MS2 and MS13
State-of-the-Art FFT --- Algorithms, Implementations, and Applications

- **Organizers:**
  - **Daisuke Takahashi**
    University of Tsukuba, Japan
  - **Franz Franchetti**
    Carnegie Mellon University, USA
  - **Samar A. Aseeri**
    King Abdullah University of Science & Technology (KAUST), Saudi Arabia
  - **Benson K. Muite**
    University of Tartu, Estonia

Aim of this minisymposium

- The fast Fourier transform (FFT) is widely used in many areas of science and engineering.
- This minisymposium is an opportunity to discuss high performance and parallel implementations of FFT, mathematical encapsulations of FFT algorithms that are amenable to automatic implementations tuned to hardware platforms.
- It is also a venue to discuss applications and performance results of FFT on current and emerging platforms such as many-core processors, GPUs, and distributed-memory systems.
- http://www.fft.report

MS2: Part I of II

- **1:00-1:20 Implementation of Parallel FFTs on Cluster of Intel Xeon Phi Processors**

  Daisuke Takahashi, University of Tsukuba, Japan
- **1:25-1:45 SPIRAL FFT**

  Franz Franchetti, Carnegie Mellon University, USA
- **1:50-2:10 Pipelining Fast Fourier Transform on the OpenPOWER Cluster**

  Jun Doi, IBM Research - Tokyo, Japan
- **2:15-2:35 Automatic FFT Kernel Generation for CUDA GPUs**

  Akira Nukada, Tokyo Institute of Technology, Japan

MS13: Part II of II

- **3:10-3:30 Fast Fourier Transforms (fft)**

  Samar A. Aseeri, King Abdullah University of Science & Technology (KAUST), Saudi Arabia

- **3:35-3:55 Use of the Fast Fourier Transform in Solving Partial Differential Equations**

  Benson K. Muite, University of Tartu, Estonia

- **4:00-4:20 Parallel Fast Gauss Transform**

  Hari Sundar, University of Utah, USA

- **4:25-4:45 Implementation of OpenFFT and Its Application to Industrial Problems**

  Truong Vinh Truong Duy, Nissan ARC, Japan

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 cluster of Intel Xeon Phi coprocessors has been presented [Park et al. 2013].
- However, to the best of our knowledge, parallel 1-D FFT with automatic tuning on cluster of Intel Xeon Phi processors has not yet been reported.
- We propose an implementation of a parallel 1-D FFT with automatic tuning on cluster of Intel Xeon Phi processors.

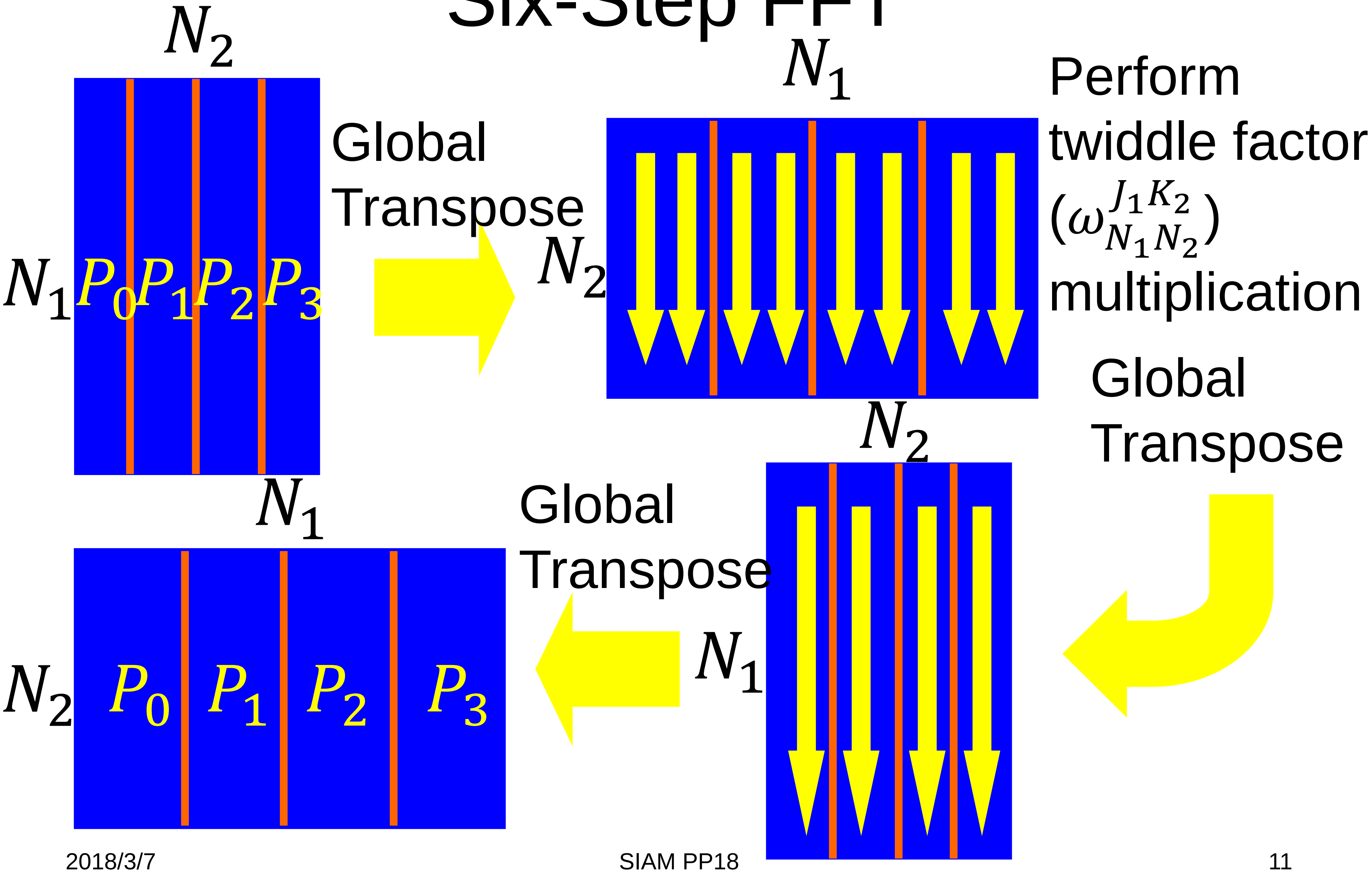
Approach

- 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 all-to-all communication and the computation-communication overlap.

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



Global Transpose

Perform twiddle factor ($\omega_{N_1 N_2}^{J_1 K_2}$) multiplication

Global Transpose

Global Transpose

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]

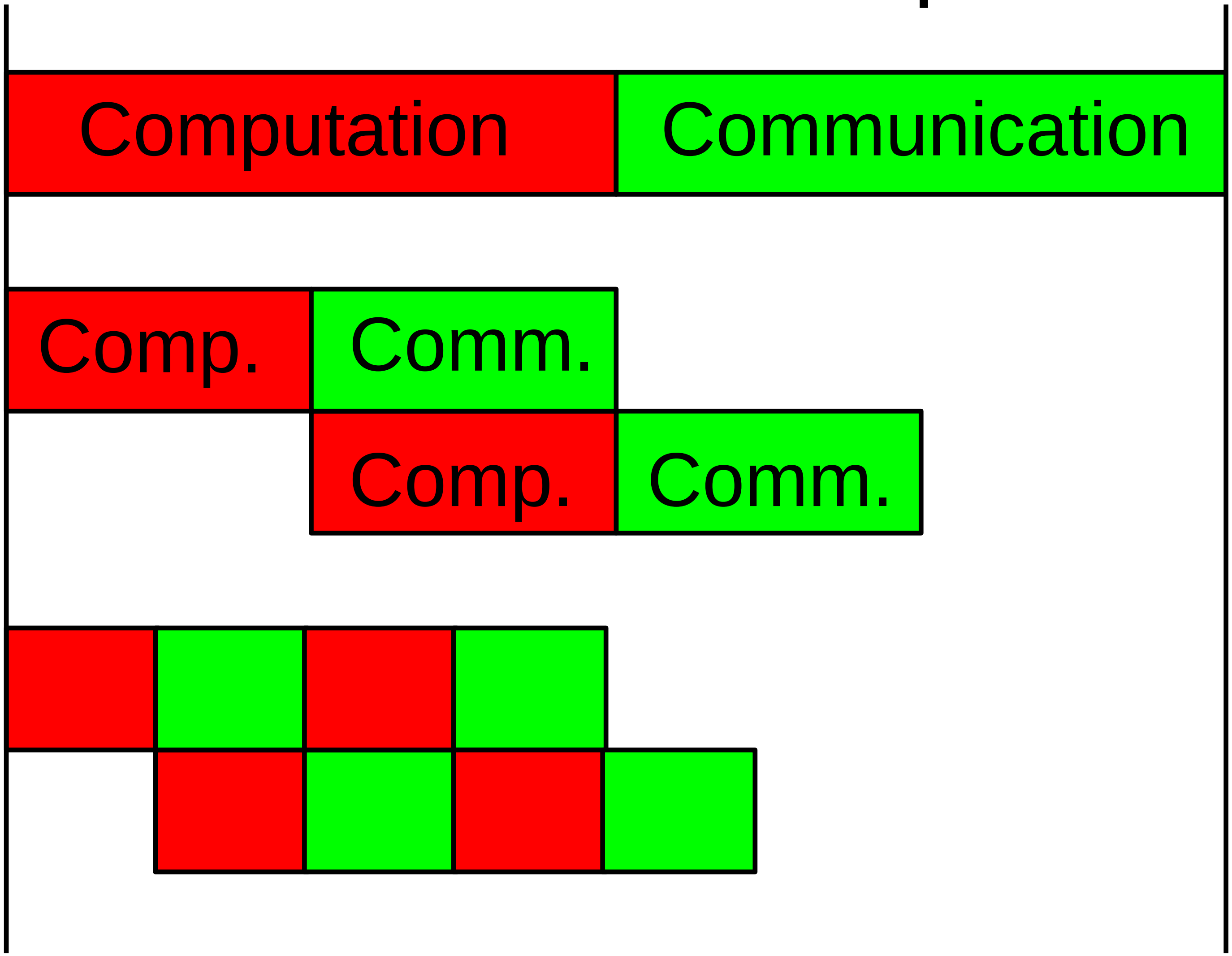
```
!$OMP PARALLEL
!$OMP MASTER
    MPI communication              ← MPI communication is performed
                                     on the master thread
!$OMP END MASTER ← No barrier synchronization
!$OMP DO SCHEDULE(DYNAMIC)
    DO I=1,N
        Computation              ← Computation is performed
                                   by a thread other than the
    END DO                         master thread
!$OMP DO ← Implicit barrier
    DO I=1,N  synchronization
        Computation using the        ← Computation is performed
        result of communication        after completion of the
    END DO                             MPI communication
!$OMP END PARALLEL
```

Pipelined Computation-Communication Overlap

Without overlap

Overlap (NDIV=2)

Overlap (NDIV=4)



Automatic Tuning of Parallel 1-D FFT

- The automatic tuning process consists of two steps:
  - Automatic tuning of all-to-all communication
  - Selection of the number of divisions NDIV for the computation-communication overlap

Optimizing of All-to-All Communication

- An optimized all-to-all collective algorithm for multi-core systems connected using modern InfiniBand network interfaces [Kumar et al. 08].
- The all-to-all algorithm completes in two steps, intra-node exchange and inter-node exchange.

Two-Phase All-to-All Algorithm

- We extend the all-to-all algorithm to the general case of $P = P_x \times P_y$ MPI processes.

1. Local array transpose from
   $(N/P^2, P_x, P_y)$ to $(N/P^2, P_y, P_x)$,
   where N is the total number of elements.

   Then P_y simultaneous all-to-all communications across P_x MPI processes are performed.
2. Local array transpose from
   $(N/P^2, P_y, P_x)$ to $(N/P^2, P_x, P_y)$.
   Then P_x simultaneous all-to-all communications across P_y MPI processes are performed.

Automatic Tuning of All-to-All Communication

- The two-phase all-to-all algorithm requires twice the total amount of communications compared with the ring algorithm.
- However, for small to medium messages, the two-phase all-to-all algorithm is better than the ring algorithm due to the smaller startup time.
- Automatic tuning of all-to-all communication can be accomplished by performing a search over the parameters of all of P_x and P_y.
- If $P = P_x \times P_y$ is a power of two, the size of search space is $\log_2 P$.

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.

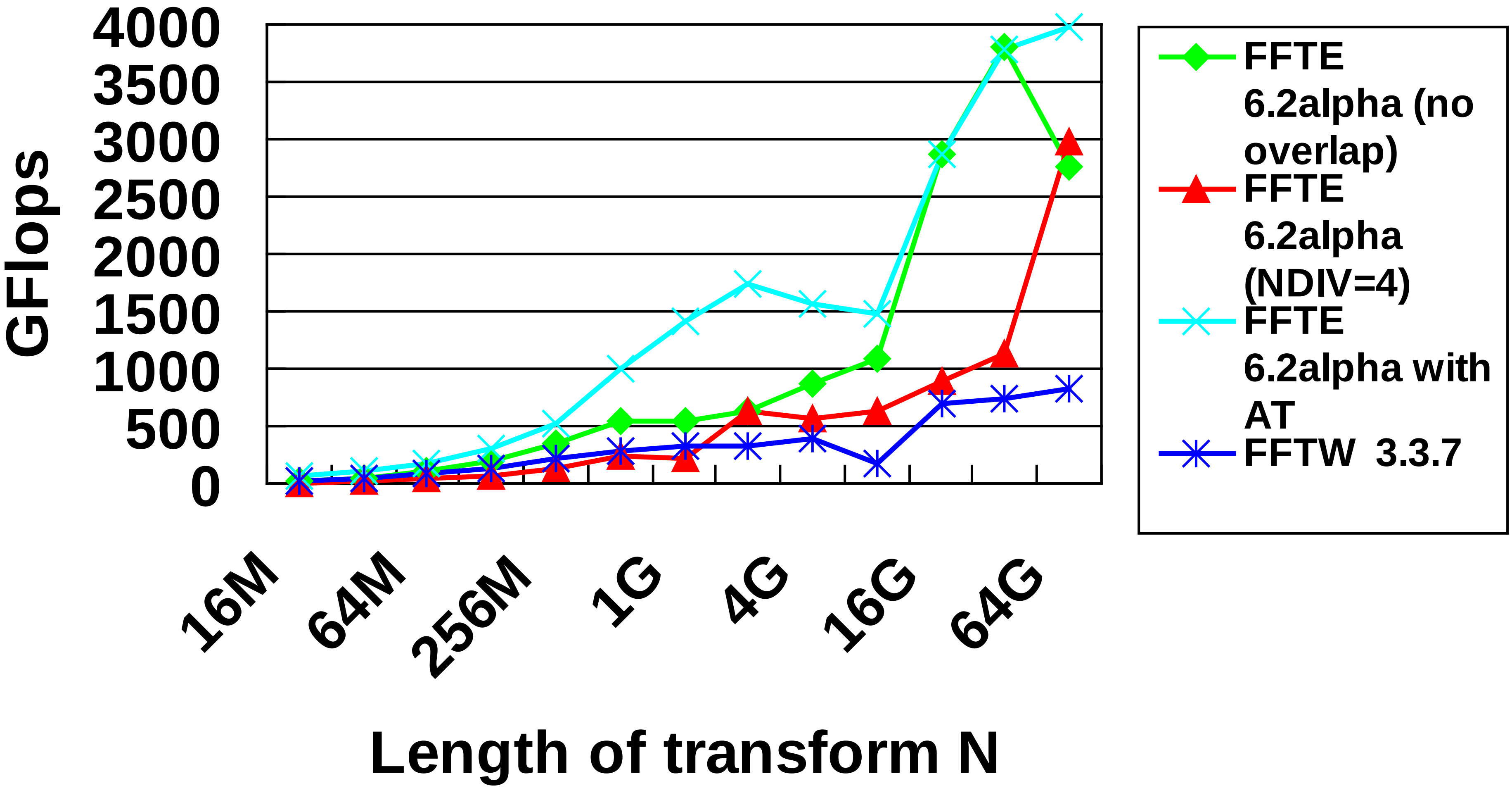
Performance Results

- To evaluate the parallel 1-D FFT with automatic tuning (AT), we compared its performance with that of the FFTW 3.3.7, 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 18.0.1.163 (for FFTE)
Intel C compiler 18.0.1.163 (for FFTW)
  - Compiler option: "-O3 -xMIC-AVX512 -qopenmp"
  - MPI library: Intel MPI 2018.1.163
  - flat/quadrant, MCDRAM only, KMP_AFFINITY=compact
  - Each MPI process has 64 cores and 64 threads.

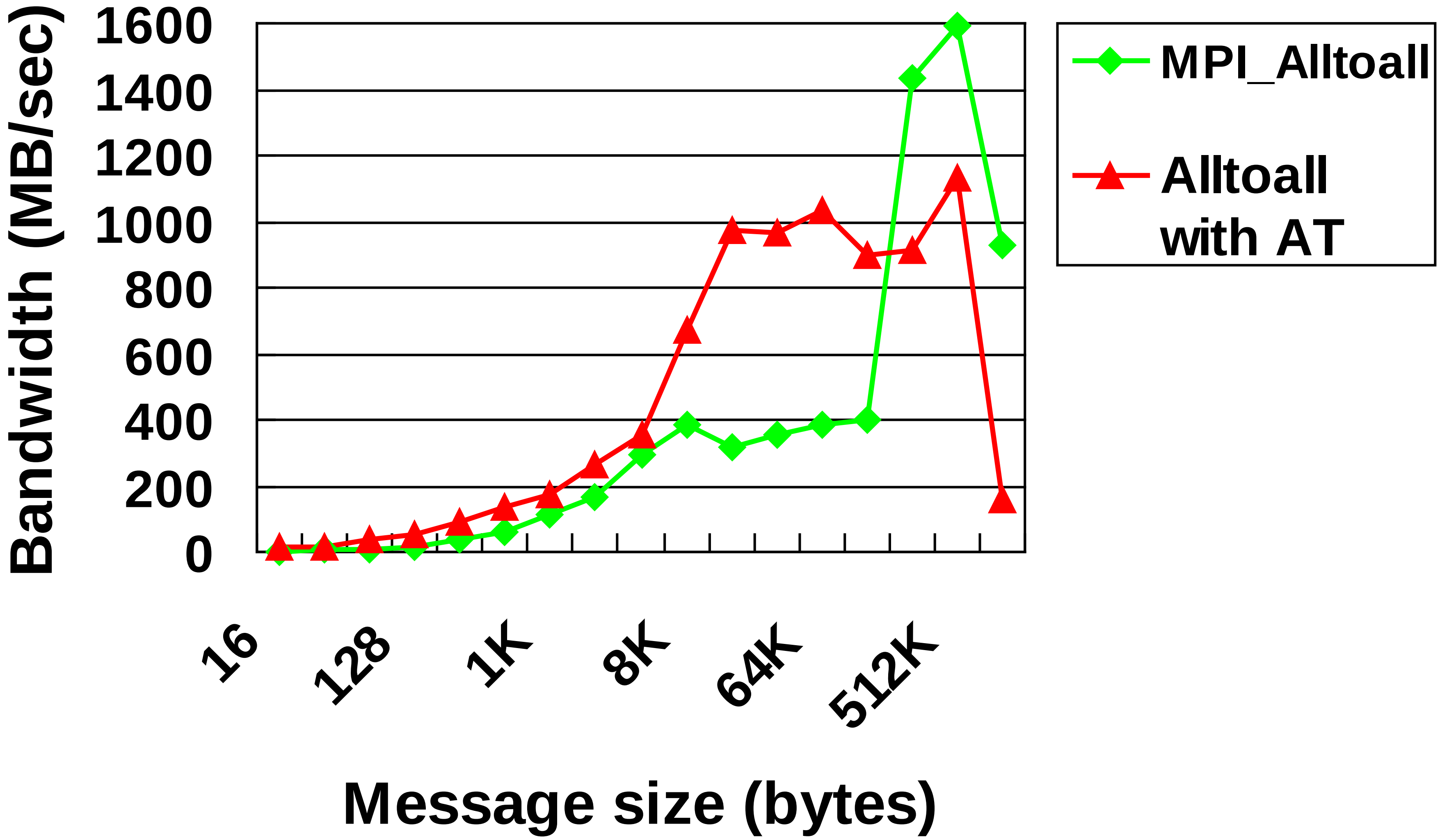
Results of automatic tuning of parallel 1-D FFTs (Oakforest-PACS, 1024 nodes)

	FFTE 6.2alpha			FFTE 6.2alpha with AT			
N	P	NDIV	GFlops	P_x	P_y	NDIV	GFlops
16M	1024	4	8.4	256	4	1	59.2
64M	1024	4	35.5	128	8	1	174.2
256M	1024	4	129.4	64	16	1	520.3
1G	1024	4	210.3	32	32	1	1415.6
4G	1024	4	560.0	8	128	1	1573.8
16G	1024	4	901.6	1024	1	1	2858.7
64G	1024	4	2983.1	1024	1	2	3984.5

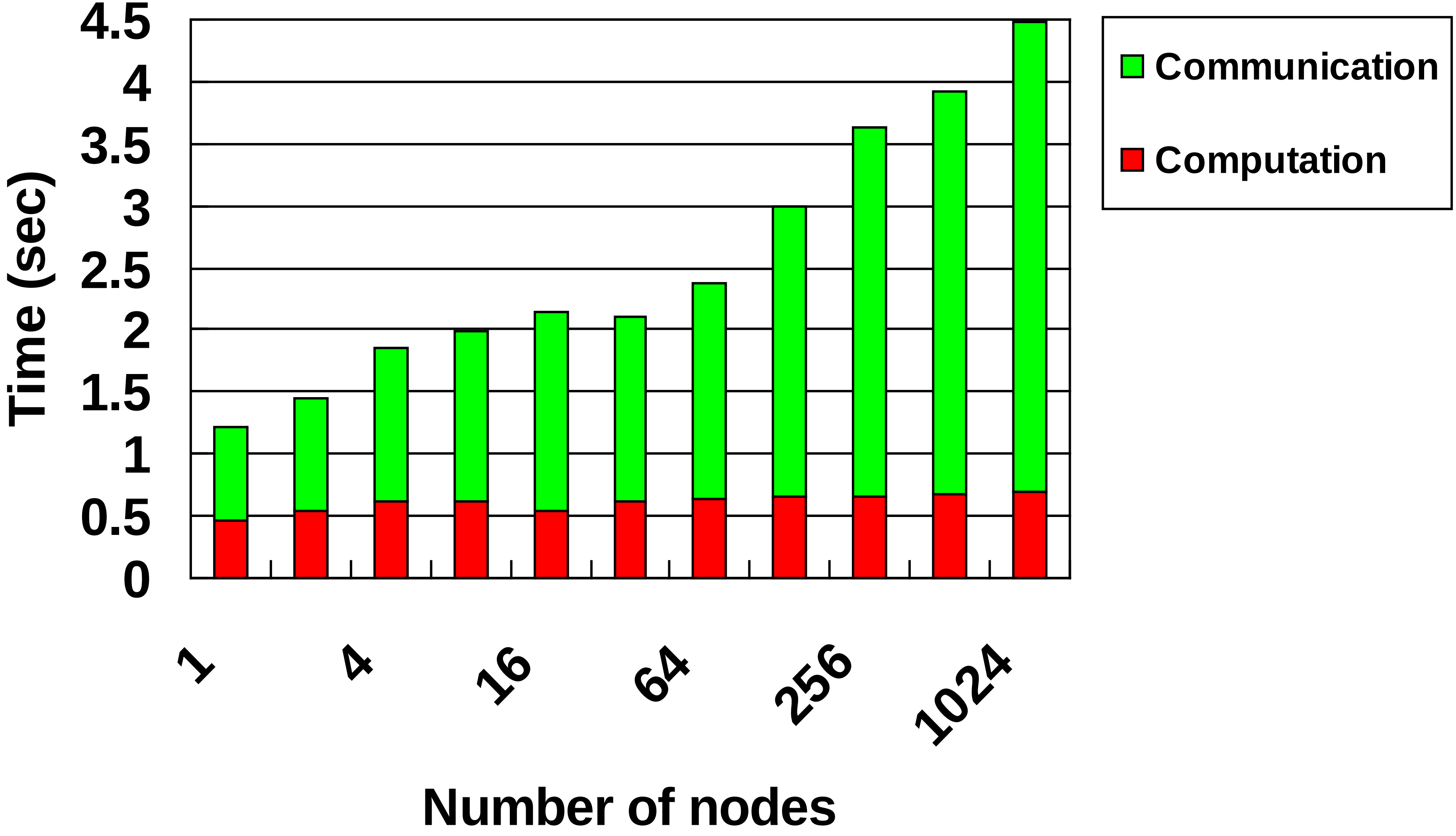
Performance of parallel 1-D FFTs（Oakforest-PACS，1024 nodes）



Performance of all-to-all communication （Oakforest-PACS，1024 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 cluster of Intel Xeon Phi processors.
- 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 all-to-all communication and the computation-communication overlap, 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 cluster of Intel Xeon Phi processors.