Implementation of Parallel Number-Theoretic Transform on Manycore Clusters

Daisuke Takahashi
Center for Computational Sciences
University of Tsukuba, Japan

Outline

- Background
- Objectives
- Number-theoretic Transform (NTT)
- Six-Step NTT Algorithm
- Parallel Implementation of NTT
- Performance Results
- Conclusion

Background

- The fast Fourier transform (FFT) is an algorithm that is widely used today in scientific and engineering computing.
- FFTs are often computed using complex or real numbers, but it is known that these transforms can also be computed in a ring and a finite field [Pollard 1971].
- Such a transform is called the number-theoretic transform (NTT).
- The NTT is used for homomorphic encryption, polynomial multiplication, and multiple-precision multiplication.

Related Works (1/2)

- The number theory library (NTL) [Shoup et al.] is a C++ library for performing number-theoretic computations and implements NTT.
  - Although the NTL is thread-safe, the parallel NTT is not supported.
- Spiral-generated modular FFTs have been proposed [Meng et al. 2010 and 2013].
  - Experiments were performed using 32-bit integers and 16-bit primes with Intel SSE4 instructions.

Related Works (2/2)

- An implementation of NTT using the Intel AVX-512IFMA (Integer Fused Multiply-Add) instructions has been proposed [Boemer et al. 2021].
  - This implementation is available as the Intel Homomorphic Encryption (HE) Acceleration Library.
  - Intel HEXL targets the typical data size $n = [2^{10}, 2^{17}]$ of NTTs used in homomorphic encryption and is not parallelized.
- An Implementation of Parallel Number-Theoretic Transform Using Intel AVX-512 Instructions has been proposed [Takahashi 2022].
  - NTT kernels are vectorized using the Intel AVX-512 instructions.
  - Six-step NTT is parallelized using OpenMP.

Objectives

- We consider accelerating NTT for larger data sizes by parallelization, targeting polynomial multiplication and multiple-precision multiplication.
- We parallelize the six-step NTT using MPI and OpenMP.

Number-Theoretic Transform (NTT)

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

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

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

Stockham Radix-2 NTT Algorithm

Algorithm 1 Stockham radix-2 NTT algorithm

Input: $n = 2^q,\ X_0(j) = x(j),\ 0 \le j \le n-1$, and ω_n is the primitive n-th root of unity.

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

$l \leftarrow n/2$
$m \leftarrow 1$
for t **from** 1 **to** q **do**
    for j **from** 0 **to** $l-1$ **do**
        for k **from** 0 **to** $m-1$ **do**
            $c_0 \leftarrow X_{t-1}(k + jm)$
            $c_1 \leftarrow X_{t-1}(k + jm + lm)$
            $X_t(k + 2jm) \leftarrow (c_0 + c_1) \bmod p$
            $X_t(k + 2jm + m) \leftarrow \omega_n^{jm}(c_0 - c_1) \bmod p$
        end for
    end for
    $l \leftarrow l/2$
    $m \leftarrow 2m$
end for

Modular Arithmetic in NTT

- The butterfly operation of the NTT can be performed using modular addition, subtraction, and multiplication.
- The modular addition $c = (a + b) \bmod N$ for $0 \leq a, b < N$ can be replaced by the addition $c = a + b$ and the conditional subtraction $c - N$ when $c \geq N$.
- Modular multiplication includes modulo operations, which are slow due to the integer division process.
- However, Montgomery multiplication [Montgomery 1985] and Shoup's modular multiplication [Harvey 2014] are known to avoid this problem.

Shoup's Modular Multiplication Algorithm [Harvey 2014]

Algorithm 2 Shoup's modular multiplication algorithm

Input: A, B, N such that $0 \leq A, B < N$, $N < \beta/2$
precomputed $B' = \lfloor B\beta/N \rfloor$

Output: $C = AB \bmod N$

1: $q \leftarrow \lfloor AB'/\beta \rfloor$
2: $C \leftarrow (AB - qN) \bmod \beta$
3: **if** $C \geq N$ **then**
4: $\quad C \leftarrow C - N$
5: **return** C.

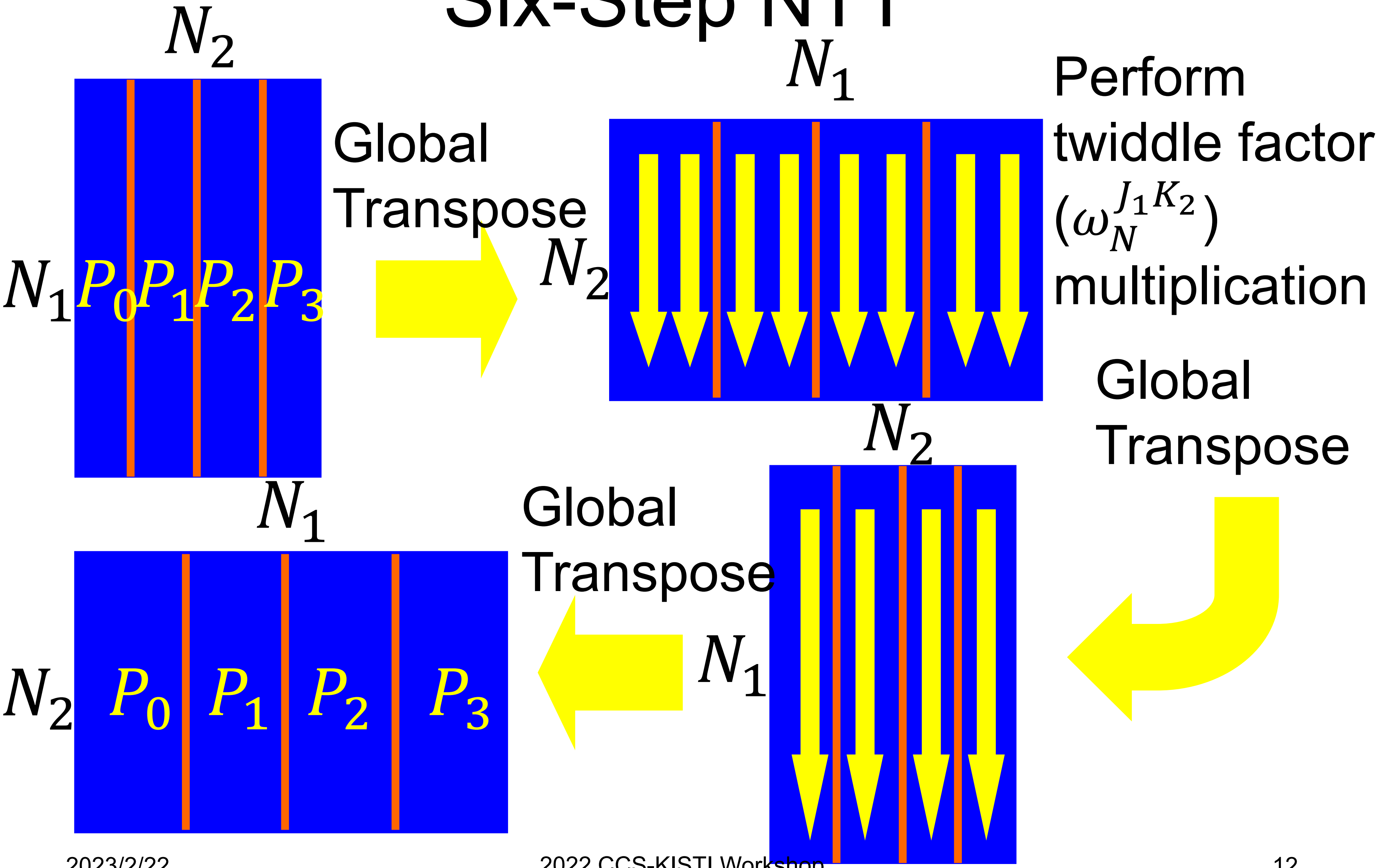
The upper half of AB'

Subtraction of the lower half of qN from the lower half of AB

Six-Step NTT Algorithm

- If n has factors n_1 and n_2 $(n = n_1 \times n_2)$, in the same way as the six-step FFT algorithm [Bailey 1990], the following six-step NTT algorithm [Takahashi 2022] is derived:
- Step 1: Transposition
- Step 2: n_1 individual n_2-point multicolumn NTTs
- Step 3: Twiddle factor ($\omega_n^{j_1 k_2}$) multiplication
- Step 4: Transposition
- Step 5: n_2 individual n_1-point multicolumn NTTs
- Step 6: Transposition

Parallel NTT Algorithm Based on Six-Step NTT



Parallelization of Six-Step NTT

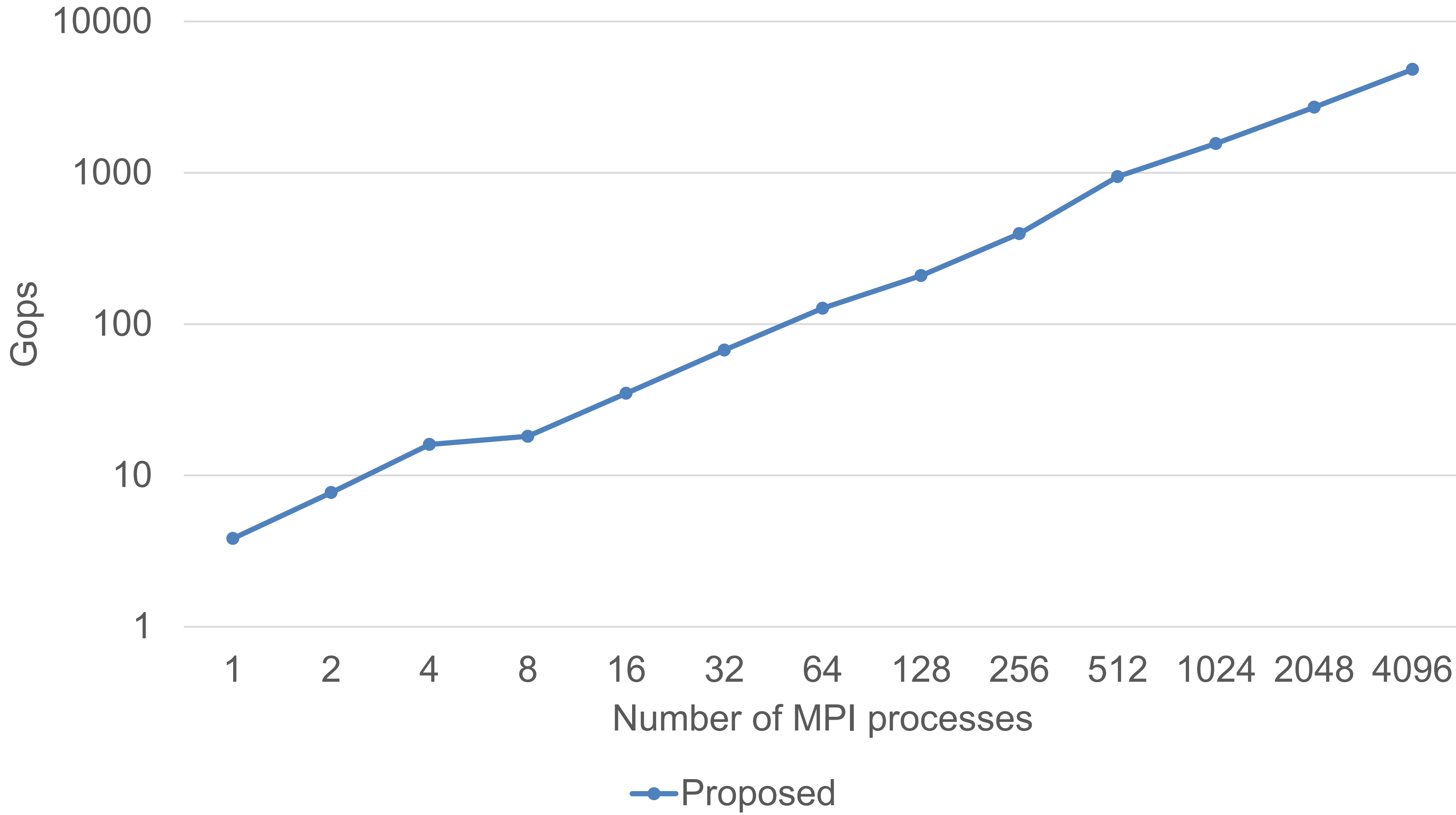
```
/* Step 1: transpose nx*ny to ny*nx */
#pragma omp parallel for collapse(2) private(i, k)
  for (j = 0; j < nny; j++)
    for (k = 0; k < nproc; k++)
      for (i = 0; i < nnx; i++)
        b[i + j * nnx + k * (nnx * nny)] = a[i + k * nnx + j * (nnx * nproc)];
  MPI_Alltoall(b, nn / nproc, MPI_UNSIGNED_LONG_LONG, a, nn / nproc,
               MPI_UNSIGNED_LONG_LONG, MPI_COMM_WORLD);
  trans(a, b, nnx, ny);

/* Step 2: (nx / nproc) individual ny-point multicolumn NTTs */
#pragma omp parallel for
  for (j = 0; j < nnx; j++)
    nttsub(&b[j * ny], &a[j * ny], wy, wwy, ny, ipy, np);
…
```

Performance Results

- For performance evaluation, we measure the performance of the proposed implementation of the six-step NTT with a modulus of 63 bits.
- The performance was measured on the Fujitsu PRIMEHPC FX1000 at the University of Tokyo.
  - 7680 nodes, Peak 25.9 PFlops
  - CPU: A64FX (48 cores + assistant 2 or 4 cores, 2.2 GHz)
  - Interconnect: Tofu interconnect D (Link bandwidth 6.8 GB/s)
  - Compiler: Fujitsu C/C++ Compiler 4.8.1
  - Compiler option: “-Nclang -Kfast -Kopenmp”
- Each MPI process has 12 cores and 12 threads, i.e., 4 MPI processes per node.
- The giga-operations per second (Gops) values are each based on $(3/2)N \log_2 N$ for a transform of size $N = 2^m$.

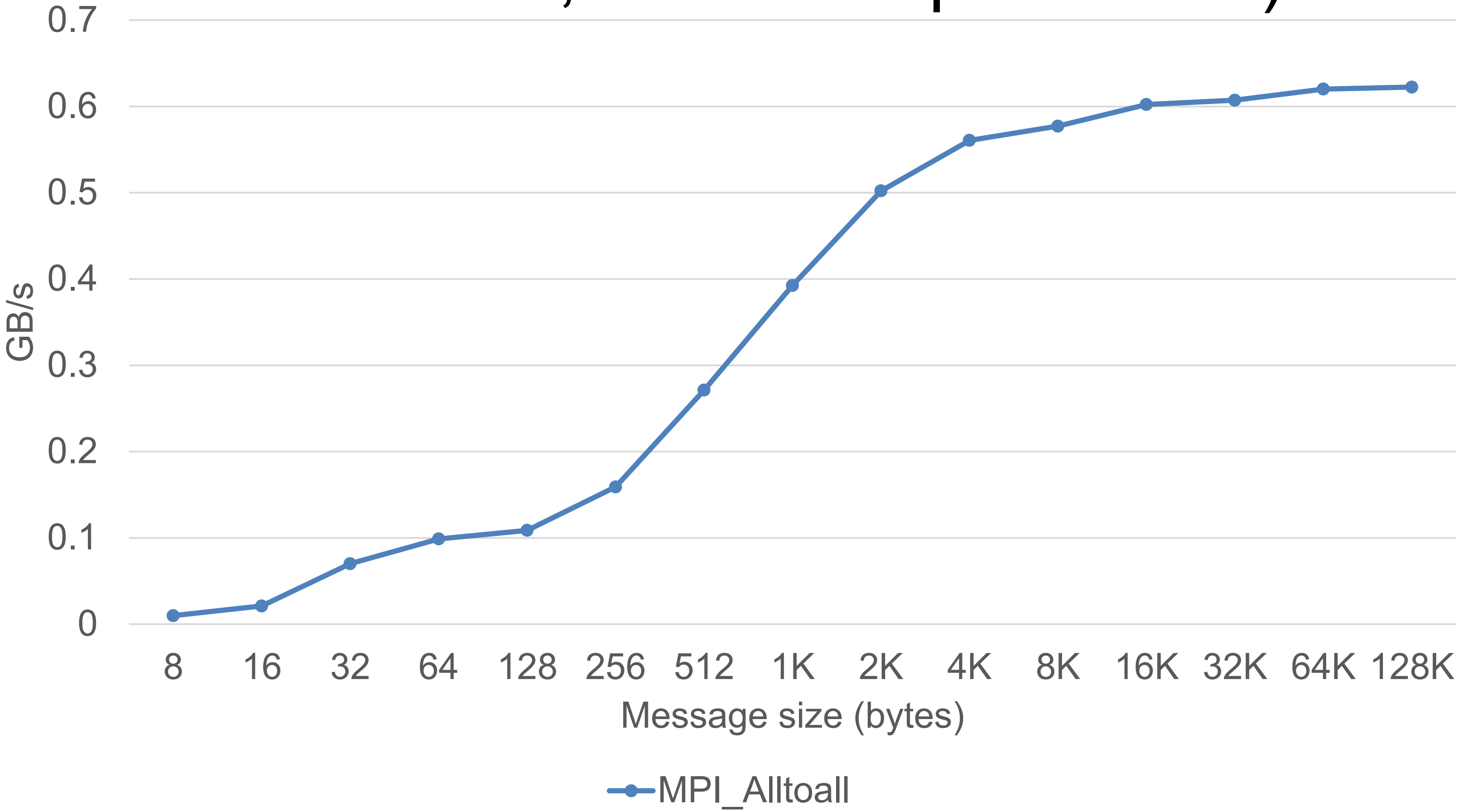
Performance of Parallel NTTs ($N = 2^{26} \times$ MPI processes)



Discussion

- The reason for the smaller performance growth when the number of MPI processes is increased from 4 to 8 is that up to 4 MPI processes are communicating within the node.
- For 2^{38}-point NTT on 4096 MPI processes, approximately 80% of the execution time is taken up by all-to-all communication.
- The Fujitsu PRIMEHPC FX1000 uses Tofu interconnect D, a 6-dimensional torus network, but as the number of nodes increases, the maximum number of hops also increases, resulting in lower all-to-all communication bandwidth.

Performance of MPI_Alltoall (Fujitsu PRIMEHPC FX1000, 1024 nodes, 4096 MPI processes)



Conclusion

- We proposed an implementation of the parallel NTT on manycore clusters.
- The butterfly operation of the NTT can be performed using modular addition, subtraction, and multiplication.
- We parallelized the six-step NTT using MPI and OpenMP.
- We successfully achieved a performance of over 4831 Gops on a Fujitsu Supercomputer PRIMEHPC FX1000 (1024 nodes) for a 2^{38}-point NTT with a modulus of 63 bits.