FFTE: A High-Performance FFT Library and Parallel Implementation of Classical Gram-Schmidt Orthogonalization

Daisuke Takahashi

Center for Computational Sciences/
Graduate School of Systems and Informaiton Engineering
University of Tsukuba

Outline

- FFTE: A High-Performance FFT Library
  - Features, Design and Approach
  - Performance Results
- Parallel Implementation of Classical Gram-Schmidt Orthogonalization Using Matrix Multiplication (with T. Yokozawa, T. Boku and M. Sato)
  - Column-wise Blocking CGS Algorithm
  - Recursive Blocking CGS Algorithm
  - Performance Results

FFTE: A High-Performance FFT Library

- FFTE is a Fortran subroutine library for computing the Fast Fourier Transform (FFT) in one or more dimensions.
- It includes complex, mixed-radix and parallel transforms.
- FFTE is typically faster than other publically-available FFT implementations, and is even competitive with vendor-tuned libraries.

FFTE Library - Features

- High speed
  - Supports Intel’s SSE2/SSE3 instructions.
- Parallel transforms
  - Shared / Distributed memory parallel computers (OpenMP, MPI and OpenMP + MPI)
- High portability
  - Fortran77 + OpenMP + MPI
  - Intel’s intrinsics for SSE2/SSE3 instructions.
- HPC Challenge Benchmark
  - FFTE’s 1-D parallel FFT routine has been incorporated into the HPC Challenge (HPCC) benchmark.

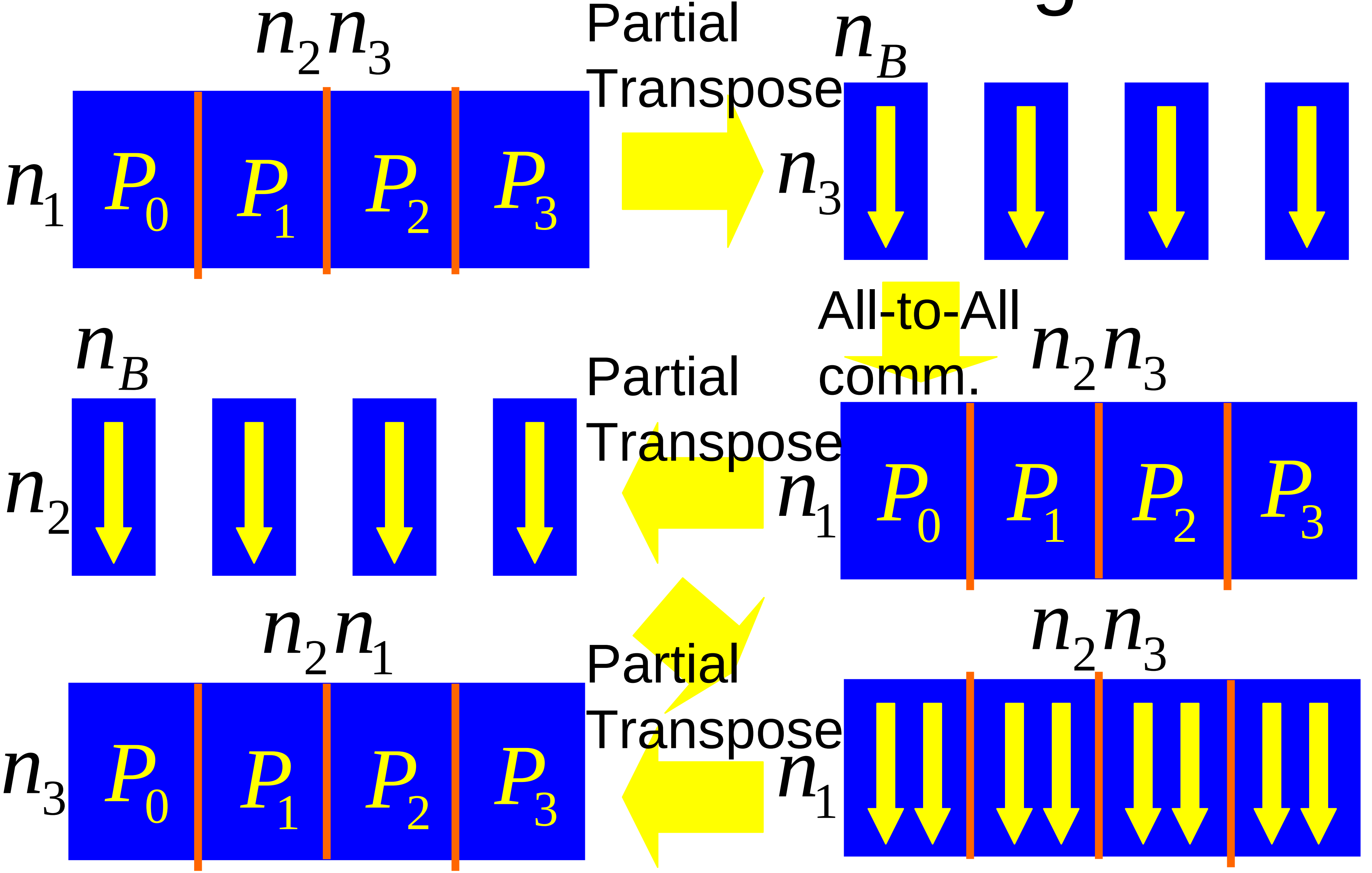
FFTE Library - Design

- Performance
  - One goal for large FFTs is to minimize the number of cache misses.
- Ease of use: routine interfaces
  - Similar to sequential SGI SCSL or Intel MKL routines
- Portability
  - Communication: MPI
  - Computation: Fortran77 + OpenMP

FFTE Library - Approach

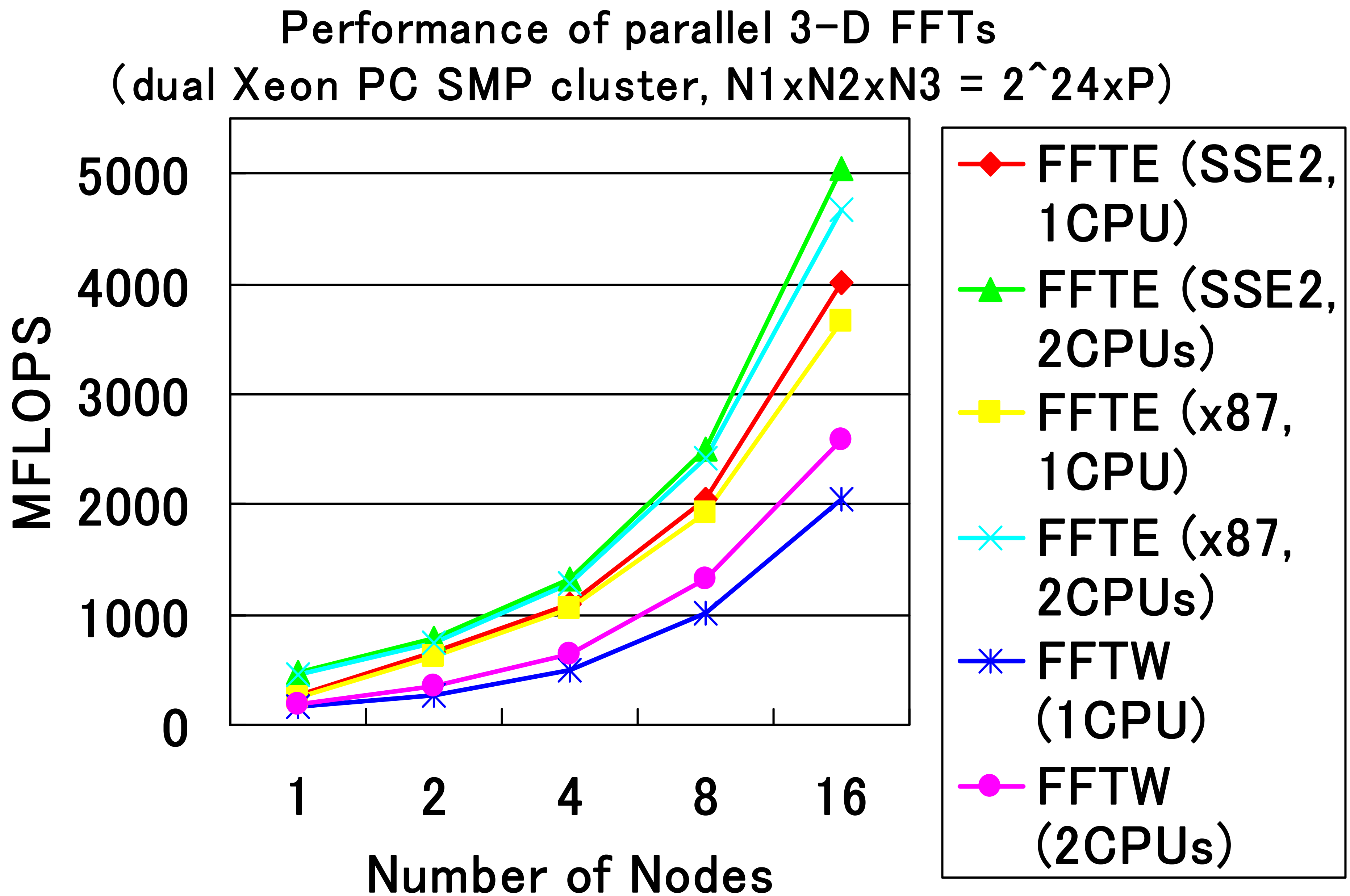
- Many FFT routines work well when data sets fit into a cache.
- When a problem size exceeds the cache size, however, the performance of these FFT routines decreases dramatically.
- Some previously presented three-dimensional FFT algorithms require
  - Three multicolumn FFTs.
  - Three data transpositions.
    → The chief bottlenecks in cache-based processors.
- We combine the multicolumn FFTs and transpositions to reduce the number of cache misses.

Parallel Block 3-D FFT Algorithm

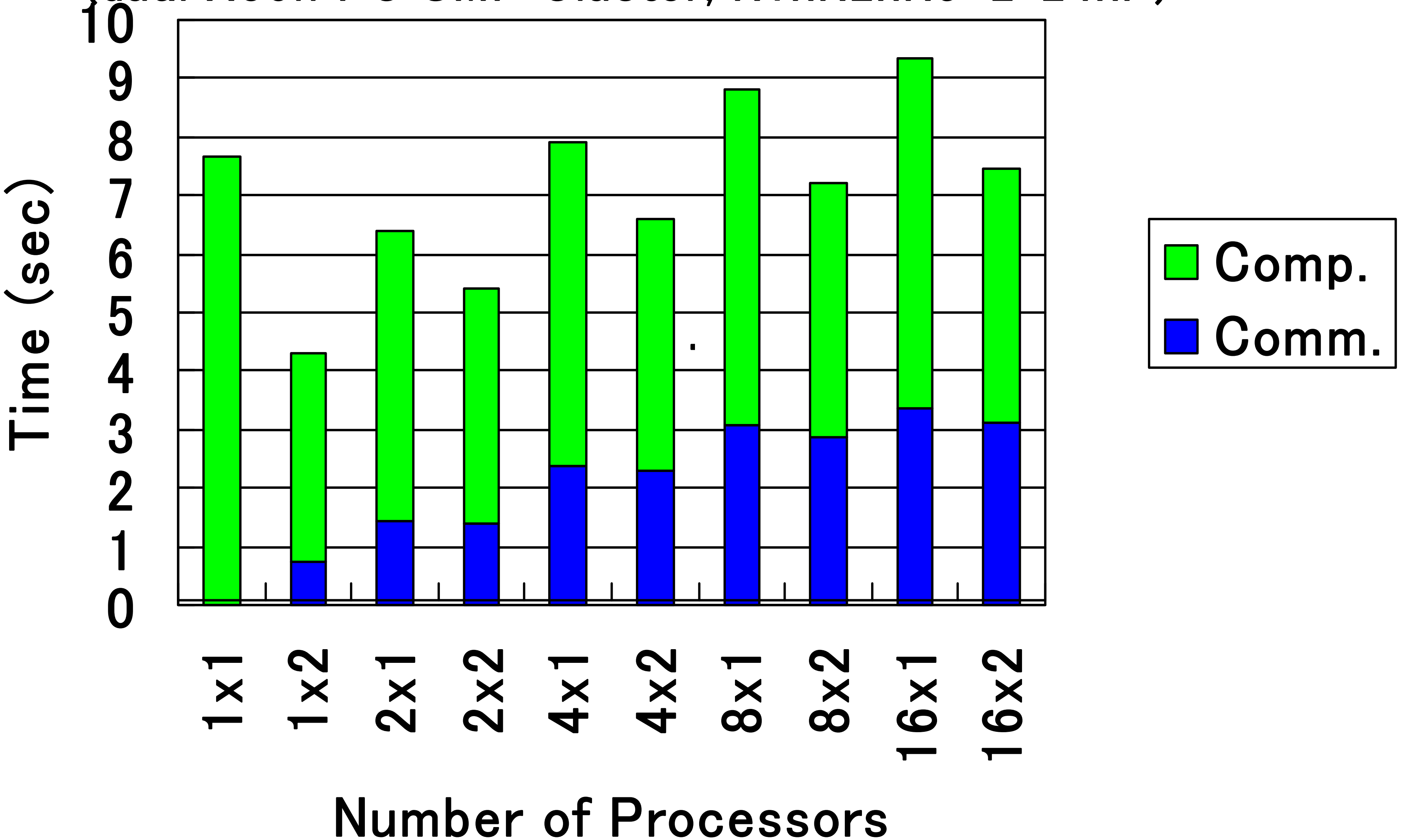


Performance Results

- To evaluate the implemented parallel 3-D FFTs, we compared
  - The implemented parallel 3-D FFT, named FFTE (ver 3.2, supports SSE2, using MPI)
  - FFTW (ver. 2.1.5, not support SSE2, using MPI)
- Target parallel machine:
  - A 16-node dual PC SMP cluster (dual Xeon 2.8GHz, 2GB DDR-SDRAM / node, Linux 2.4.20smp).
  - Interconnected through a Myrinet-2000 switch.
  - MPI-SCore [http://www.pccluster.org] was used.
  - We used an intra-node MPI library for the PC SMP cluster.


Performance of parallel 3-D FFTs
(dual Xeon PC SMP cluster, N1xN2xN3 = 2^24xP)
MFLOPS
5000
4000
3000
2000
1000
0
1
2
4
8
16
Number of Nodes
FFTE (SSE2, 1CPU)
FFTE (SSE2, 2CPUs)
FFTE (x87, 1CPU)
FFTE (x87, 2CPUs)
FFTW (1CPU)
FFTW (2CPUs)

Breakdown of parallel 3-D FFTs
(dual Xeon PC SMP Cluster, N1xN2xN3=2^24xP)



Parallel Implementation of Classical Gram-Schmidt Orthogonalization

- Gram-Schmidt orthogonalization process is one of the fundamental algorithms in linear algebra.
  - Density-funtional theory (DFT) code includes Gram-Schmidt orthogonalization of a large set of wave functions.
- Two basic computational variants of the Gram-Schmidt process exist:
  - Classical Gram-Schmidt algorithm (CGS)
  - Modified Gram-Schmidt algorithm (MGS)
    - Often selected for practical application
    - More stable than the CGS algorithm

Background

- Modified Gram-Schmidt (MGS) algorithm
  – Cannot be expressed by Level-2 BLAS.
  – Parallel implementation requires additional communication.
- Classical Gram-Schmidt (CGS) algorithm
  – Can be expressed by Level-2 BLAS.
  – CGS with DGKS correction [Daniel et al. ‘76] is one of the most efficient way to perform the orthogonalization process.

Classical Gram-Schmidt Algorithm

- The CGS orthogonalization can be performed by using Level-2 BLAS.
- Suitable for parallelization.
- The CGS algorithm is not stable.

$$
\begin{aligned}
&\text{do } j = 1, n \\
&\quad \mathbf{q_j} = \mathbf{a_j} \\
&\quad \text{do } i = 1, j-1 \\
&\quad\quad \mathbf{q_j} = \mathbf{q_j} - (\mathbf{q_i}, \mathbf{a_j})\mathbf{q_i} \\
&\quad \text{end do} \\
&\quad \mathbf{q_j} = \mathbf{q_j} / \|\mathbf{q_j}\| \\
&\text{end do}
\end{aligned}
$$

$\mathbf{a_j}$	Raw data vector
$\mathbf{q_j}$	Orthogonalized vector
$\|\mathbf{q_j}\|$	Euclid norm
$(\mathbf{q_i}, \mathbf{a_j})$	Inner product

Naive Implementation

- CGS for m by n matrix can be computed by
  - $(m-1)$ matrix-vector multiplications (GEMV: Level-2 BLAS)
  - m normalizations (NRM2 and SCAL: Level-1 BLAS)

Let the matrix $\mathbf{A}$ is denoted as $\mathbf{A}=(\mathbf{a}_1\mathbf{a}_2\cdots\mathbf{a}_n)$.

$$\mathbf{q}_1=\mathbf{a}_1,\quad \underline{\mathbf{q}_1=\mathbf{q}_1/\|\mathbf{q}_1\|}$$

$$\mathbf{q}_2=\underline{\mathbf{a}_2-(\mathbf{q}_1,\mathbf{a}_2)\mathbf{q}_1},\quad \underline{\mathbf{q}_2=\mathbf{q}_2/\|\mathbf{q}_2\|}$$

$$\mathbf{q}_3=\underline{\mathbf{a}_3-(\mathbf{q}_1,\mathbf{a}_3)\mathbf{q}_1-(\mathbf{q}_2,\mathbf{a}_3)\mathbf{q}_2},\quad \underline{\mathbf{q}_3=\mathbf{q}_3/\|\mathbf{q}_3\|}$$

$$\mathbf{q}_4=\underline{\mathbf{a}_4-(\mathbf{q}_1,\mathbf{a}_4)\mathbf{q}_1-(\mathbf{q}_2,\mathbf{a}_4)\mathbf{q}_2-(\mathbf{q}_3,\mathbf{a}_4)\mathbf{q}_3},\quad \underline{\mathbf{q}_4=\mathbf{q}_4/\|\mathbf{q}_4\|}$$

$$\mathbf{q}_5=\underline{\mathbf{a}_5-(\mathbf{q}_1,\mathbf{a}_5)\mathbf{q}_1-(\mathbf{q}_2,\mathbf{a}_5)\mathbf{q}_2-(\mathbf{q}_3,\mathbf{a}_5)\mathbf{q}_3-(\mathbf{q}_4,\mathbf{a}_5)\mathbf{q}_4},\quad \underline{\mathbf{q}_5=\mathbf{q}_5/\|\mathbf{q}_5\|}$$

$$\mathbf{q}_6=\underline{\mathbf{a}_6-(\mathbf{q}_1,\mathbf{a}_6)\mathbf{q}_1-(\mathbf{q}_2,\mathbf{a}_6)\mathbf{q}_2-(\mathbf{q}_3,\mathbf{a}_6)\mathbf{q}_3-(\mathbf{q}_4,\mathbf{a}_6)\mathbf{q}_4-(\mathbf{q}_5,\mathbf{a}_6)\mathbf{q}_5},\quad \underline{\mathbf{q}_6=\mathbf{q}_6/\|\mathbf{q}_6\|}$$

GEMV

NRM2, SCAL

CGS using Matrix Multiplication

- The CGS orthogonalization of a matrix can be changed into a matrix multiplication (GEMM: Level-3 BLAS).

$$\mathbf{q}_1 = \mathbf{a}_1, \quad \underline{\mathbf{q}_1 = \mathbf{q}_1 / \|\mathbf{q}_1\|}$$

$$\mathbf{q}_2 = \underline{\mathbf{a}_2 - (\mathbf{q}_1, \mathbf{a}_2)\mathbf{q}_1}, \quad \underline{\mathbf{q}_2 = \mathbf{q}_2 / \|\mathbf{q}_2\|}$$

$$\mathbf{q}_3 = \underline{\mathbf{a}_3 - (\mathbf{q}_1, \mathbf{a}_3)\mathbf{q}_1 - (\mathbf{q}_2, \mathbf{a}_3)\mathbf{q}_2}, \quad \underline{\mathbf{q}_3 = \mathbf{q}_3 / \|\mathbf{q}_3\|}$$

$$\mathbf{q}_4 = \underline{\mathbf{a}_4 - (\mathbf{q}_1, \mathbf{a}_4)\mathbf{q}_1 - (\mathbf{q}_2, \mathbf{a}_4)\mathbf{q}_2 - (\mathbf{q}_3, \mathbf{a}_4)\mathbf{q}_3}, \quad \underline{\mathbf{q}_4 = \mathbf{q}_4 / \|\mathbf{q}_4\|}$$

$$\mathbf{q}_5 = \underline{\mathbf{a}_5 - (\mathbf{q}_1, \mathbf{a}_5)\mathbf{q}_1 - (\mathbf{q}_2, \mathbf{a}_5)\mathbf{q}_2 - (\mathbf{q}_3, \mathbf{a}_5)\mathbf{q}_3} \underline{- (\mathbf{q}_4, \mathbf{a}_5)\mathbf{q}_4}, \quad \underline{\mathbf{q}_5 = \mathbf{q}_5 / \|\mathbf{q}_5\|}$$

$$\mathbf{q}_6 = \underline{\mathbf{a}_6 - (\mathbf{q}_1, \mathbf{a}_6)\mathbf{q}_1 - (\mathbf{q}_2, \mathbf{a}_6)\mathbf{q}_2 - (\mathbf{q}_3, \mathbf{a}_6)\mathbf{q}_3} \underline{- (\mathbf{q}_4, \mathbf{a}_6)\mathbf{q}_4 - (\mathbf{q}_5, \mathbf{a}_6)\mathbf{q}_5}, \quad \underline{\mathbf{q}_6 = \mathbf{q}_6 / \|\mathbf{q}_6\|}$$

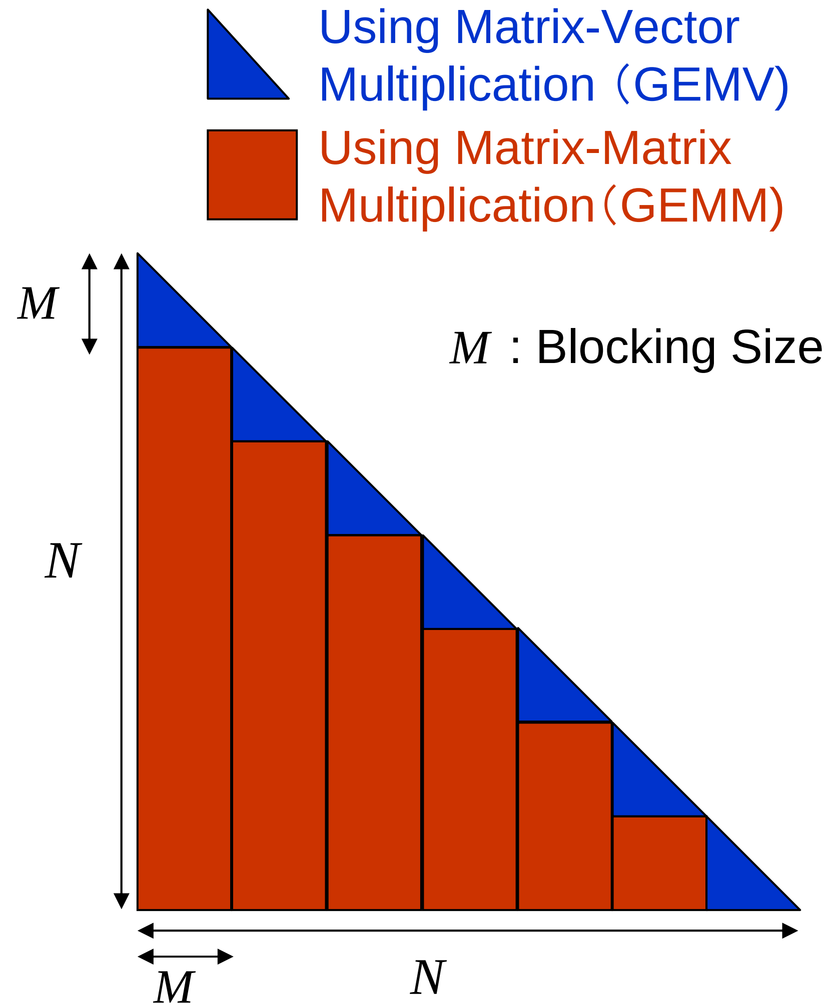
GEMM

GEMV

NRM2, SCAL

Column-wise Blocking CGS

$$
\begin{aligned}
&\text{begin CBCGS}(A, Q, N, M, s, h)\\
&\quad \text{do } j = s, h, M\\
&\qquad \mathbf{q}_j = \mathbf{q}_j / \|\mathbf{q}_j\|\\
&\qquad \text{do } i = j+1, j+M\\
&\qquad\quad \mathbf{w} = Q^T_{s,s+i}\mathbf{a}_{s+i}\\
&\qquad\quad \mathbf{q}_{s+i} = Q_{s-i,s}\mathbf{w}\\
&\qquad\quad \mathbf{q}_{s+i} = \mathbf{q}_{s+i} / \|\mathbf{q}_{s+i}\|\\
&\qquad \text{end do}\\
&\qquad W = Q^T_{j+M,N-(j+M)} A_{j,j+M}\\
&\qquad Q_{j+M,N-(j+M)} = W Q_{j+M,N-(j+M)}\\
&\quad \text{end do}\\
&\text{end}
\end{aligned}
$$



Recursive Blocking CGS

begin RBCGS(A, Q, N, M, s, h)

if $(h \le M)$ then

$$\mathbf{q}_s = \mathbf{q}_s / \|\mathbf{q}_s\|$$

do $i = 1, h$

$$\mathbf{w} = Q^T_{s,s+i}\mathbf{a}_{s+i}$$

$$\mathbf{q}_{s+i} = Q_{s-i,s}\mathbf{w}$$

$$\mathbf{q}_{s+i} = \mathbf{q}_{s+i} / \|\mathbf{q}_{s+i}\|$$

end do

else

RBCGS($A, Q, N, M, s, h/2$)

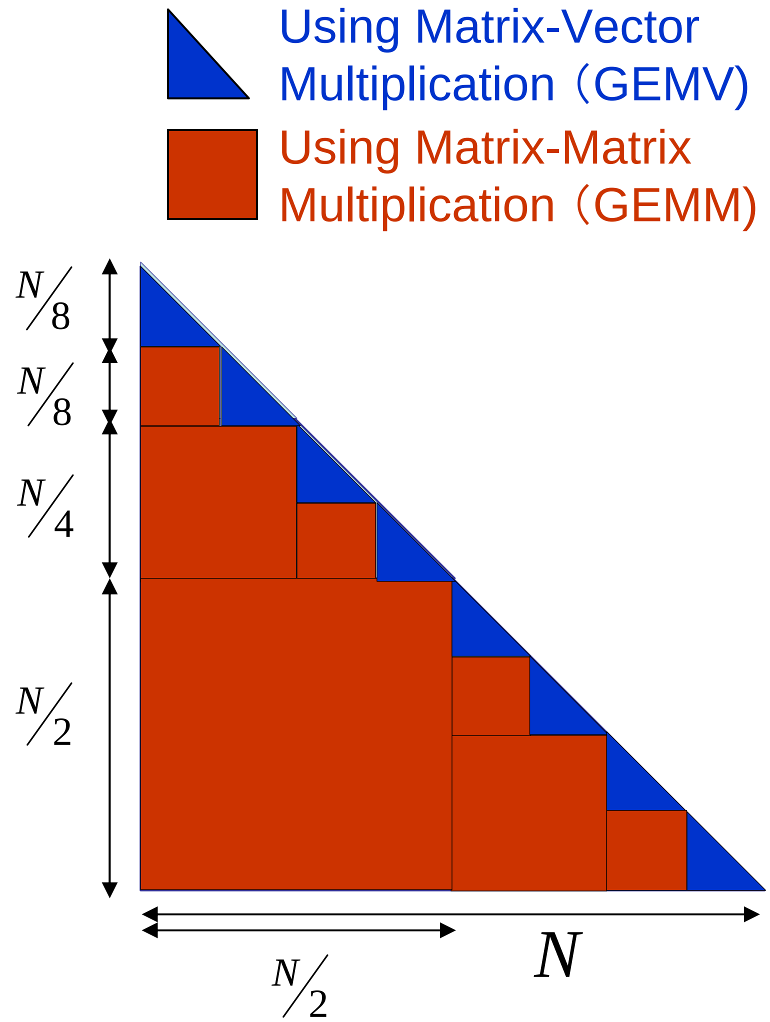
$$W = Q^T_{s-(h/2),s+(h/2)} A_{s-(h/2),s+(h/2)}$$

$$Q_{s-(h/2),s+(h/2)} = W Q_{s-(h/2),s+(h/2)}$$

RBCGS($A, Q, N, M, s + h/2, h/2$)

end if

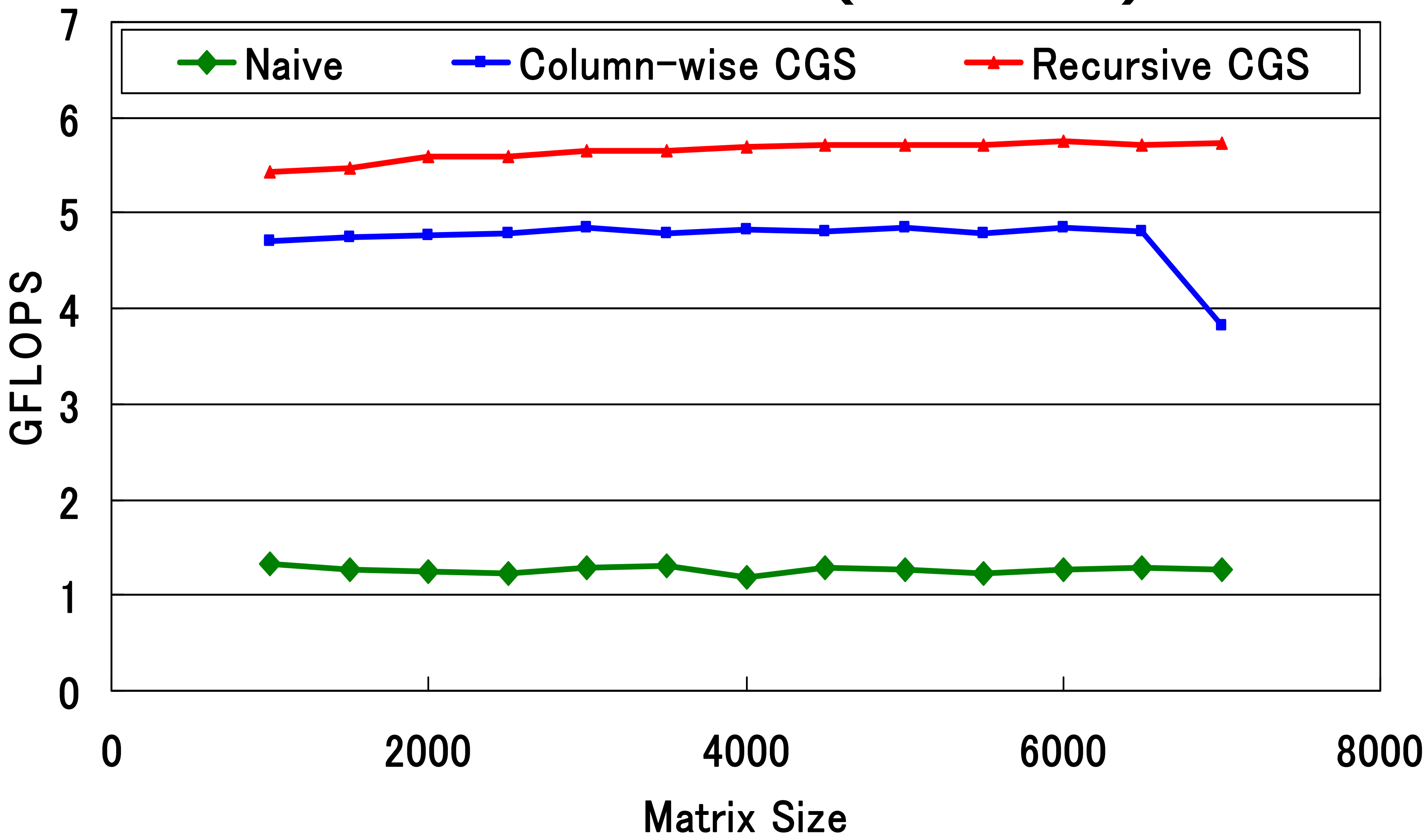
end



Evaluation Environment

- A 32-node Xeon PC-cluster
  - Xeon 3GHz, 1GB DDR2 Memory
- 1000Base-T Gigabit Ethernet
- OpenMPI 1.2.3
- GotoBLAS r1.19
- Intel C Compiler 10.0
- Linux 2.6.20-1.2933.fc6smp
- All programs were run in 64-bit mode

Performance on Xeon 3GHz (1CPU)



Performance on 32 node 3GHz Xeon PC Cluster