Implementation of Parallel FFTs on Cluster of Intel Xeon Phi Processors

Daisuke Takahashi

Center for Computational Sciences
University of Tsukuba, Japan

Outline

- Background
- Objectives
- Six-Step FFT Algorithm
- In-Cache FFT Algorithm and Vectorization
- Computation-Communication Overlap
- Automatic Tuning of Parallel 1-D FFT
- Performance Results
- Conclusion

Background

- The fast Fourier transform (FFT) is widely used in science and engineering.
- Parallel FFTs on distributed-memory parallel computers require intensive all-to-all communication, which affects their performance.
- How to overlap the computation and the all-to-all communication is an issue that needs to be addressed for parallel FFTs.
- Moreover, we need to select the optimal parameters according to the computational environment and the problem size.

Objectives

- Several FFT libraries with automatic tuning have been proposed.
  - FFTW, SPIRAL, and UHFFT
- An Implementation of parallel 1-D FFT on Xeon Phi (Knights Corner) cluster has been presented [Park et al. 2013].
- However, to the best of our knowledge, parallel 1-D FFT with automatic tuning on Xeon Phi (Knights Landing) cluster has not yet been reported.
- We propose an implementation of a parallel 1-D FFT with automatic tuning on Xeon Phi cluster.

Approach

- The parallel 1-D FFT implemented is based on the six-step FFT algorithm [Bailey 90], which requires two multicolumn FFTs and three data transpositions.
- Using this method, we have implemented an automatic tuning facility for selecting the optimal parameters of the computation-communication overlap, the radices, and the block size.

Discrete Fourier Transform (DFT)

- 1-D discrete Fourier transform (DFT) is given by

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

where $\omega_n = e^{-2\pi i/n}$ and $i = \sqrt{-1}$.

2-D Formulation

- If n has factors n_1 and n_2 ($n = n_1 \times n_2$), then the indices j and k can be expressed as

$$j = j_1 + j_2 n_1, \quad k = k_2 + k_1 n_2 .$$

- Substituting the indices j and k, we derive the following equation:

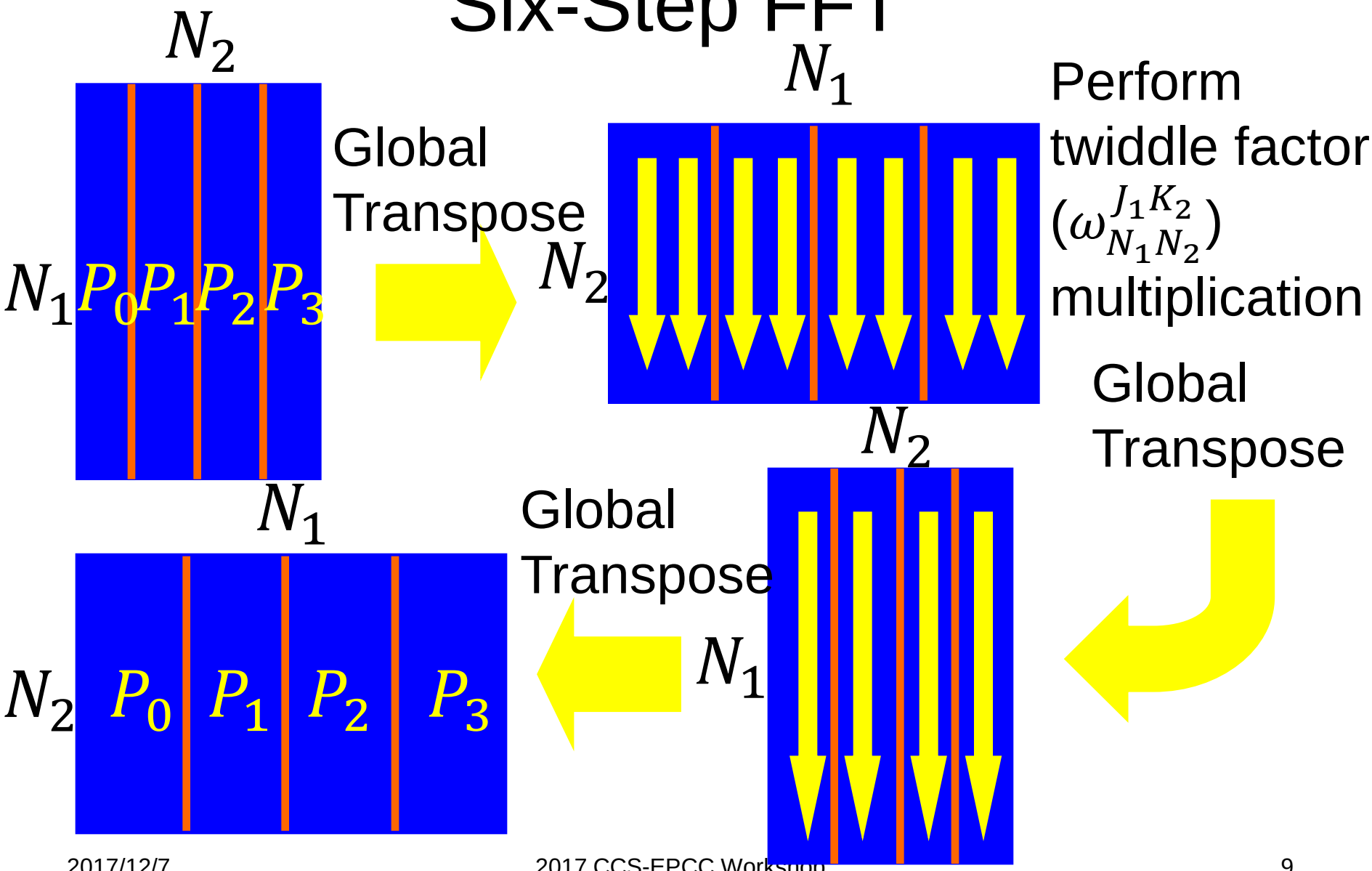
$$y(k_2, k_1) = \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} x(j_1, j_2)\, \omega_{n_2}^{j_2 k_2}\, \omega_{n_1 n_2}^{j_1 k_2}\, \omega_{n_1}^{j_1 k_1} .$$

- An n-point FFT can be decomposed into an n_1-point FFT and an n_2 -point FFT.

Six-Step FFT Algorithm

- This derivation leads to the following six-step FFT algorithm [Bailey90]:
  - Step 1: Transpose
  - Step 2: Perform n_1 individual n_2-point multicolumn FFTs
  - Step 3: Perform twiddle factor ($\omega_{n_1 n_2}^{j_1 k_2}$) multiplication
  - Step 4: Transpose
  - Step 5: Perform n_2 individual n_1-point multicolumn FFTs
  - Step 6: Transpose

Parallel 1-D FFT Algorithm Based on Six-Step FFT



In-Cache FFT Algorithm and Vectorization

- For in-cache FFT, we used radix-2, 3, 4, 5, 8, 9, and 16 FFT algorithms based on the mixed-radix FFT algorithms [Temperton 83].
- Automatic vectorization was used to access the Intel AVX-512 instructions on the Knights Landing processor.
- Although higher radix FFTs require more floating-point registers to hold intermediate results, the Knights Landing processor has 32 ZMM 512-bit registers.

Optimization of Parallel 1-D FFT on Knights Landing Processor

```
      COMPLEX*16 X(N1,N2),Y(N2,N1)
!$OMP PARALLEL DO COLLAPSE(2) PRIVATE(I,J,JJ)
      DO II=1,N1,NB
         DO JJ=1,N2,NB
            DO I=II,MIN(II+NB-1,N1)
               DO J=JJ,MIN(JJ+NB-1,N2)
                  Y(J,I)=X(I,J)
               END DO
            END DO
         END DO
      END DO
!$OMP PARALLEL DO
      DO I=1,N1
         CALL IN_CACHE_FFT(Y(1,I),N2)
      END DO
      …
```

To expand the outermost loop, the double-nested loop can be collapsed into a single-nested loop.

Computation-Communication Overlap [Idomura et al. 2014]

```
!$OMP PARALLEL
!$OMP MASTER
      MPI communication
!$OMP END MASTER
!$OMP DO SCHEDULE(DYNAMIC)
      DO I=1,N
         Computation
      END DO
!$OMP DO
      DO I=1,N
         Computation using the
         result of communication
      END DO
!$OMP END PARALLEL
```

← MPI communication is performed on the master thread

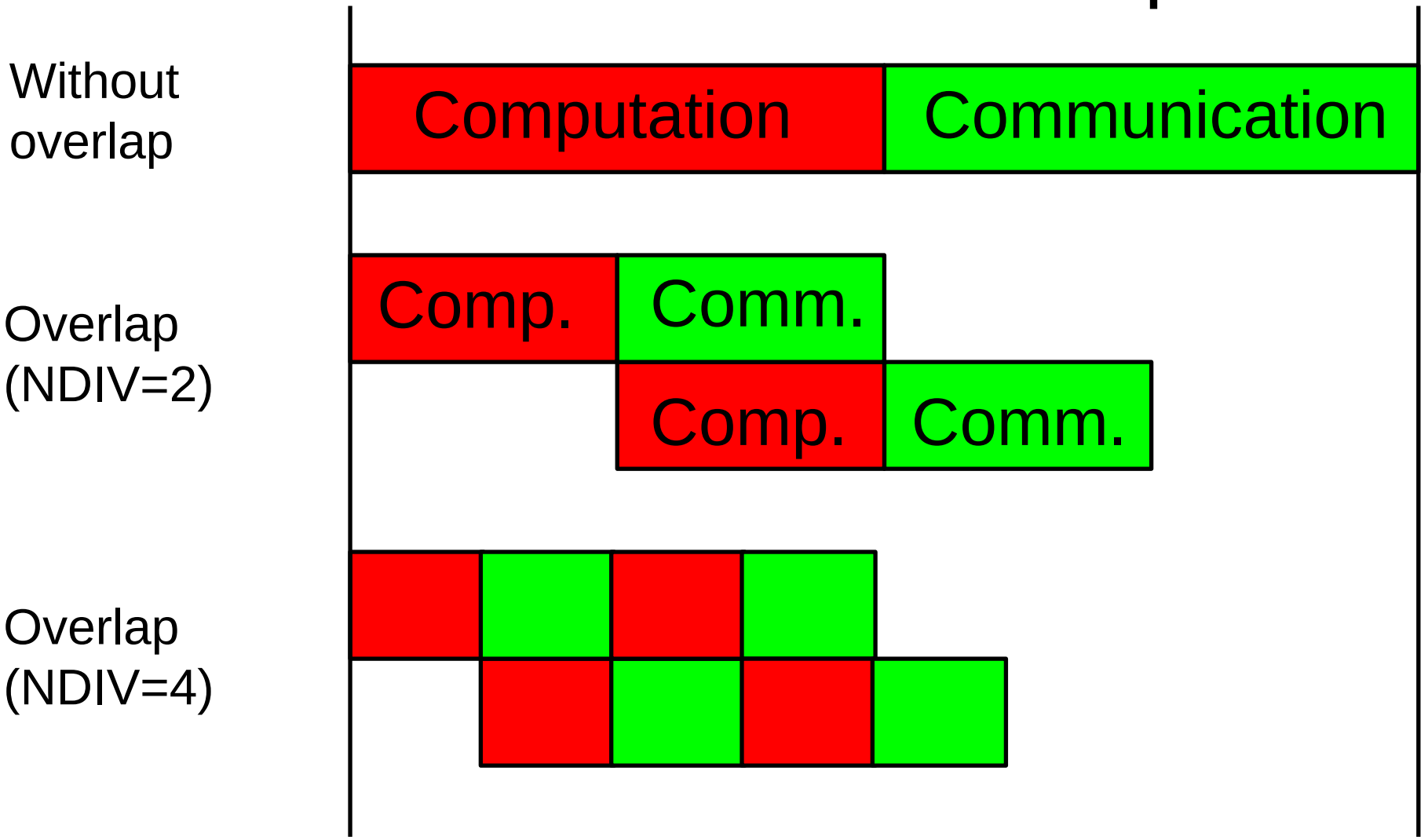
← No barrier synchronization

← Computation is performed by a thread other than the master thread

← Implicit barrier synchronization

← Computation is performed after completion of the MPI communication

Pipelined Computation-Communication Overlap



Automatic Tuning of Parallel 1-D FFT

- 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 the block size NB

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.
- The default overlapping parameter of the original FFTE 6.2alpha is NDIV=4.
- In our implementation, the overlapping parameter NDIV is varied between 1, 2, 4, 8 and 16.

Selection of Radices

- As long as condition $N = N_1 N_2$ is satisfied, we can select N_1 and N_2 arbitrarily for $N_1, N_2 \geq P$.
- We need to select the best combination and order of N_1 and N_2 for computing a parallel 1-D FFT.
- If N and P are a power of two, N_1 is varied between $P, 2P, \ldots, \sqrt{N}$, then $N_2 = N / N_1$.
- In this case, the size of search space is $\log_2(\sqrt{N}/P)$.

Selection of Block Size

- The default blocking parameter of the original FFTE 6.2alpha is NB=32.
- Although the optimal block size may depend on the problem size, the block size NB can also be varied.
- In our implementation, the block size NB is varied between 4, 8, 16, 32 and 64.

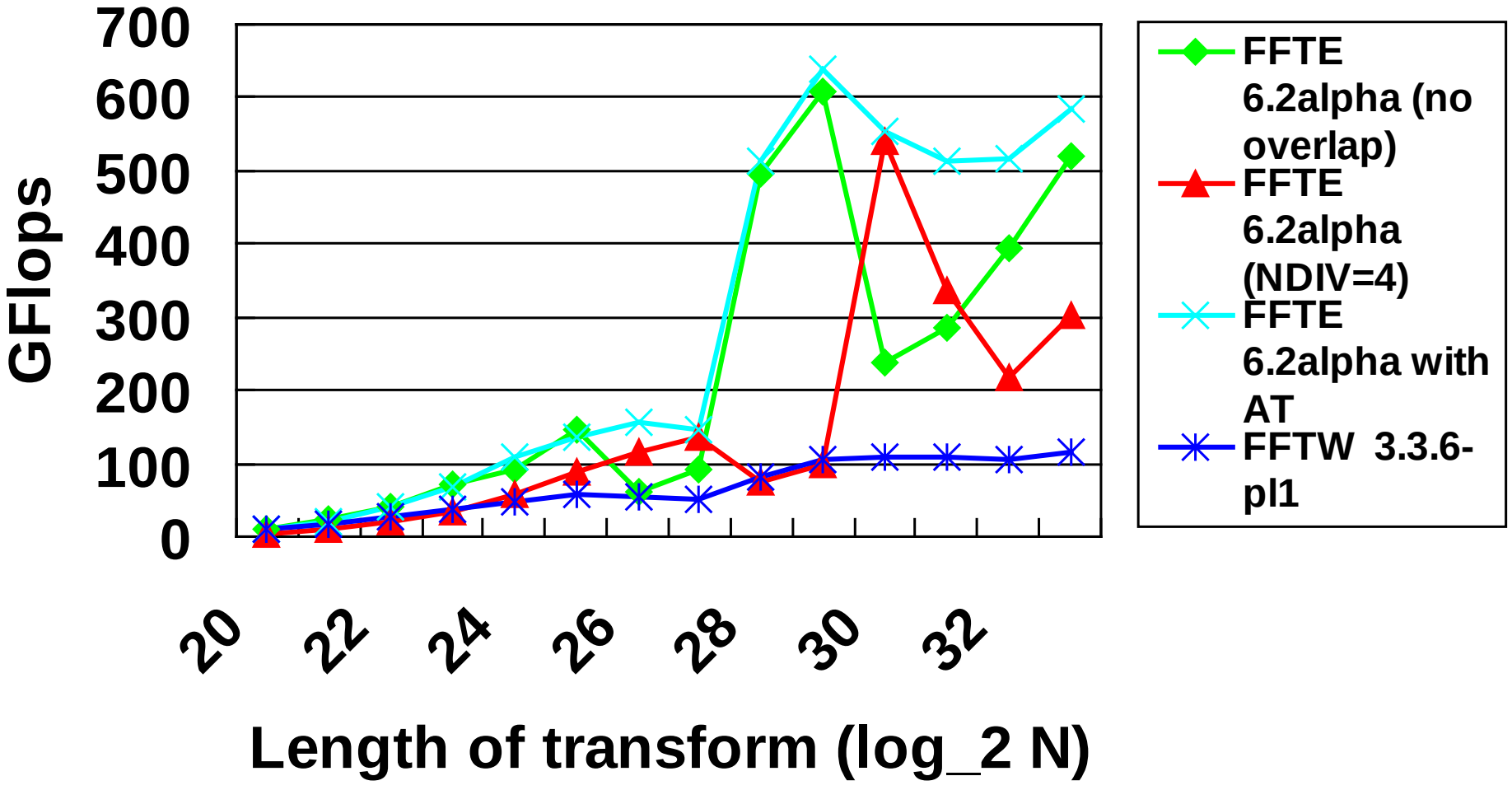
Performance Results

- To evaluate the parallel 1-D FFT with automatic tuning (AT), we compared its performance with that of the FFTW 3.3.6-pl1, the FFTE 6.2alpha (http://www.ffte.jp/) and the FFTE 6.2alpha with AT.
- The performance was measured on the Oakforest-PACS at Joint Center for Advanced HPC (JCAHPC).
  - 8208 nodes, Peak 25.008 PFlops
  - CPU: Intel Xeon Phi 7250 (68 cores, Knights Landing 1.4 GHz)
  - Interconnect: Intel Omni-Path Architecture
  - Compiler: Intel Fortran compiler 17.0.1.132
  - Compiler option: “mpiifort -O3 -xMIC-AVX512 -qopenmp”
  - MPI library: Intel MPI 2017.1.132
  - flat/quadrant, MCDRAM only, KMP_AFFINITY=compact
  - Each MPI process has 64 cores and 64 threads.

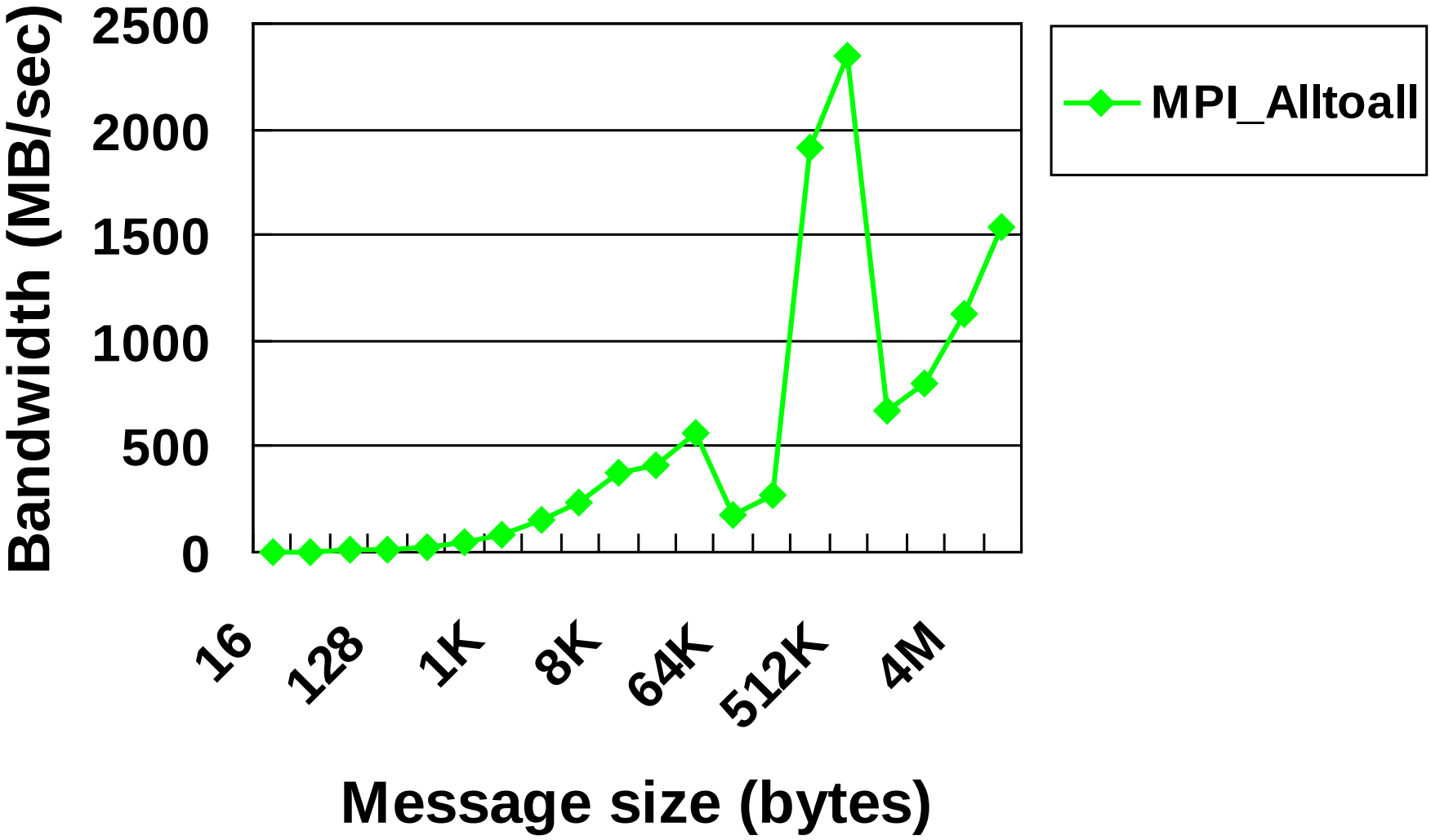
Results of automatic tuning of parallel 1-D FFTs (Oakforest-PACS, 128 nodes)

	FFTE 6.2alpha					FFTE 6.2alpha with AT				
N	N1	N2	NB	NDIV	GFlops	N1	N2	NB	NDIV	GFlops
16M	4K	4K	32	4	57.8	4K	4K	32	1	109.4
64M	8K	8K	32	4	116.9	8K	8K	16	2	154.8
256M	16K	16K	32	4	73.3	8K	32K	16	1	513.8
1G	32K	32K	32	4	541.7	32K	32K	64	4	554.9
4G	64K	64K	32	4	217.0	64K	64K	32	16	516.5

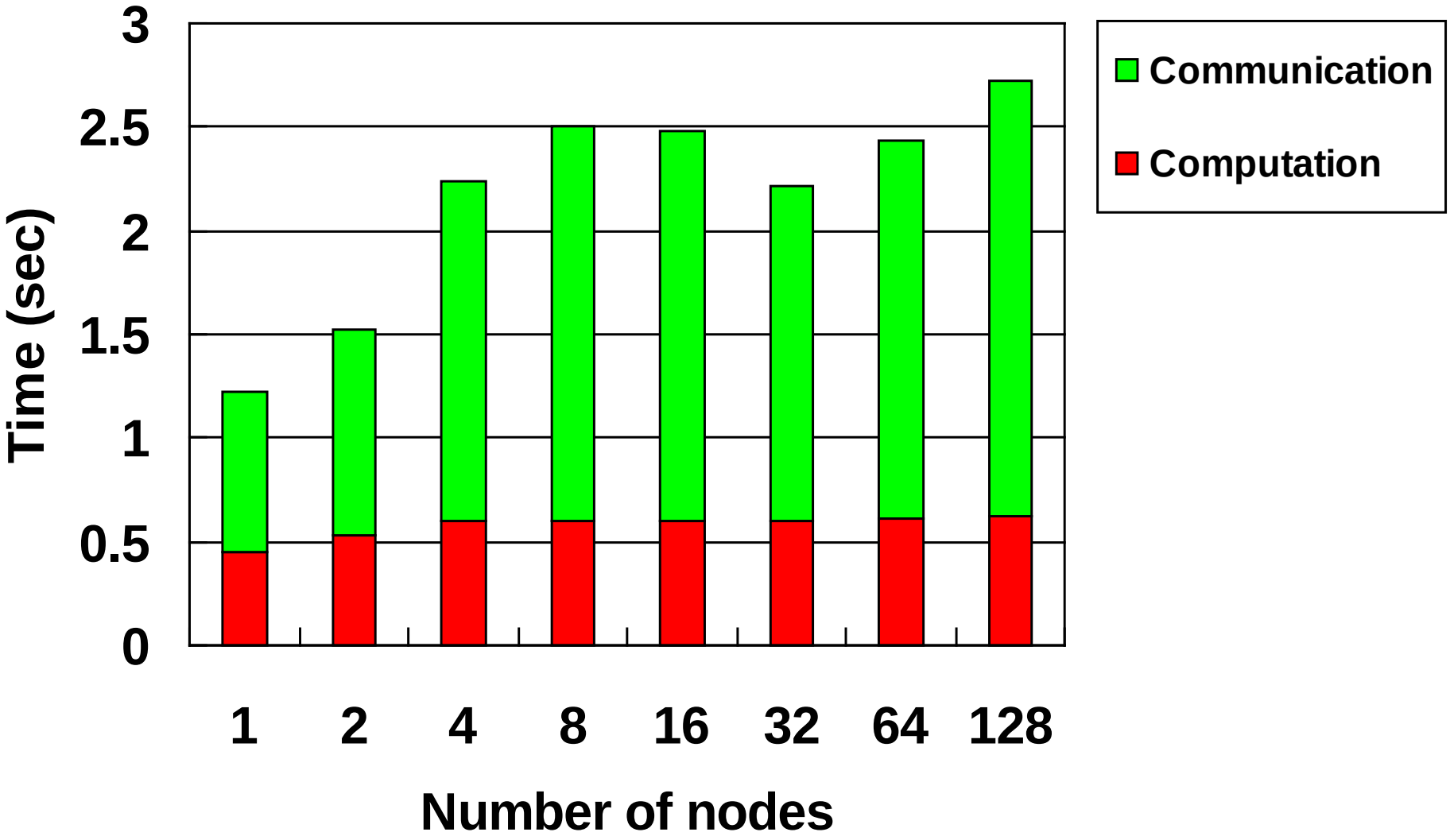
Performance of parallel 1-D FFTs
（Oakforest-PACS，128 nodes）



Performance of all-to-all communication （Oakforest-PACS，128 nodes）



Breakdown of execution time in FFTE 6.2alpha (no overlap, Oakforest-PACS, N=2^26×number of nodes)



Conclusion

- We proposed an implementation of parallel 1-D FFT with automatic tuning on Xeon Phi cluster.
- We used a computation-communication overlap method that introduces a communication thread with OpenMP.
- An automatic tuning facility for selecting the optimal parameters of the computation-communication overlap, the radices, and the block size was implemented.
- The performance results demonstrate that the proposed implementation of a parallel 1-D FFT with automatic tuning is efficient for improving the performance on Xeon Phi cluster.