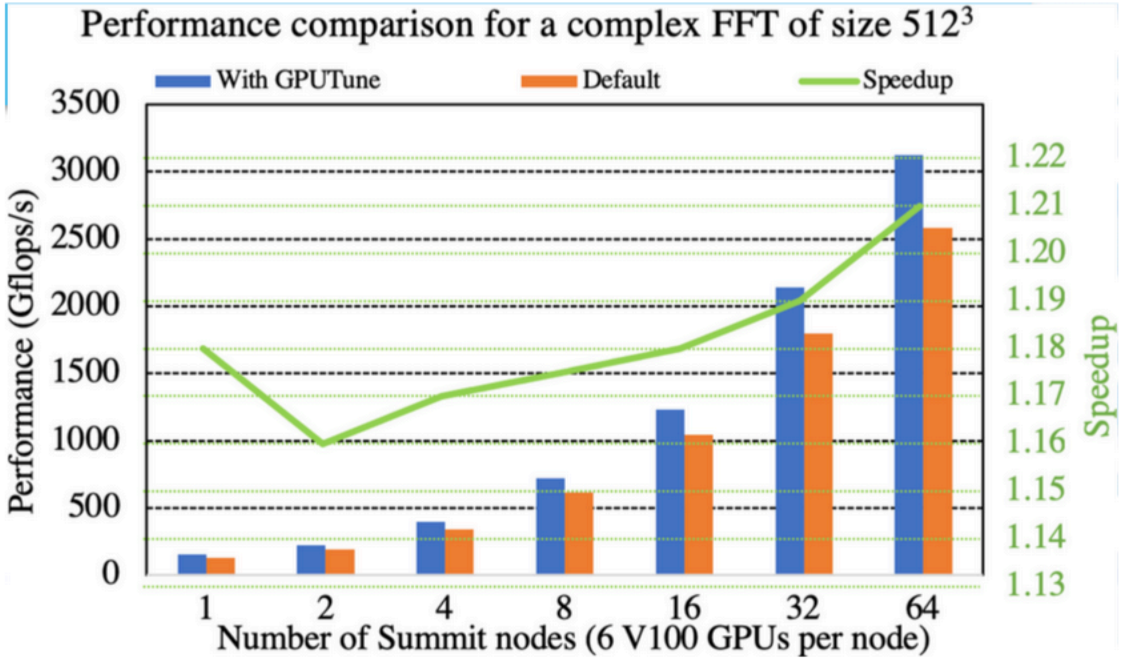
```
FFT3d::FFT3d(LAMMPS *lmp, MPI_Comm comm, int nfast, int nmid, int nslow,
             int in_ilo, int in_ihi, int in_jlo, int in_jhi,
             int in_klo, int in_khi,
             int out_ilo, int out_ihi, int out_jlo, int out_jhi,
             int out_klo, int out_khi,
             int scaled, int permute, int *nbuf, int usecollective) : Pointers(lmp)
{
  #ifndef HEFFTE
  plan = fft_3d_create_plan(comm,nfast,nmid,nslow,
                            in_ilo,in_ihi,in_jlo,in_jhi,in_klo,in_khi,
                            out_ilo,out_ihi,out_jlo,out_jhi,out_klo,out_khi,
                            scaled,permute,nbuf,usecollective);
  if (plan == nullptr) error->one(FLERR,"Could not create 3d FFT plan");
  #else
  heffte::plan_options options = heffte::default_options<heffte_backend>();
  options.algorithm = (usecollective == 0) ?
                        heffte::reshape_algorithm::p2p_plined
                      : heffte::reshape_algorithm::alltoallv;
  options.use_reorder = (permute != 0);
  hscale = (scaled == 0) ? heffte::scale::none : heffte::scale::full;

  heffte_plan = std::unique_ptr<heffte::fft3d<heffte_backend>>(
        new heffte::fft3d<heffte_backend>(
                heffte::box3d<>({in_ilo,in_jlo,in_klo}, {in_ihi, in_jhi, in_khi}),
                heffte::box3d<>({out_ilo,out_jlo,out_klo}, {out_ihi, out_jhi, out_khi}),
                comm, options)
    );
  *nbuf = heffte_plan->size_workspace();
  heffte_workspace.resize(heffte_plan->size_workspace());
  #endif
```

```
void FFT3d::compute(FFT_SCALAR *in, FFT_SCALAR *out, int flag)
{
  #ifndef HEFFTE
  fft_3d((FFT_DATA *) in,(FFT_DATA *) out,flag,plan);
  #else
  if (flag == 1)
      heffte_plan->forward(reinterpret_cast<std::complex<FFT_SCALAR>*>(in),
                           reinterpret_cast<std::complex<FFT_SCALAR>*>(out),
                           reinterpret_cast<std::complex<FFT_SCALAR>*>(heffte_workspace.data())
                          );
  else
      heffte_plan->backward(reinterpret_cast<std::complex<FFT_SCALAR>*>(in),
                            reinterpret_cast<std::complex<FFT_SCALAR>*>(out),
                            reinterpret_cast<std::complex<FFT_SCALAR>*>(heffte_workspace.data()),
                            hscale
                           );
  #endif
}
```

Tuning heFFTe

- 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%)



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)