CCS HPC Winter Seminar

High Performance Parallel Computing Technology for Computational Sciences

“Parallel Numerical Algorithm 2”

Daisuke Takahashi
Center for Computational Sciences
University of Tsukuba

Contents of Lecture

- Fast Fourier Transform (FFT)
- Cooley-Tukey FFT and parallelization
- Six-Step FFT and parallelization
- Nine-Step FFT and blocking, parallelization

Fast Fourier Transform (FFT)

- The fast Fourier transform (FFT) is an algorithm for computing the discrete Fourier transform (DFT).
- Example applications in the scientific field
  - Solution of partial differential equations
  - Convolution, correlation calculations
  - Density function theory in first-principles calculations
- Example applications in the engineering field
  - Spectrum analyzers
  - CT scanners, MRI, and other image processing
  - With the OFDM (orthogonal frequency multiplex modulation) used in digital terrestrial television broadcasting and wireless LAN, FFTs are used in modulation/demodulation processing.

Discrete Fourier Transform (DFT)

- Discrete Fourier transform (DFT) is given by

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

$$0 \le k \le n-1,\ \omega_n = e^{-2\pi i/n}$$

Matrix-based DFT Formulation (1/4)

- When $n = 4$, a DFT can be computed as follows:

$$
\begin{aligned}
y(0) &= x(0)\omega^0 + x(1)\omega^0 + x(2)\omega^0 + x(3)\omega^0 \\
y(1) &= x(0)\omega^0 + x(1)\omega^1 + x(2)\omega^2 + x(3)\omega^3 \\
y(2) &= x(0)\omega^0 + x(1)\omega^2 + x(2)\omega^4 + x(3)\omega^6 \\
y(3) &= x(0)\omega^0 + x(1)\omega^3 + x(2)\omega^6 + x(3)\omega^9
\end{aligned}
$$

Matrix-based DFT Formulation (2/4)

- Can be expressed more simply when a matrix is used.

$$\begin{bmatrix} y(0) \\ y(1) \\ y(2) \\ y(3) \end{bmatrix} = \begin{bmatrix} \omega^0 & \omega^0 & \omega^0 & \omega^0 \\ \omega^0 & \omega^1 & \omega^2 & \omega^3 \\ \omega^0 & \omega^2 & \omega^4 & \omega^6 \\ \omega^0 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix}$$

- Requires n^2 complex multiplications and $n(n-1)$ complex additions.

Matrix-based DFT Formulation (3/4)

- Using the relation $\omega_n^{jk} = \omega_n^{jk \bmod n}$, can be written as follows:

$$\begin{bmatrix} y(0) \\ y(1) \\ y(2) \\ y(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^0 & \omega^2 \\ 1 & \omega^3 & \omega^2 & \omega^1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix}$$

Matrix-based DFT Formulation (4/4)

- Decomposition of the matrix allows the number of multiplications to be reduced.

$$\begin{bmatrix} y(0) \\ y(2) \\ y(1) \\ y(3) \end{bmatrix} = \begin{bmatrix} 1 & \omega^0 & 0 & 0 \\ 1 & \omega^2 & 0 & 0 \\ 0 & 0 & 1 & \omega^1 \\ 0 & 0 & 1 & \omega^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & \omega^0 & 0 \\ 0 & 1 & 0 & \omega^0 \\ 1 & 0 & \omega^2 & 0 \\ 0 & 1 & 0 & \omega^2 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix}$$

Performing this recursively, the amount of calculations can be reduced to $O(n \log n)$.
(The number of data n must be a composite number.)

Comparison of the Amount of Operations Needed for Calculating DFTs and FFTs

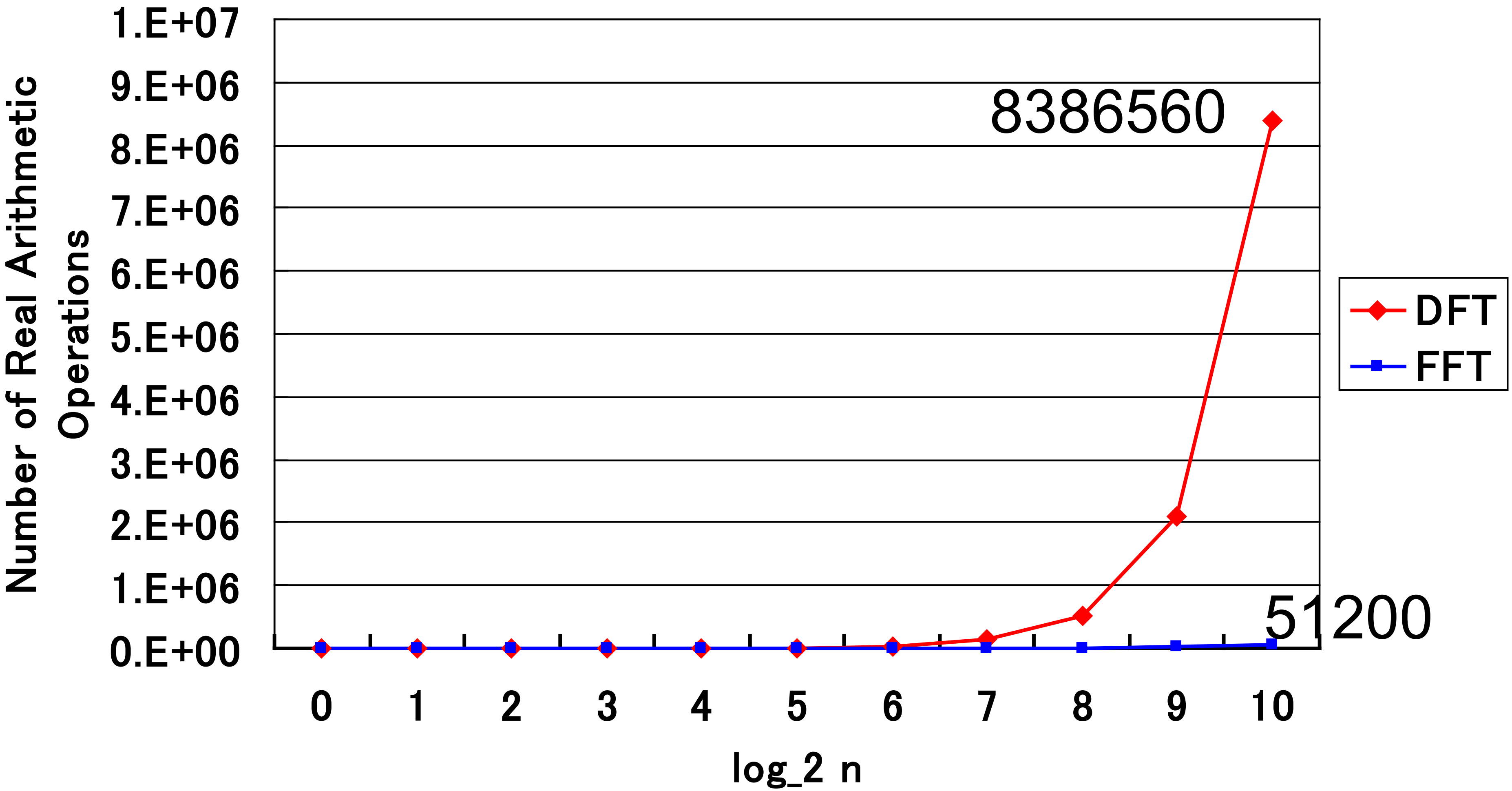
- Number of real operations for DFTs

$$T_{DFT} = 8n^2 - 2n$$

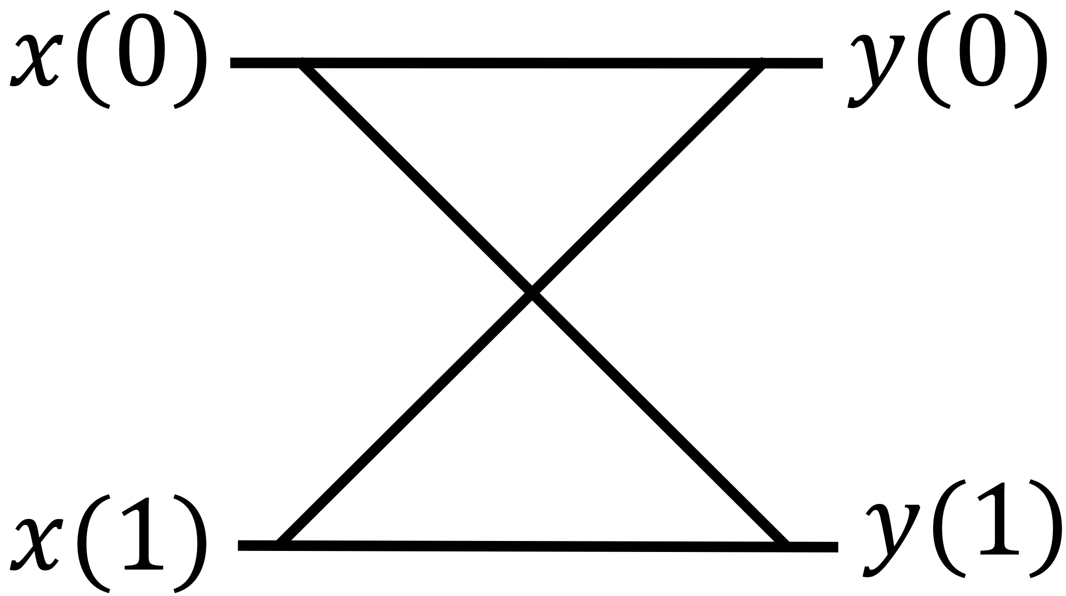
- Number of real operations for FFTs (When n is a power of two)

$$T_{FFT} = 5n \log_2 n$$

Comparison of the Amount of Operations Needed for Calculating DFTs and FFTs

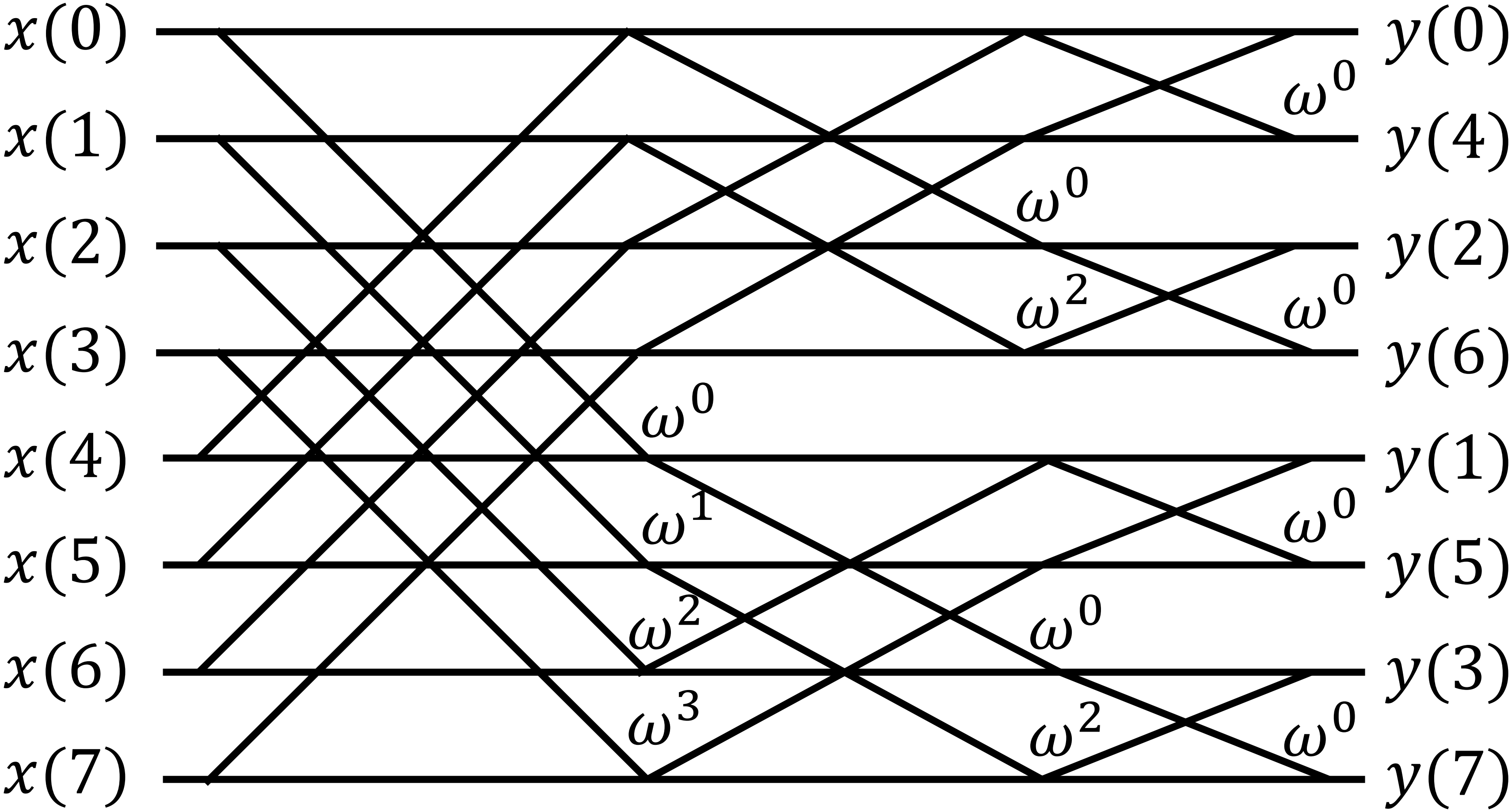


Butterfly Operation



$$y(0) = x(0) + x(1)$$
$$y(1) = \omega\{x(0) + x(1)\}$$

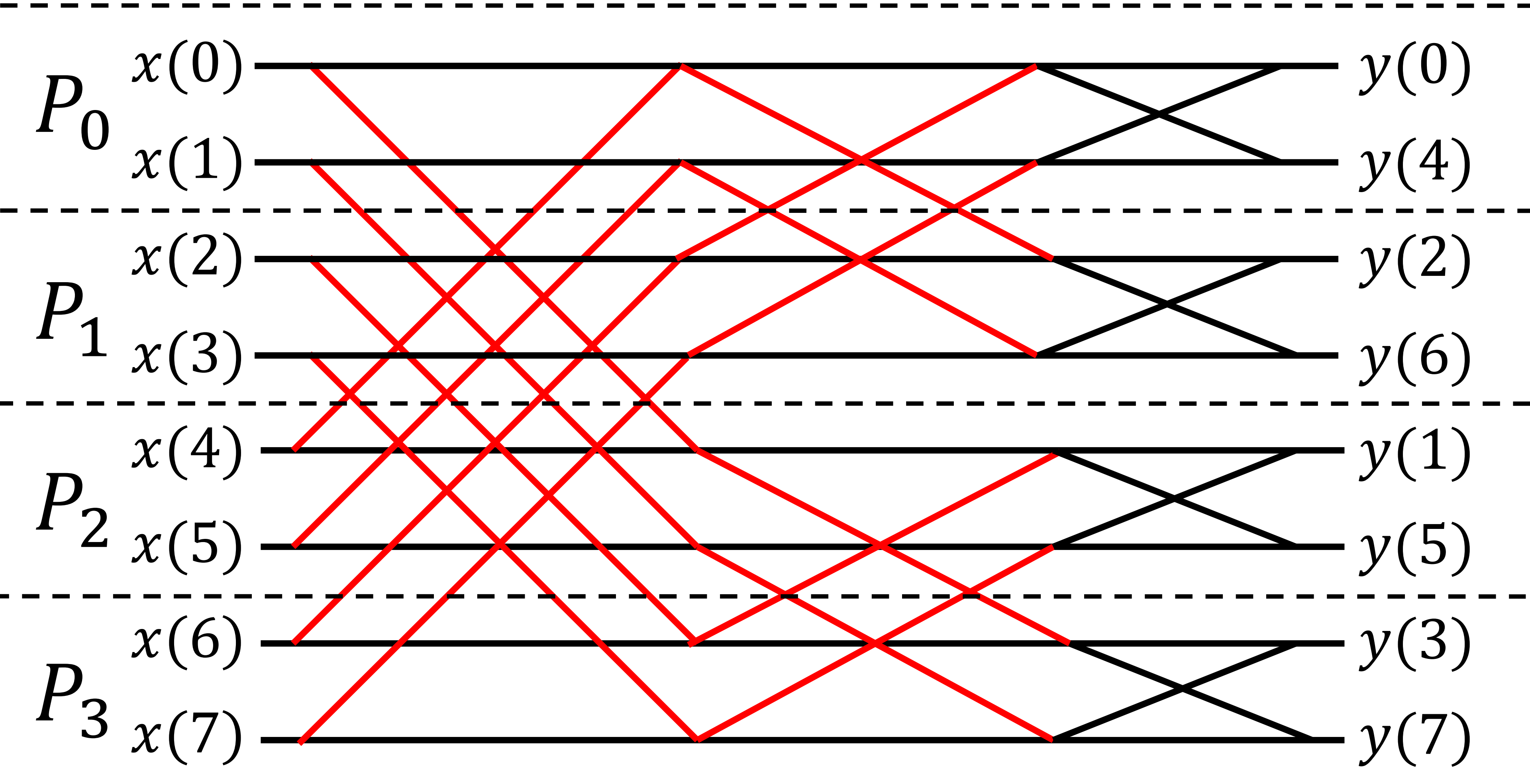
Cooley-Tukey FFT Signal Flow Diagram



Example of FFT Kernel

```
      SUBROUTINE FFT2(A,B,W,M,L)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(2,M,L,*),B(2,M,2,*),W(2,*)
C
      DO J=1,L
        WR=W(1,J)
        WI=W(2,J)
        DO I=1,M
          B(1,I,1,J)=A(1,I,J,1)+A(1,I,J,2)
          B(2,I,1,J)=A(2,I,J,1)+A(2,I,J,2)
          B(1,I,2,J)=WR*(A(1,I,J,1)-A(1,I,J,2))-WI*(A(2,I,J,1)-A(2,I,J,2))
          B(2,I,2,J)=WR*(A(2,I,J,1)-A(2,I,J,2))+WI*(A(1,I,J,1)-A(1,I,J,2))
        END DO
      END DO
      RETURN
      END
```

Parallelization of Cooley-Tukey FFT



Amount of Communication with Parallel Cooley-Tukey FFT

- If n is the number of nodes in a parallel Cooley-Tukey FFT, $\log_2 P$ stage communication is required.
- Because (n/P) double-precision complex number data is communicated (MPI_Send, MPI_Recv) at each stage, the total amount of communication is as follows:

$$T_{Cooley-Tukey} = \frac{16n}{P} \log_2 P \text{ (bytes)}$$

FFT Algorithm for $n = n_1 n_2$

- Given by $n = n_1 n_2$

$$j = j_1 + j_2 n_1,\ \ j_1 = 0, 1, \dots, n_1 - 1,\ j_2 = 0, 1, \dots, n_2 - 1$$
$$k = k_2 + k_1 n_2,\ k_1 = 0, 1, \dots, n_1 - 1,\ k_2 = 0, 1, \dots, n_2 - 1$$

- Using the above expression, the DFT formulation can be rewritten as follows:

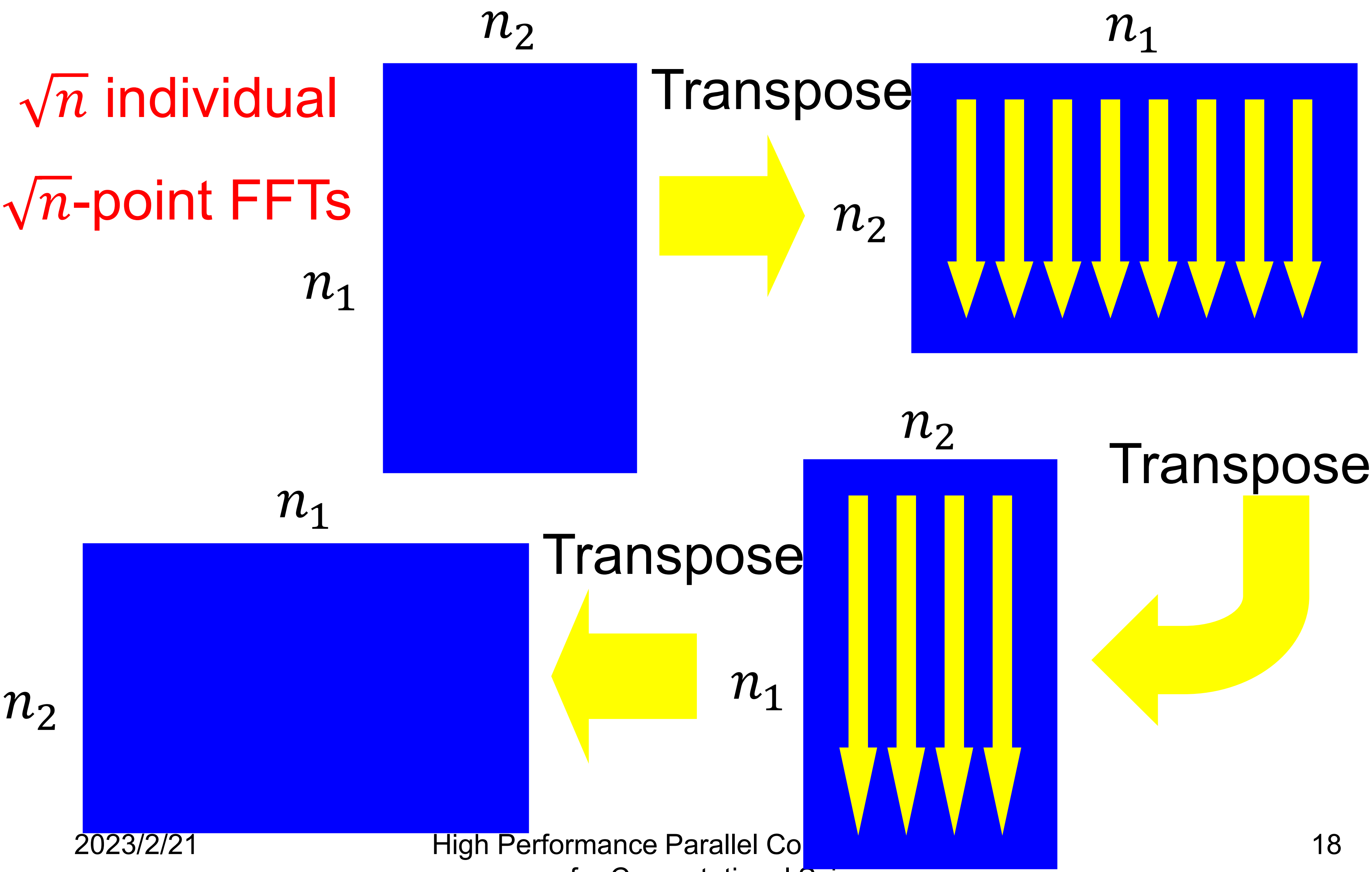
$$y(k_2, k_1) = \sum_{j_1=0}^{n_1-1} \left[\sum_{j_2=0}^{n_2-1} x(j_1, j_2) \omega_{n_2}^{j_2 k_2}\, \omega_{n_1 n_2}^{j_1 k_2} \right] \omega_{n_1}^{j_1 k_1}$$

- An n-point FFT decomposes into an n_1-point FFT and an n_2-point FFT.

Six-Step FFT Algorithm

1. Matrix transposition
2. n_1 individual n_2-point multicolumn FFT
3. Twiddle factor ($\omega_{n_1 n_2}^{j_1 k_2}$) multiplication
4. Matrix transposition
5. n_2 individual n_1-point multicolumn FFT
6. Matrix transposition

Six-Step FFT Algorithm

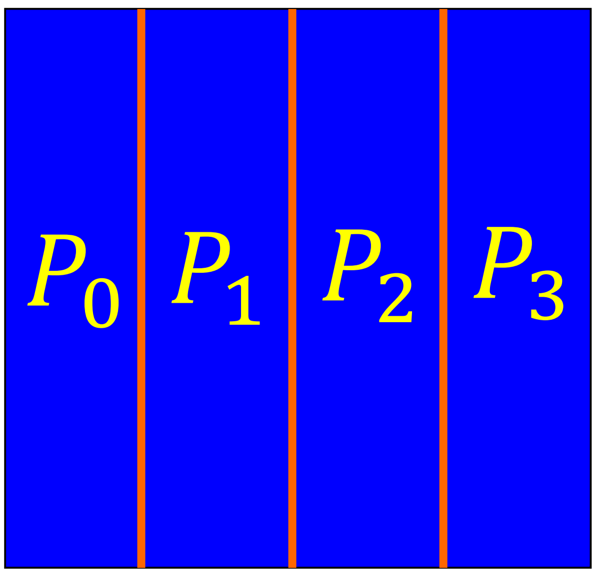


Six-Step FFT Program Example

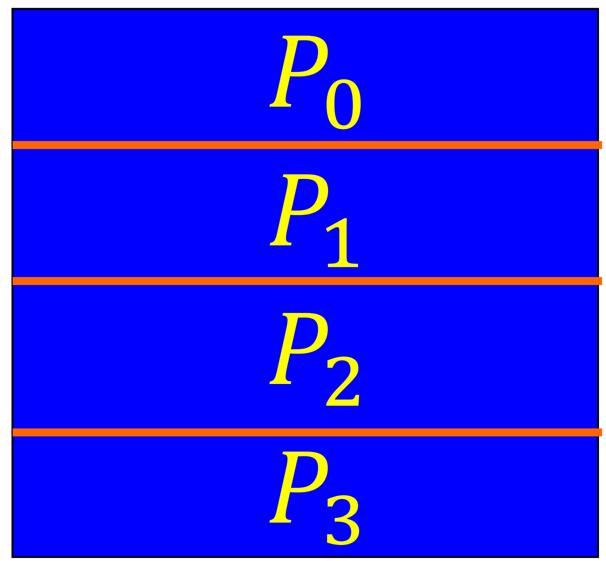
```
      SUBROUTINE FFT(A,B,W,N1,N2)
      COMPLEX*16 A(*),B(*),W(*)
C
      CALL TRANS(A,B,N1,N2)                Matrix transposition
      DO J=1,N1
        CALL FFT2(B((J-1)*N2+1),N2)        N1 individual N2-point multicolumn FFT
      END DO
      DO I=1,N1*N2
        B(I)=B(I)*W(I)                     Twiddle factor (W) multiplication
      END DO
      CALL TRANS(B,A,N2,N1)                Matrix transposition
      DO J=1,N2
        CALL FFT2(A((J-1)*N1+1),N1)        N2 individual N1-point multicolumn FFT
      END DO
      CALL TRANS(A,B,N1,N2)                Matrix transposition
      RETURN
      END
```

Method for Distribution an Array

- When using MPI for parallelization, memory can be conserved if the array is divided at each node.
- Block distribution
  - Contiguous areas are divided by the number of nodes.



Block distribution divided at each column



Block distribution divided at each row

Matrix Transposition Using All-to-All Communication (MPI_Alltoall)

P0	P1	P2	P3
0	8	16	24
1	9	17	25
2	10	18	26
3	11	19	27
4	12	20	28
5	13	21	29
6	14	22	30
7	15	23	31

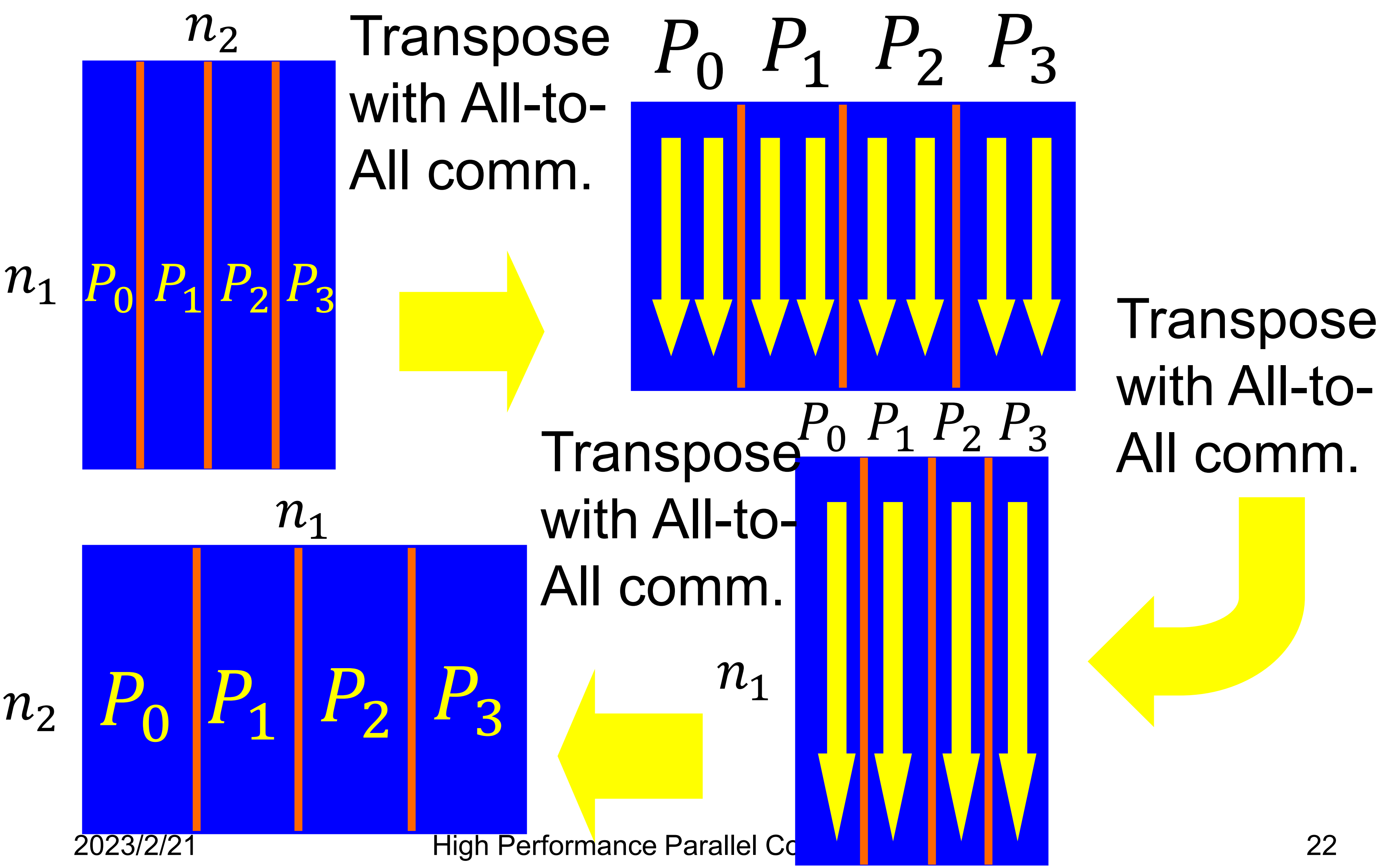
All-to-All Comm.

P0	P1	P2	P3
0	2	4	6
1	3	5	7
8	10	12	14
9	11	13	15
16	18	20	22
17	19	21	23
24	26	28	30
25	27	29	31

Local Transposition

P0	P1	P2	P3
0 1	2 3	4 5	6 7
8 9	10 11	12 13	14 15
16 17	18 19	20 21	22 23
24 25	26 27	28 29	30 31

Parallel Six-Step FFT Algorithm



Parallel Six-Step FFT Program Example

```
      SUBROUTINE PARAFFT(A,B,W,N1,N2,NPU)
      COMPLEX*16 A(*),B(*),W(*)
C
      CALL PTRANS(A,B,N1,N2,NPU)          Global matrix transposition using MPI_ALLTOALL
      DO J=1,N1/NPU
        CALL FFT2(B((J-1)*N2+1),N2)       (N1/NPU) individual N2-point multicolumn FFT
      END DO
      DO I=1,(N1*N2)/NPU
        B(I)=B(I)*W(I)                    Twiddle factor (W) multiplication
      END DO
      CALL PTRANS(B,A,N2,N1,NPU)          Global matrix transposition using MPI_ALLTOALL
      DO J=1,N2/NPU
        CALL FFT2(A((J-1)*N1+1),N1)       (N2/NPU) individual N1-point multicolumn FFT
      END DO
      CALL PTRANS(A,B,N1,N2,NPU)          Global matrix transposition using MPI_ALLTOALL
      RETURN
      END
```

Amount of Communication of Parallel Six-Step FFT

- If P is the number of nodes in a parallel six-step FFT, all-to-all communication is required three times.
- With all-to-all communication, because each node sends an (n/P^2) double-precision complex data to $P-1$ nodes, the total amount of communication is as follows:

$$T_{Six-Step} = 3 \cdot (P-1) \cdot \frac{16n}{P^2} \text{ (Bytes)}$$

Comparison of Amount of Communication with Parallel Cooley-Tukey FFT and Parallel Six-Step FFT

- Amount of communication with parallel Cooley-Tukey FFT

$$T_{Cooley-Tukey} = \frac{16n}{P} \log_2 P$$

- Amount of communication with parallel six-step FFT

$$T_{Six-Step} = 3 \cdot (P-1) \cdot \frac{16n}{P^2}$$

- Of these two methods, when $P > 8$, the parallel six-step FFT will have the lower amount of communication.

Problems with the Six-Step FFT

- In a multicolumn FFT, when $\sqrt{n}$-point each column FFT exceeds the cache size, the performance will decrease significantly.
- A distributed-memory parallel computer, when processing a large-size FFT (2^{24} points or more, for example), will be unable to achieve high performance.

3-D Formulation

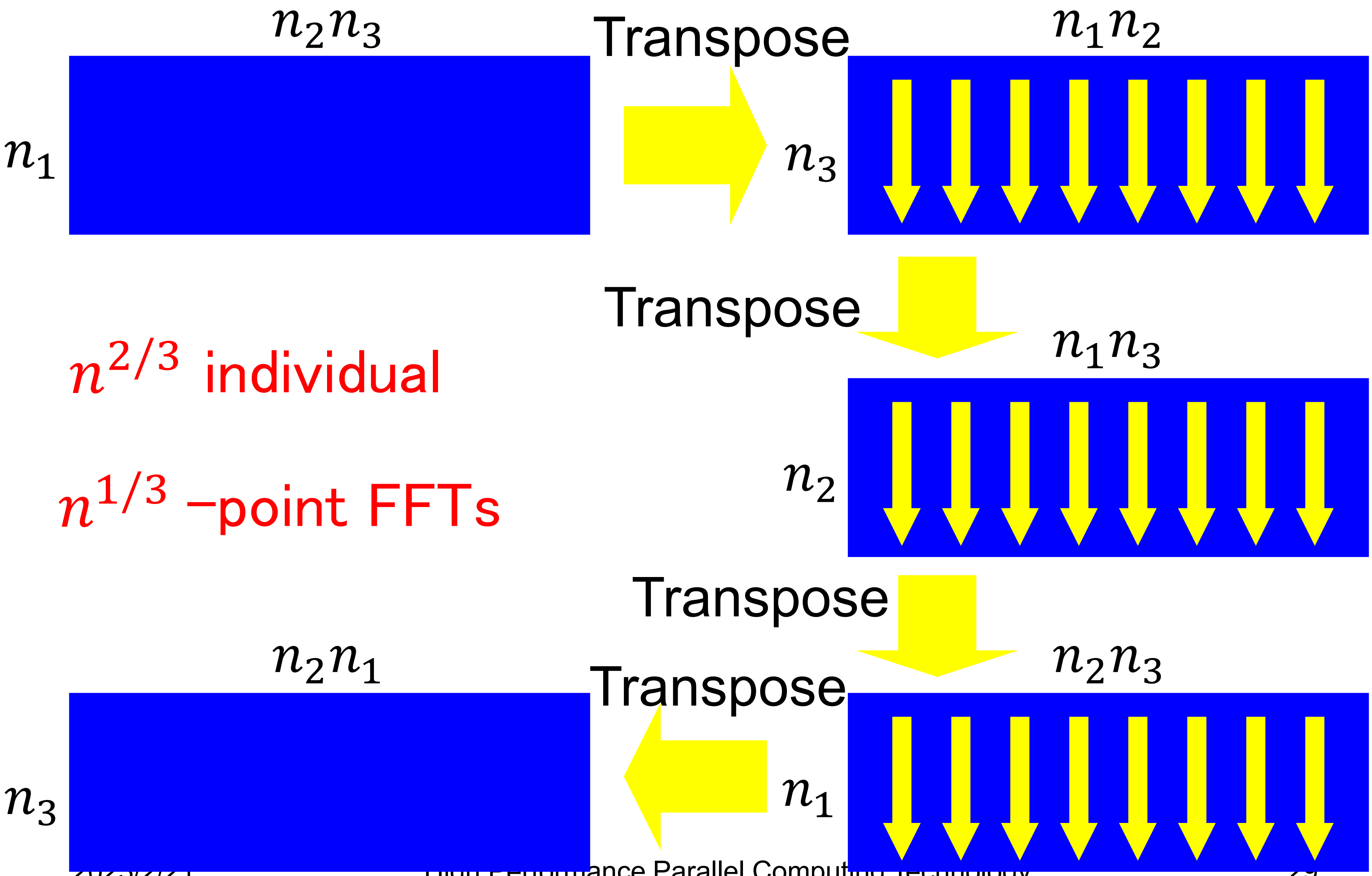
- For very large FFTs, we should switch a 3-D formulation.
- If n has factors n_1, n_2 and n_3 then

$$y(k_3, k_2, k_1) = \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} \sum_{j_3=0}^{n_3-1} x(j_1, j_2, j_3) \omega_{n_3}^{j_3 k_3} \omega_{n_2 n_3}^{j_2 k_3} \omega_{n_2}^{j_2 k_2} \omega_{n}^{j_1 k_3} \omega_{n_1 n_2}^{j_1 k_2} \omega_{n_1}^{j_1 k_1}$$

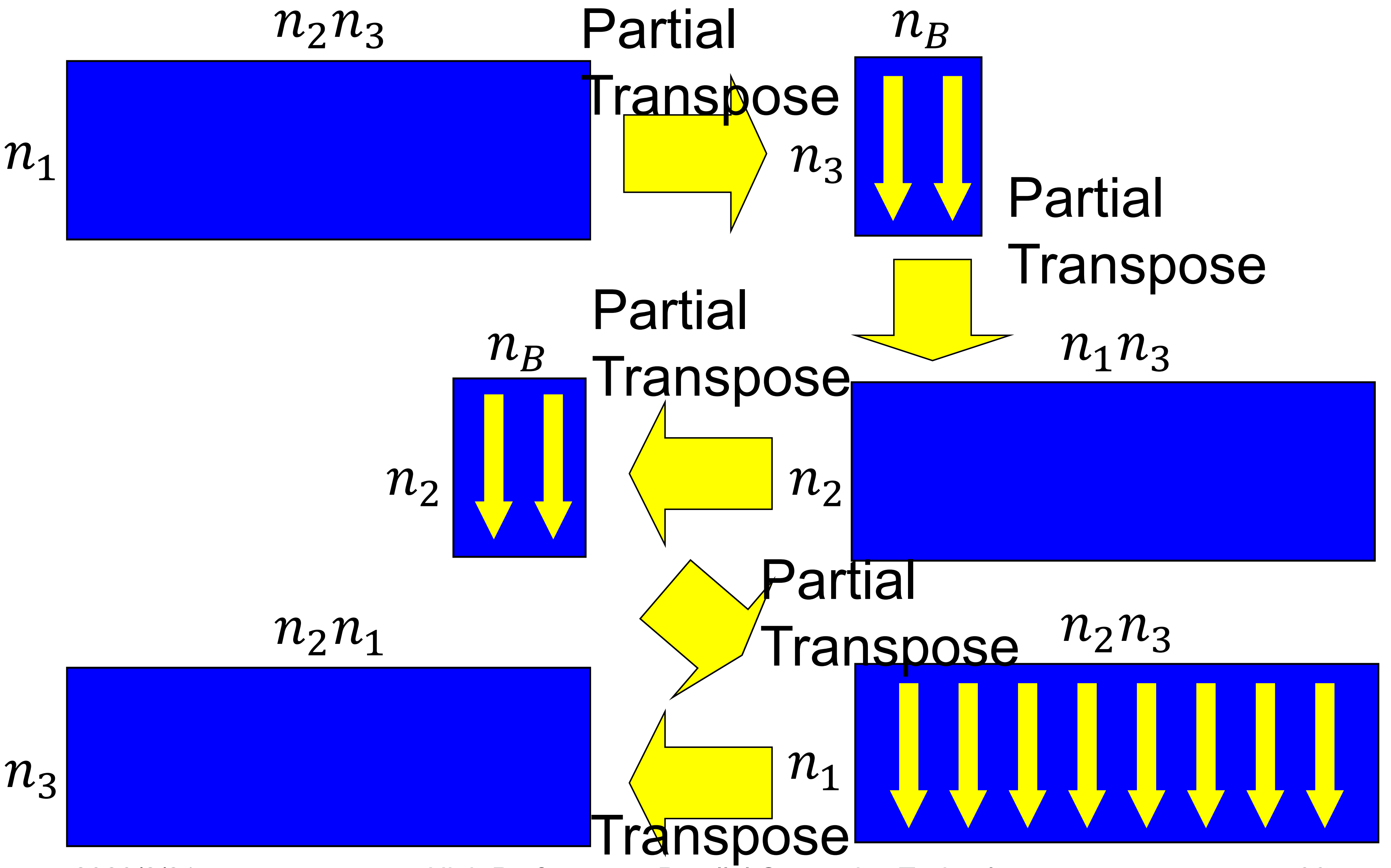
Nine-Step FFT Algorithm

1. Matrix transposition
2. $n_1 n_2$ individual n_3-point multicolumn FFT
3. Twiddle factor ($\omega_{n_2 n_3}^{j_2 k_3}$) multiplication
4. Matrix transposition
5. $n_1 n_3$ individual n_2-point multicolumn FFT
6. Twiddle factor ($\omega_{n}^{j_1 k_3} \omega_{n_1 n_2}^{j_1 k_2}$) multiplication
7. Matrix transposition
8. $n_2 n_3$ individual n_1-point multicolumn FFT
9. Matrix transposition

Nine-Step FFT Algorithm



Block Nine-Step FFT Algorithm



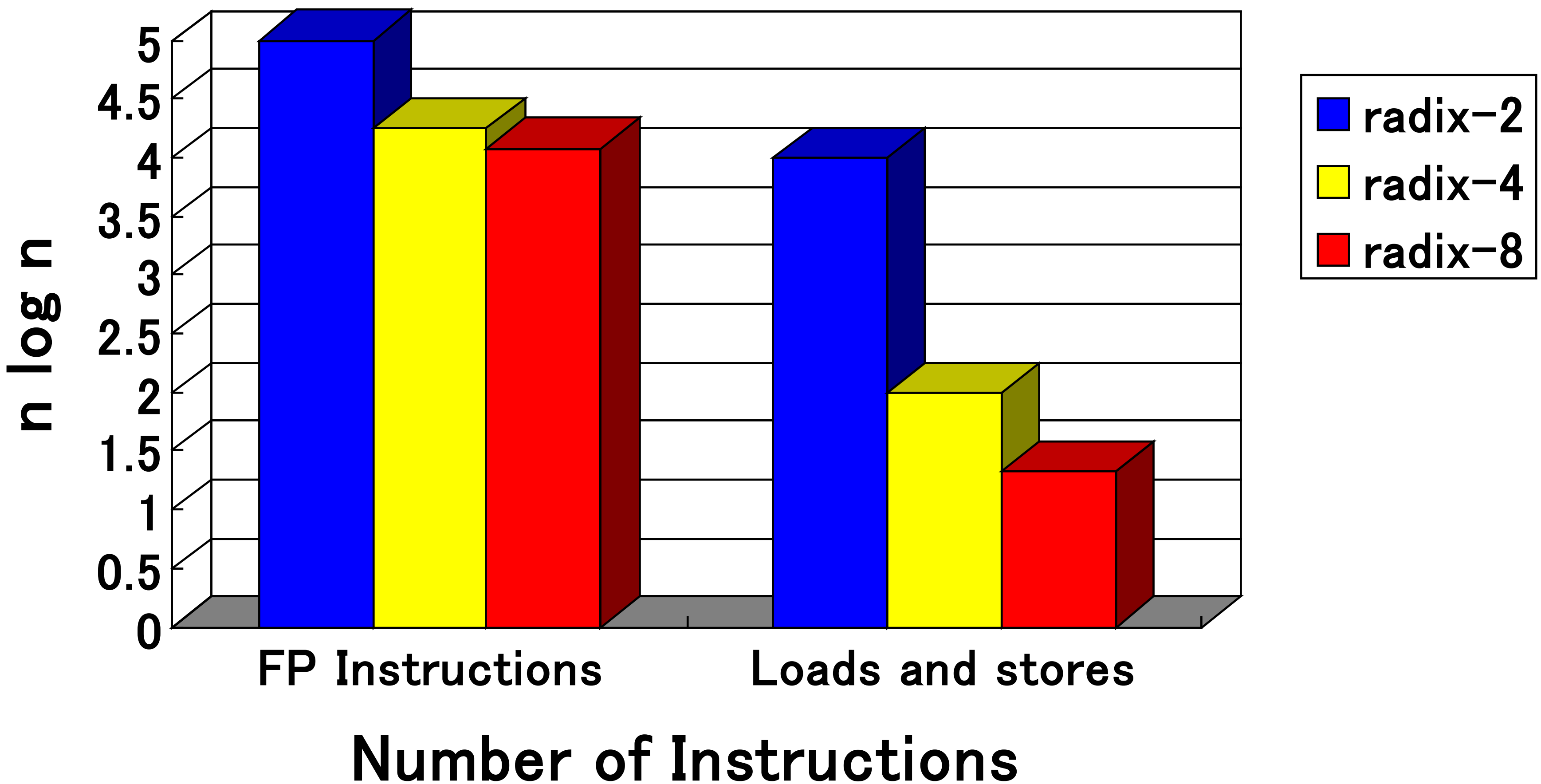
In-Cache FFT Algorithm

- In a multicolumn FFT, the following can be conceived of as in-cache FFTs, whereby each column FFT is placed in the cache.
  - Cooley-Tukey algorithm (bit-reversal permutation is needed)
  - Stockham algorithm (bit-reversal permutation is unnecessary)
- The higher radices are more efficient in terms of both memory and floating-point operations.
- In view of the high ratio of floating-point instructions to memory operations, the radix-8 FFT is more advantageous than the radix-4 FFT.

Real Inner-Loop Operations for Radix-2, 4 and 8 FFT Kernels

	Radix-2	Radix-4	Radix-8
Loads and Stores	8	16	32
Multiplications	4	12	32
Additions	6	22	66
Total floating-point operations ($n \log_2 n$)	5	4.25	4.083
Floating-point instructions	10	34	98
Floating-point / memory ratio	1.25	2.125	3.063

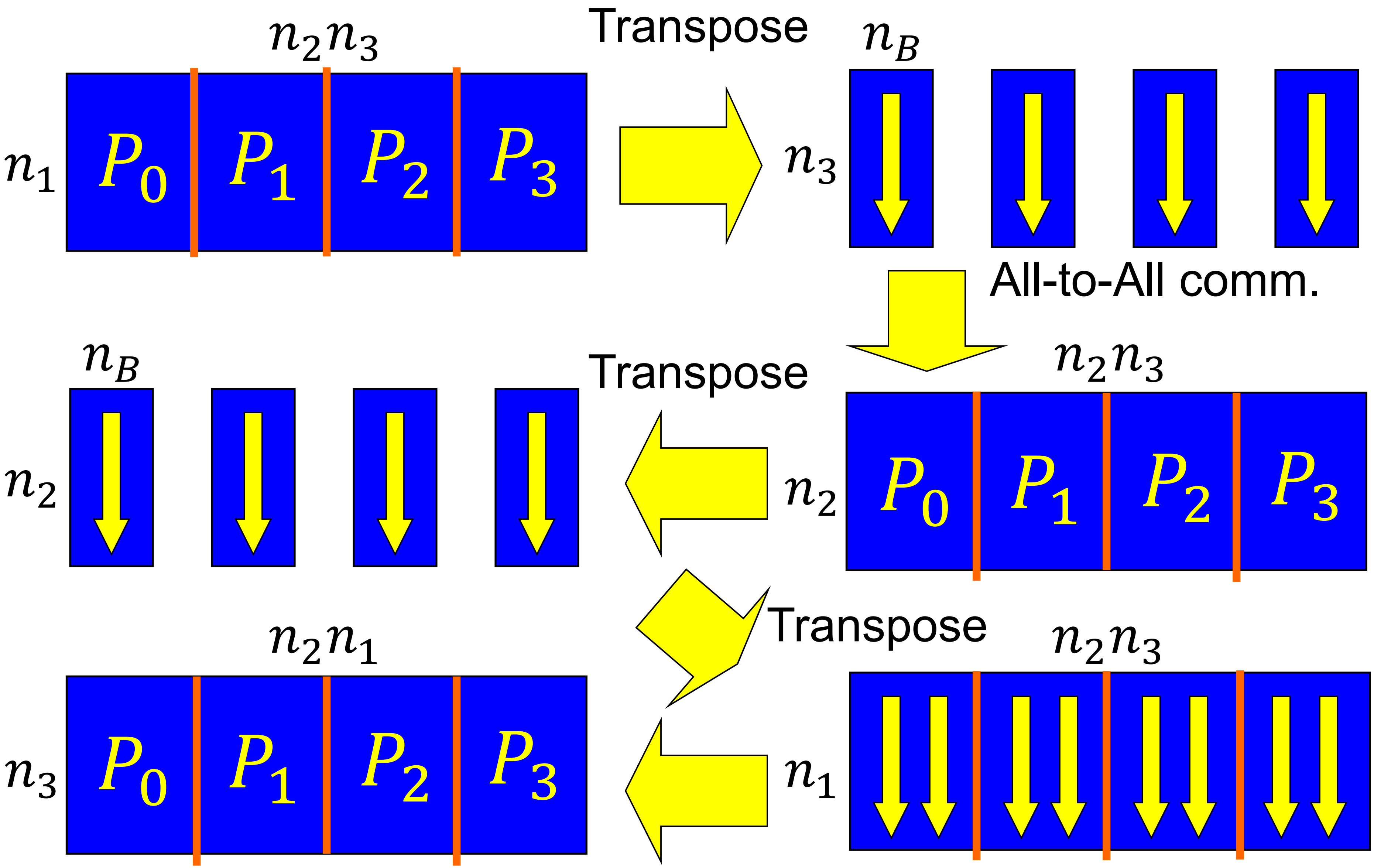
Number of Instructions for FFTs



Blocking of a Nine-Step FFT

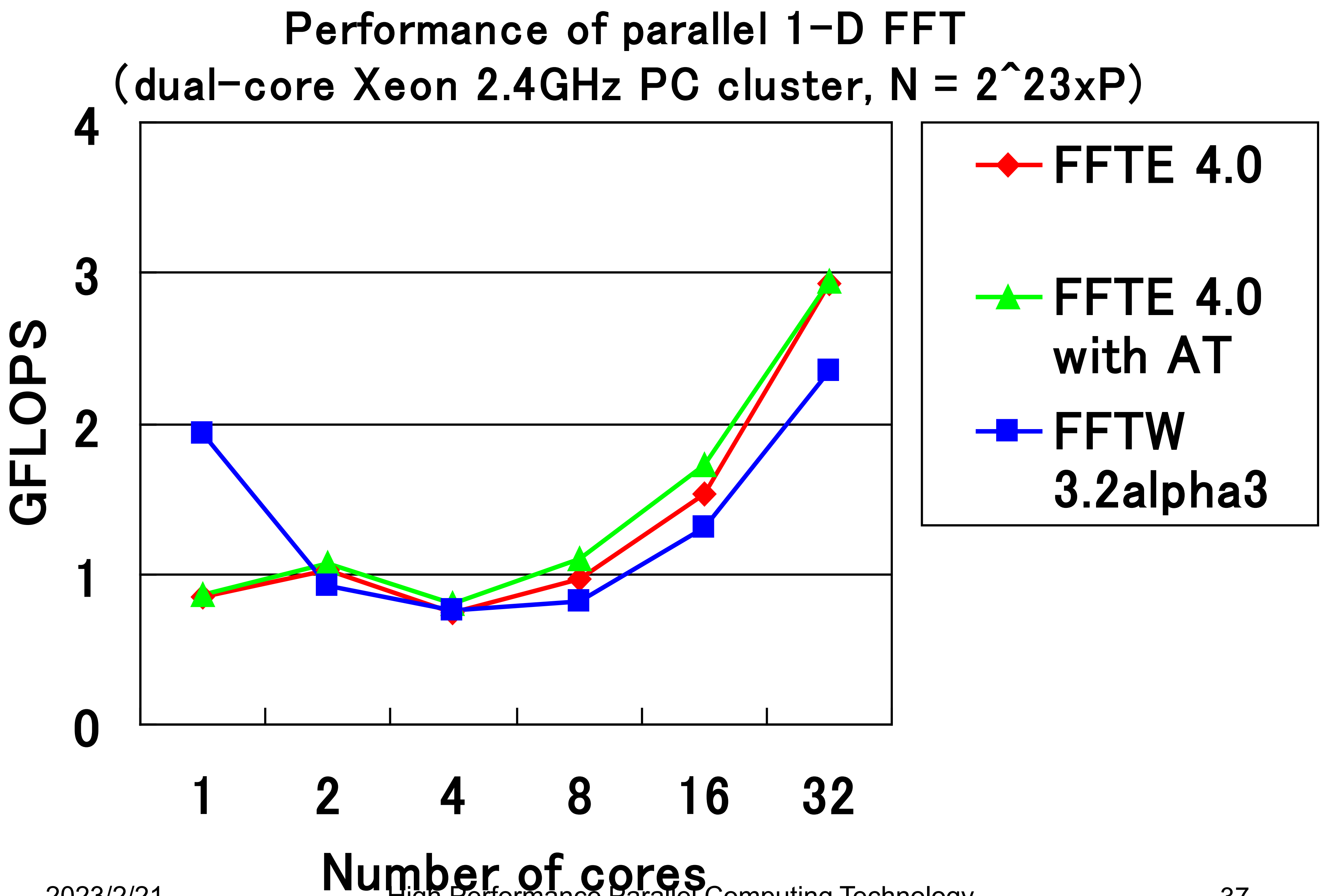
- Data in the cache, having been used for matrix transposition, can also be used with the multicolumn FFTs, thereby increasing the reusability of data in the cache.
- Once data from the main memory has been loaded into the cache, have it remain in cache as much as possible.
- Reuse data in the cache as much as possible, and when that data is truly no longer needed, write it back to the main memory.

Parallel Nine-Step FFT Algorithm

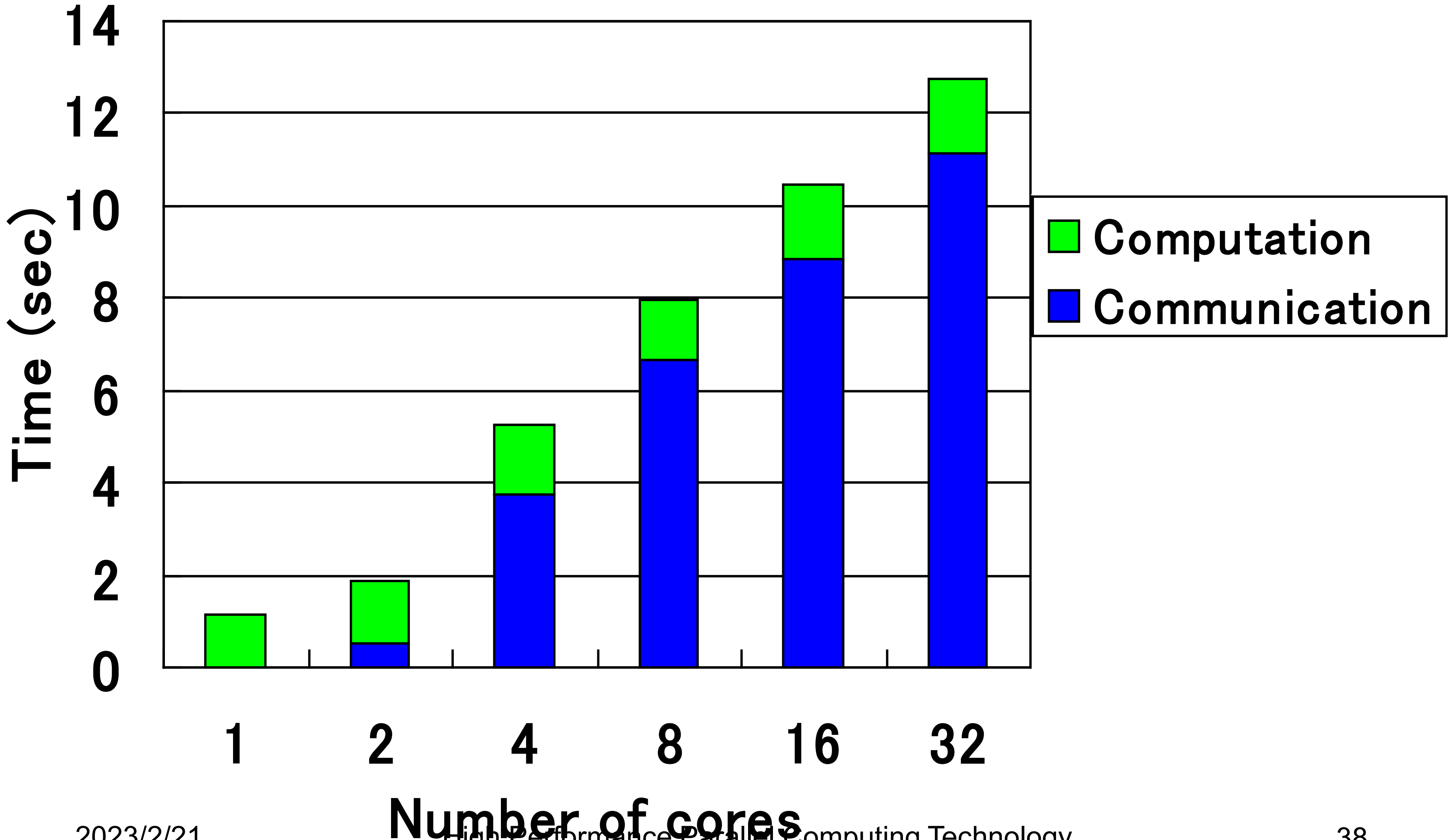


Advantages of a Block Nine-Step FFT

- With an ordinary FFT algorithm such as the Stockham FFT
  - Number of operations: $5n \log_2 n$
  - Number of main memory accesses: $4n \log_2 n$
- With a block nine-step FFT
  - Number of operations: $5n \log_2 n$
  - Number of main memory accesses: Ideally $12n$
- Because a portion of the nine-step FFT performs $n^{1/3}$-point FFT blocking, the proposed block nine-step FFT can be called a “double blocking” algorithm.


Performance of parallel 1-D FFT
(dual-core Xeon 2.4GHz PC cluster, N = 2^23xP)
GFLOPS
0
1
2
3
4
1
2
4
8
16
32
Number of cores
FFTE 4.0
FFTE 4.0 with AT
FFTW 3.2alpha3

Breakdown of parallel 1-D FFT (dual-core Xeon 2.4GHz PC cluster, N=2^23xP)



Examples of Parallel FFT Libraries

- Commercial parallel numeric computation libraries
  - Intel Cluster MKL (Math Kernel Library)
    - OpenMP version and MPI version can be used.
  - AMD ACML (AMD Core Math Library)
    - OpenMP version can be used.
- Open source parallel FFT libraries
  - FFTW (http://www.fftw.org/)
    - OpenMP version and MPI version can be used.
  - FFTE (http://www.ffte.jp/)
    - OpenMP version, MPI version, and OpenMP+MPI version can be used.

Summary

- The FFT (fast Fourier transform) has been introduced as a parallel numeric computing algorithm.
- The key is how to distribute the problem area.
  - Block distribution, cyclic distribution, block-cyclic distribution
- With a parallel FFT, because the communication part is mainly all-to-all communication, parallelization is relatively easy.
- Not only it is important to reduce the amount of communication, but the use of blocking, etc., is also important to localize the memory accesses.

Problem 7

- Develop the following programs in arbitrary programming languages:
  (1) Discrete Fourier transform (DFT)
  (2) Fast Fourier transform (FFT)
- Then, measure the execution time of a 65536-point double-complex DFT on any of your available PCs.
- Submit the source codes and performance results.