MS73
Next Generation FFT Algorithms in Theory and Practice: Parallel Implementations and Applications

- **Organizers:**
  - **Daisuke Takahashi**
    University of Tsukuba, Japan
  - **Samar A. Aseeri**
    KAUST Supercomputing Laboratory, Saudi Arabia

Aim of this minisymposium

- The fast Fourier Transform (FFT) is an algorithm used in a wide variety of applications, yet does not make optimal use of many current hardware platforms.
- Hardware utilization performance, on its own, does not however, imply optimal problem solving.
- The purpose of this minisymposium is to enable the exchange of information between people working on alternative FFT algorithms, to those working on FFT implementations, in particular for parallel hardware.
- In addition to FFT algorithms, number-theoretical transform (NTT) is also included in the topic of this minisymposium.
- http://www.fft.report

MS73

- **10:00-10:20 Automatic Tuning for Parallel Number-Theoretic Transforms on GPU Clusters**

  Daisuke Takahashi, University of Tsukuba, Japan
- **10:25-10:45 Hybrid Quantum Annealing for Solving Nonlinear Differential Equations with Spectral Collocation**

  Samar A. Aseeri, KAUST Supercomputing Laboratory, Saudi Arabia
- **10:50-11:10 Parallelization of the Direct, Interpolative Non-Uniform Fast Fourier (NFFT) Transform for Imputation of Missing Values for Periodic Signals**

  Michael Armstrong and José Camacho*,* Universidad de Granada, Spain
- **11:15-11:35 Computational homogenization methods based on the FFT with superior accuracy**

  Flavia Gehrig, University of Duisburg-Essen, Germany;
  Matti Schneider, University of Duisburg, Germany

Automatic Tuning for Parallel Number-Theoretic Transforms on GPU Clusters

Daisuke Takahashi

Center for Computational Sciences
University of Tsukuba, Japan

Outline

- Background
- Objectives
- Number-Theoretic Transform (NTT)
- Four-Step NTT Algorithm
- Parallel Implementation of NTT
- Automatic Tuning for Parallel NTTs on GPU Clusters
- Performance Results
- Conclusion

Background

- The number-theoretic transform (NTT) is a generalization of the discrete Fourier transform to a finite field [Pollard 1971].
- The NTT is widely used for homomorphic encryption, polynomial multiplication, and multiple-precision multiplication.
- Several implementations of the NTT have been proposed.
- Many of these NTT implementations are for homomorphic encryption.

Related Works

- Implementations of the NTT on GPU have been proposed [Özerk et al. 2022]
  [Özcan and Savaş 2023].
- Vectorization and distributed-memory parallelization of the NTT to count Goldbach partitions on Arm-based supercomputers have also been proposed [Jesus et al. 2023].
- Parallel implementation of NTT on GPU clusters has been proposed [Takahashi 2025].

Objectives

- We perform the NTT on GPU clusters to further accelerate polynomial multiplication and multiple-precision multiplication.
- To the best of our knowledge, parallel NTT with automatic tuning on GPU clusters has not yet been presented.
- We propose an implementation of a parallel NTT with automatic tuning on GPU clusters.

Number-Theoretic Transform (NTT)

- The NTT can be expressed in a field $\mathbf{F}_p = \mathbf{Z}/p\mathbf{Z}$, where p is a prime number:

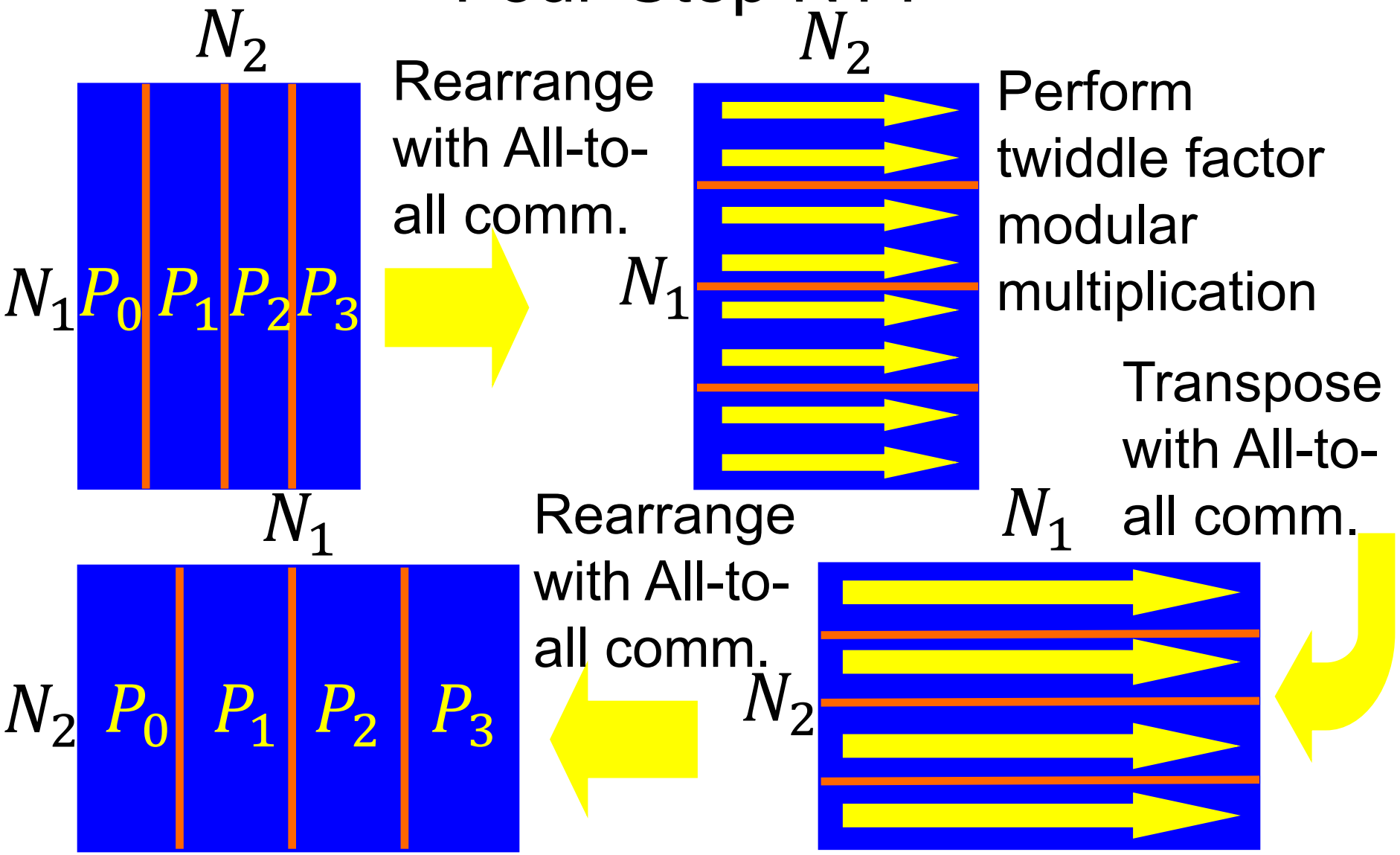
$$y(k) = \sum_{j=0}^{n-1} x(j)\omega_n^{jk} \bmod p, \quad 0 \le k \le n-1,$$

  where ω_n is the primitive n-th root of unity.
- The n-point NTT can be directly computed using $O(n^2)$ arithmetic operations.
- The number of arithmetic operations can be reduced to $O(n \log n)$ by applying an algorithm similar to the FFT.

Four-Step NTT Algorithm

- If n has factors n_1 and n_2 ($n = n_1 \times n_2$), similar to the four-step FFT algorithm [Bailey 1990], the following four-step NTT algorithm is derived:
- Step 1: n_1 simultaneous n_2-point multirow NTTs
- Step 2: Twiddle factor modular multiplication
- Step 3: Transposition
- Step 4: n_2 simultaneous n_1-point multirow NTTs

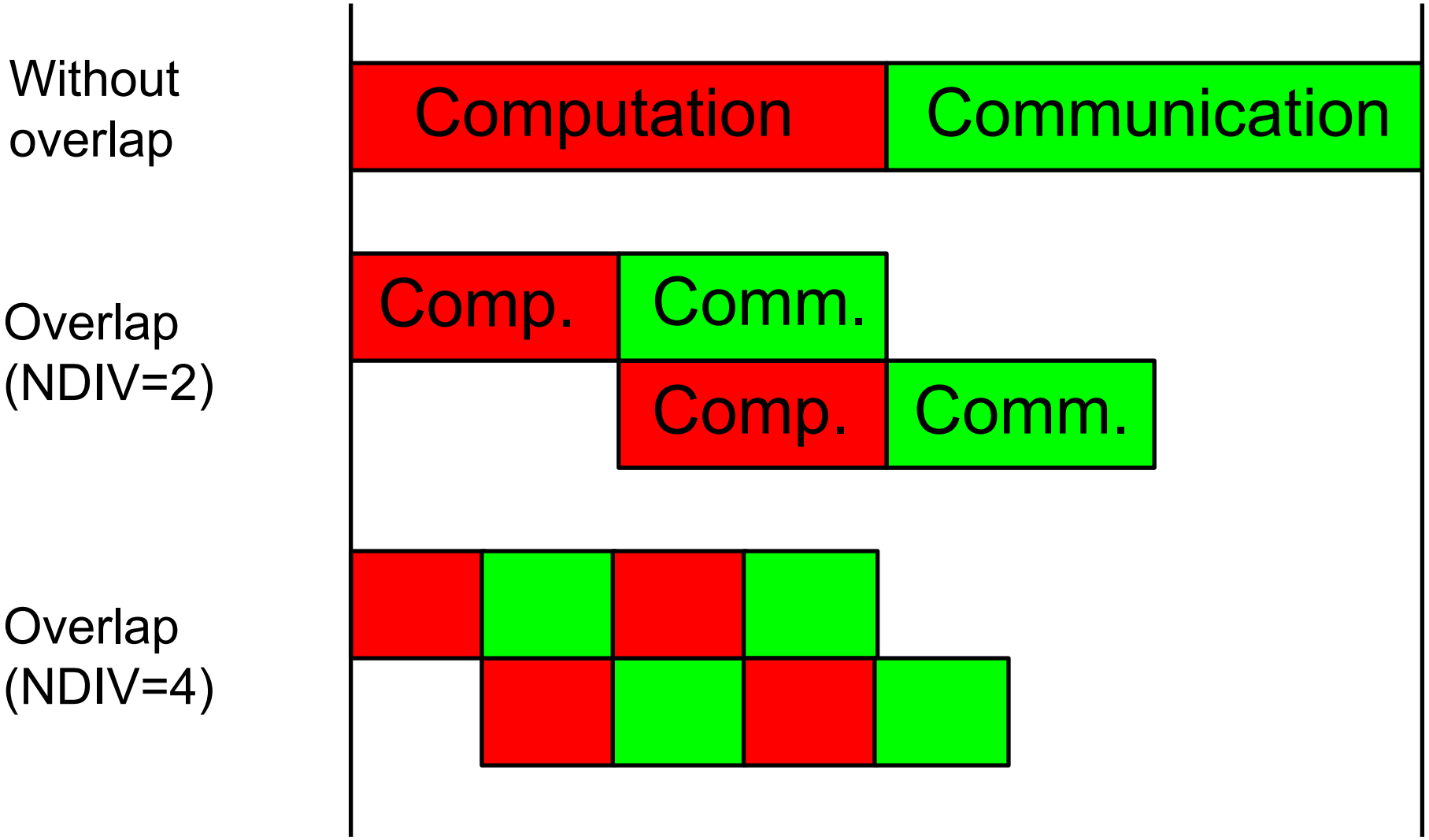
Parallel NTT Algorithm Based on Four-Step NTT



Parallelization of Four-Step NTT

```
#pragma acc data present(a[0:n1*nn2], b[0:n1*nn2], c[0:n1*nn2], d[0:n1*nn2],
w[0:n1*nn2]) {
/* Step 1: rearrange (n1 / nproc) * nproc * (n2 / nproc)
                to (n1 / nproc) * (n2 / nproc) * nproc */
#pragma acc parallel loop collapse(3)
  for (k = 0; k < nproc; k++)
    for (j = 0; j < nn2; j++)
      for (i = 0; i < nn1 / ndiv; i++)
        b[i + j * (nn1 / ndiv) + k * ((nn1 / ndiv) * nn2)]
        = a[i + k * nn1 + j * (nn1 * nproc)];
/* Step 2: all-to-all communication */
  MPI_Ialltoall(b, (nn1 * nn2) / ndiv, MPI_UNSIGNED_LONG_LONG, c,
            (nn1 * nn2) / ndiv, MPI_UNSIGNED_LONG_LONG, MPI_COMM_WORLD);
/* Step 3: (n1 / nproc) simultaneous n2-point multirow NTTs */
  multirow_ntt(c, d, nn1, n2, omega, p, mu);
…
```

Pipelined Computation-Communication Overlap



Automatic Tuning of Parallel NTTs on GPU Clusters

- The automatic tuning process consists of three steps:
  - Selection of the number of divisions NDIV for the computation-communication overlap
  - Selection of the radices (N_1 and N_2)

Selection of Number of Divisions for Computation-Communication Overlap

- When the number of divisions for computation-communication overlap is increased, the overlap ratio also increases.
- On the other hand, the performance of all-to-all communication decreases due to reducing the message size.
- Thus, a tradeoff exists between the overlap ratio and the performance of all-to-all communication.
- In our implementation, the overlapping parameter NDIV is varied between 1 (without overlap), 2, 4, 8, and 16.

Selection of the Radices

- If the condition of $N = N_1 \times N_2$ is satisfied, then we can select the arbitrary N_1 and N_2, where $N_1, N_2 \geq P$.
- We need to select the best combination and order of N_1 and N_2 for computing parallel NTT.
- If N and P are a power of two, N_1 is varied with $P, 2P, \ldots, N$, then $N_2 = N / N_1$.
- In this case, the size of search space is $\log_2(N/P)$.

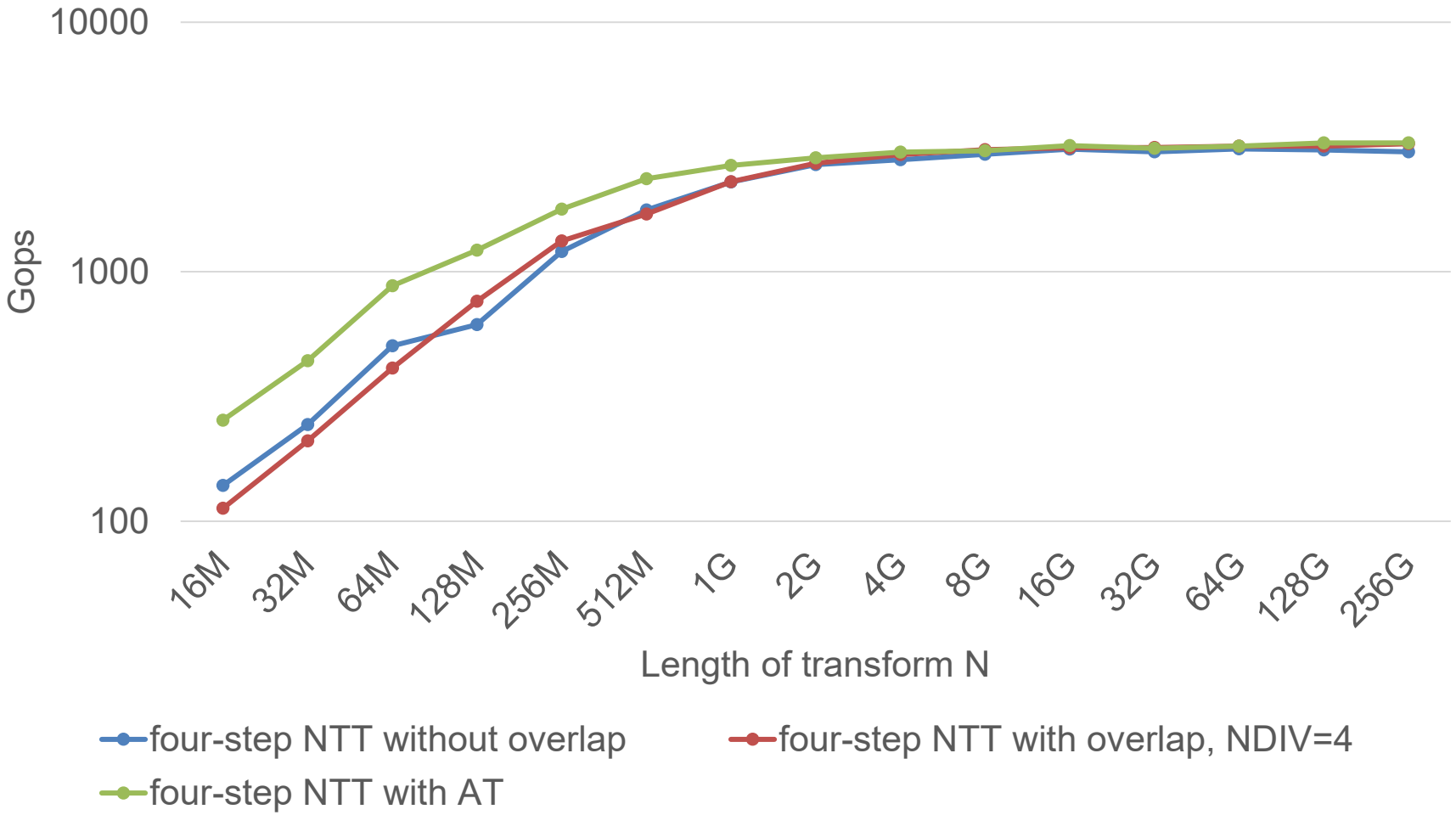
Performance Results

- For performance evaluation, we compared the performance of the following parallel NTT implementations with a modulus of 63 bits:
  - GPU implementation of the four-step NTT (without overlap)
  - GPU implementation of the four-step NTT (with overlap, NDIV=4)
  - GPU implementation of the four-step NTT (with automatic tuning)
- The giga-operations per second (Gops) values are each based on $(3/2)N \log_2 N$ for a transform of size $N = 2^m$.

Evaluation Environment

- The performance was measured on the Miyabi-G, a GPU cluster at Joint Center for Advanced HPC (JCAHPC).
  - 1120 nodes, Peak 78.8 PFlops
  - CPU：Nvidia Grace (72-core, 3.0 GHz, 3.456 TFlops)
  - GPU: NVIDIA H100 (66.9 TFlops in FP64 Tensor Core)
  - Interconnect: InfiniBand NDR
  - Compiler: NVIDIA HPC Compilers 24.9
  - MPI library: OpenMPI 4.1.7a1
  - Compiler option:
    -fast -acc=gpu -gpu=cc90,mem:separate:pinnedalloc
- In the experiment, we used 256 nodes.
- Each node has 1 MPI process.

Performance of Parallel NTTs (Miyabi-G, 256 nodes)



Results of Automatic Tuning of Parallel NTTs (Miyabi-G, 256 nodes)

	Four-step NTT with overlap				Four-step NTT with AT			
N	N1	N2	NDIV	Gops	N1	N2	NDIV	Gops
64M	8K	8K	4	411.3	4K	16K	1	877.8
256M	16K	16K	4	1327.3	4K	64K	1	1782.3
1G	32K	32K	4	2295.3	16K	64K	1	2668.0
4G	64K	64K	4	2946.4	1K	4M	2	3017.9
16G	128K	128K	4	3136.7	1K	16M	2	3204.7
64G	256K	256K	4	3185.2	32K	2M	16	3183.8
256G	512K	512K	4	3257.4	256K	1M	16	3282.2

Discussion

- On the four-step NTT (with overlap, NDIV=4), the communication message size is always divided into four parts, and when the communication message size is small, the all-to-all communication performance is low.
- Therefore, when the problem size is small, the performance of the four-step NTT (with overlap, NDIV=4) is lower than the four-step NTT (without overlap).
- As a result of automatic tuning, the optimal number of divisions for the communication message size increases as the problem size increases.

Conclusion

- We proposed an implementation of a parallel NTT with automatic tuning on GPU clusters.
- An automatic tuning facility for selecting the optimal parameters of the computation-communication overlap, and the radices was implemented.
- The performance results demonstrate that the proposed implementation of parallel NTT with automatic tuning is efficient for improving the performance on GPU clusters.