FFT and Parallel Numerical Libraries

Daisuke Takahashi
Center for Computational Sciences
University of Tsukuba

Outline

- Overview of Research Results
- Collaborations
- Fast Fourier Transform (FFT)
  - FFTE: A High-Performance FFT Library
  - Performance Results of Parallel 1-D FFT on the K computer
- Parallel Numerical Libraries: High Precision Arithmetic Operations
  - Triple and Quadruple Precision BLAS on GPUs
  - CUMP: The CUDA Multiple Precision Arithmetic Library
- Summary

Overview of Research Results (1/3)

- Fast Fourier Transform (FFT)
  - Implementation of Parallel 1-D FFT on GPU Clusters [Takahashi, IEEE CSE 2013]
  - An Implementation of Parallel 2-D FFT Using Intel AVX Instructions on Multi-Core Processors [Takahashi, ICA3PP 2012]
  - An Implementation of Parallel 1-D FFT on the K computer [Takahashi (U. Tsukuba), Uno and Yokokawa (RIKEN), IEEE HPCC 2012]
  - An Implementation of Parallel 3-D FFT with 2-D Decomposition on a Massively Parallel Cluster of Multi-core Processors [Takahashi, PPAM 2009]

Overview of Research Results (2/3)

- Triple and Quadruple Precision BLAS on GPUs
  - Implementation and Evaluation of Triple Precision BLAS Subroutines on GPUs [Mukunoki and Takahashi, IPDPSW 2012]
  - Implementation and Evaluation of Quadruple Precision BLAS Functions on GPUs [Mukunoki and Takahashi, PARA 2010]
- Multiple-Precision Arithmetic
  - Implementation of Multiple-Precision Floating-Point Arithmetic Library for GPU Computing [Nakayama and Takahashi, PDCS 2011]
  - Parallel implementation of multiple-precision arithmetic and 2,576,980,370,000 decimal digits of π calculation [Takahashi, Parallel Computing, 2010]

Overview of Research Results (3/3)

- Sparse Matrix-Vector Multiplication on GPUs
  - Optimization of Sparse Matrix-vector Multiplication for CRS Format on NVIDIA Kepler Architecture GPUs [Mukunoki and Takahashi, ICCSA 2013]
  - Automatic Tuning of Sparse Matrix-Vector Multiplication for CRS format on GPUs [Yoshizawa and Takahashi, IEEE CSE 2012]
  - Optimization of Sparse Matrix-Vector Multiplication by Auto Selecting Storage Schemes on GPU [Kubota and Takahashi, ICCSA 2011]

Collaborations (1/2)

- Collaboration between computer science and material science
  - Density-funtional theory (DFT) code includes Gram-Schmidt orthogonalization of a large set of wave functions.
  - Implemented an effective algorithm for Gram-Schmidt orthogonalization with matrix multiplication.
  - J.-I. Iwata, D. Takahashi (U. Tsukuba), A. Oshiyama (U. Tokyo), T. Boku, K. Shiraishi, S. Okada and K. Yabana (U.Tsukuba): A massively-parallel electronic-structure calculations based on real-space density functional theory, J. Comput. Phys. (2010).

Collaborations (2/2)

- Collaboration between computer science and molecular science
  - 3D reference interaction site model (3D-RISM)
  - The ordinary parallel 3D-RISM program has a limitation on the number of parallelism because of the limitations of the 3-D FFT with slab-wise decomposition.
  - Implemented a parallel 3-D FFT with 2-D (pencil-wise) decomposition.
  - The new 3D-RISM program achieved good scalability on the K computer.
  - Y. Maruyama (Keio U.), N. Yoshida (Kyushu U.), H. Tadano, D. Takahashi, M. Sato (U. Tsukuba) and F. Hirata (Inst. of Mol. Sciences): Massively Parallel Implementation of 3D-RISM Calculation with Volumetric 3D-FFT, J. Comput. Chem. (submitted).

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 real, complex, mixed-radix and parallel transforms.
- FFTE is typically faster than other publically-available FFT implementations, and is even competitive with vendor-tuned libraries.
- Available at http://www.ffte.jp/

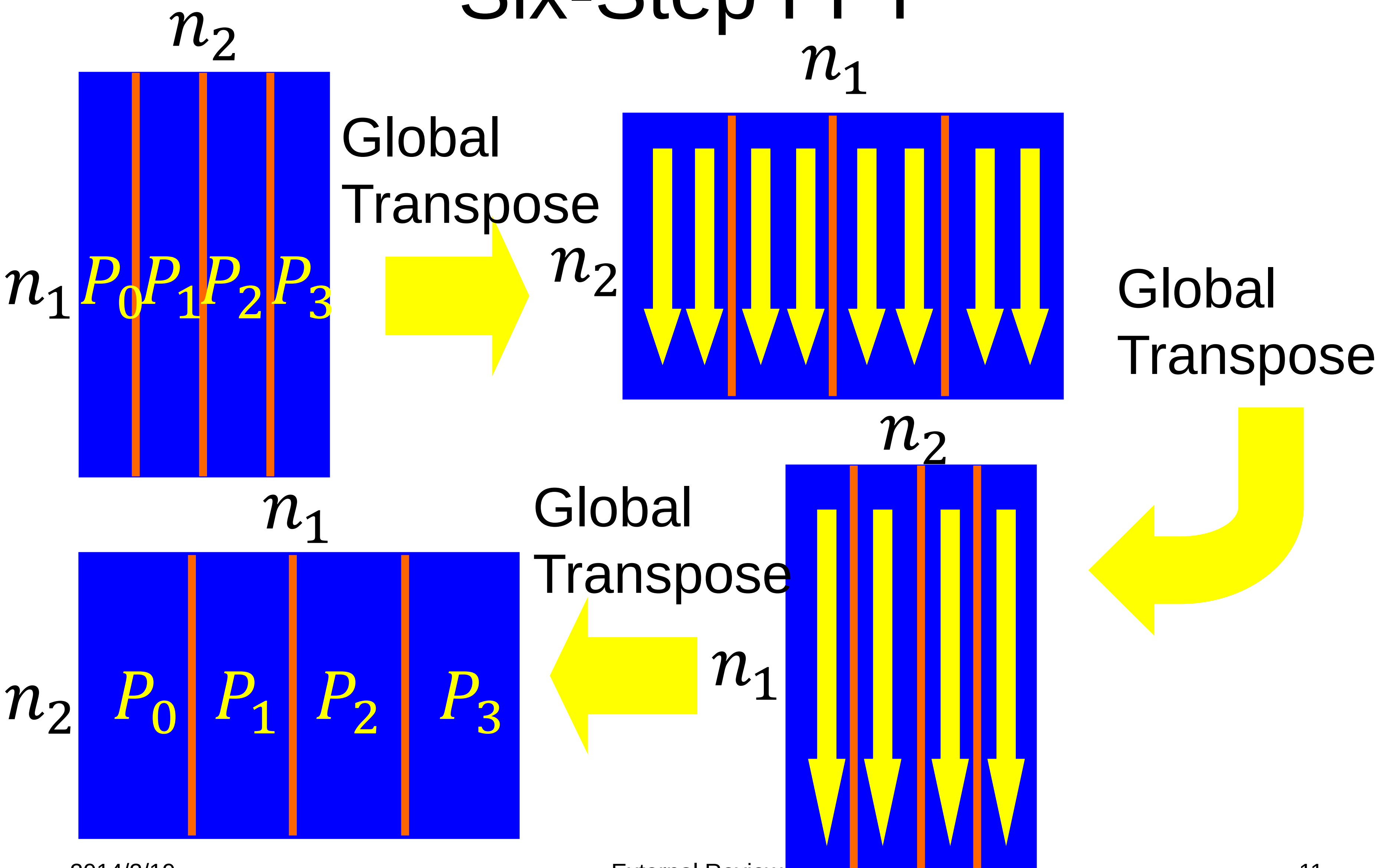
Features

- Parallel transforms
  - Shared / Distributed memory parallel computers (OpenMP, MPI and OpenMP + MPI)
- High portability
  - Fortran + OpenMP + MPI
- Data layout
  - 1-D and 2-D decomposition (for parallel 3-D FFT)
- HPC Challenge Benchmark
  - FFTE's 1-D parallel FFT routine has been incorporated into the HPC Challenge (HPCC) benchmark.

Approach: Parallel 1-D FFT

- Many FFT algorithms work well when the data sets fit into a cache.
- When the problem size exceeds the cache size, however, the performance of these FFT algorithms decreases dramatically.
- The key issue of the design for large FFTs is to minimize the number of cache misses.
- The six-step FFT algorithm requires two multicolumn FFTs and three data transpositions.
- For extremely large FFTs, each column FFT cannot fit into the cache.
- In this case, the six-step FFT can be recursively applied to each column FFT.
- We call this a recursive six-step FFT algorithm.

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



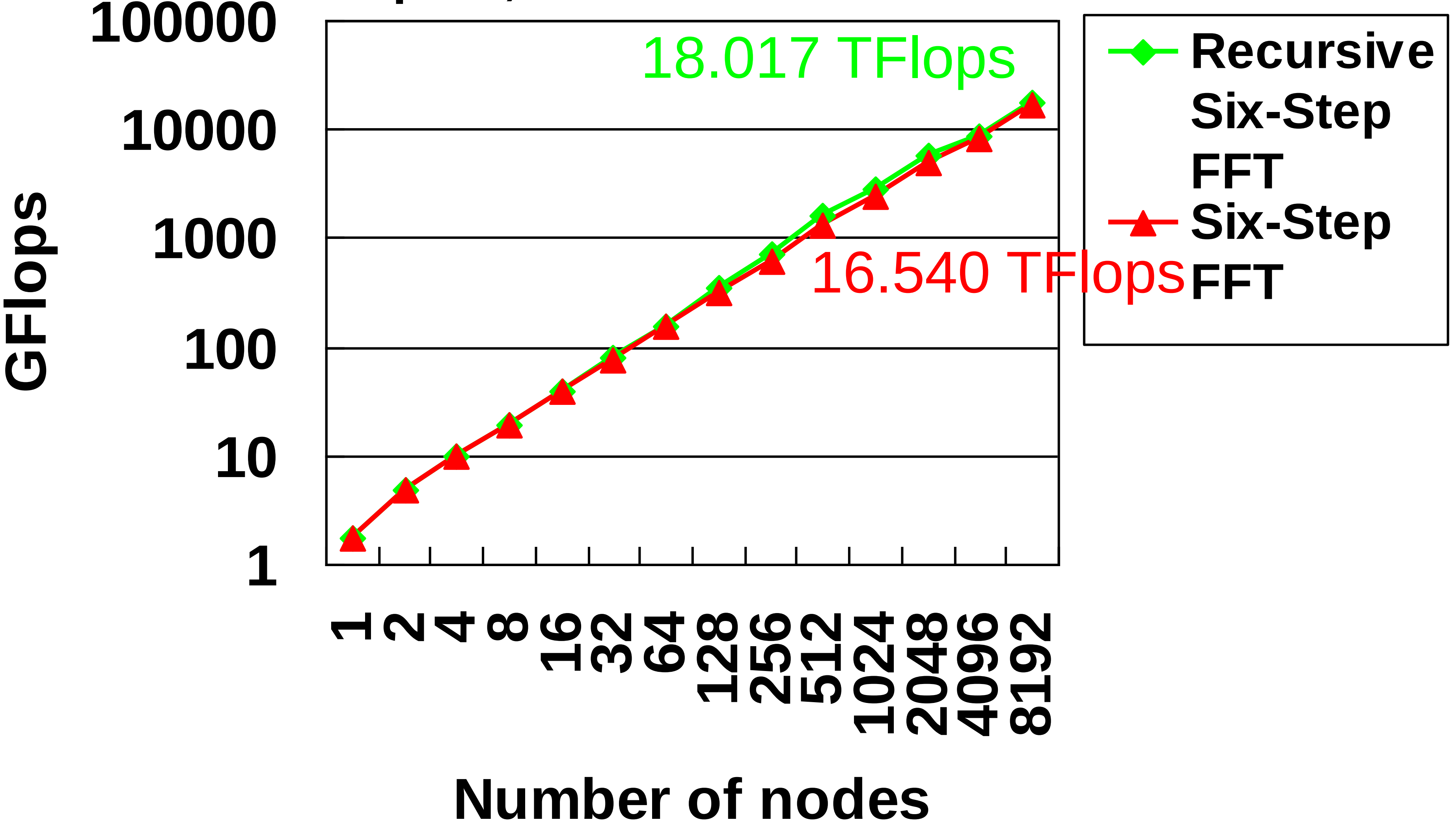
Recursive Six-Step FFT Algorithm

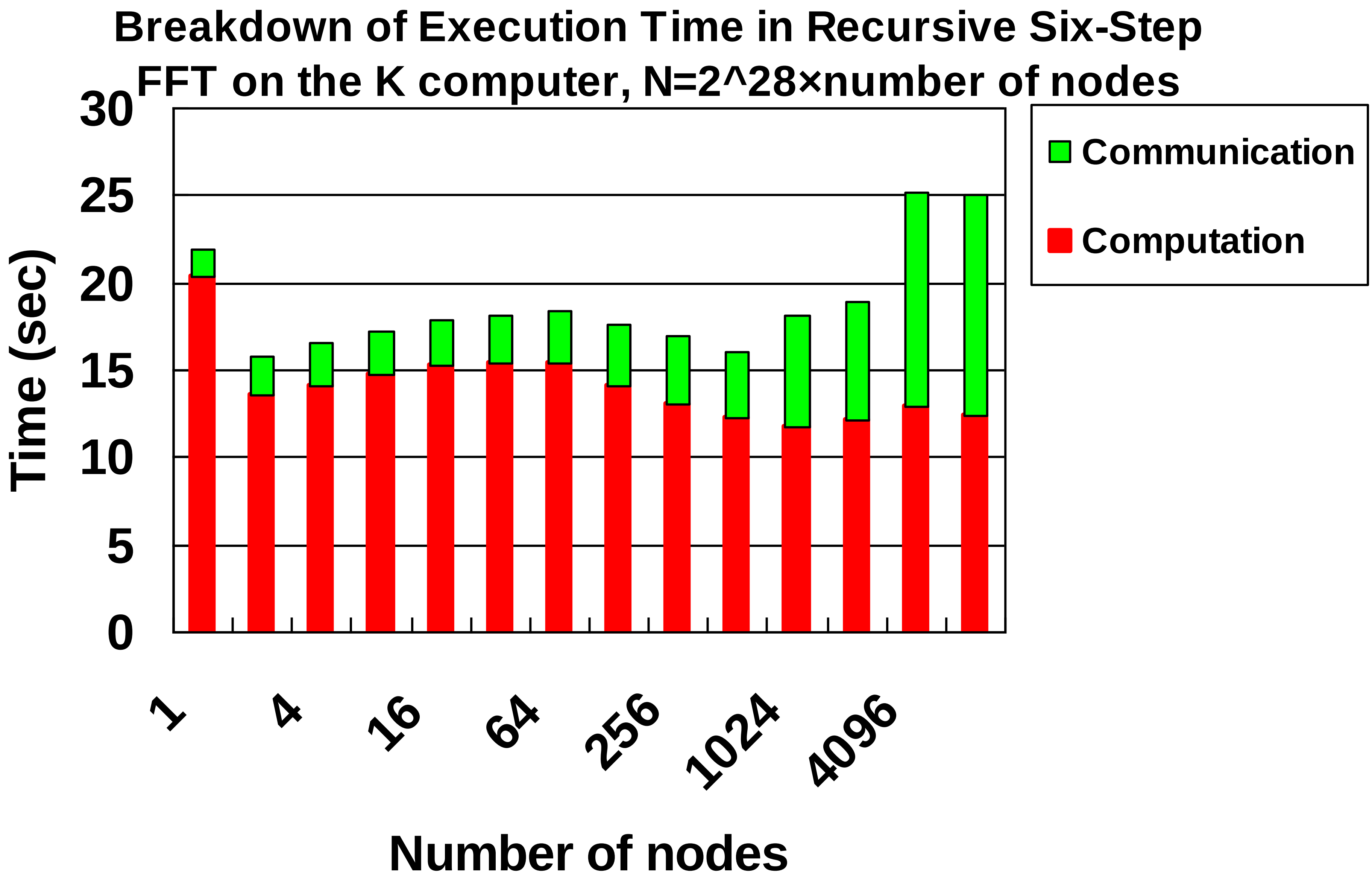
- With the multicolumn FFTs in the six-step FFT algorithm, the Stockham autosort FFT algorithm [Swarztrauber 84] works well until the $\sqrt{n}$ -point each column FFT exceeds the cache size.
- However, for extremely large FFTs (e.g., $n = 2^{40}$ -point FFT), each $\sqrt{n}$ -point column FFT is not small enough to fit into the L2 cache.
- When each $\sqrt{n}$ -point column FFT exceeds the cache size, the six-step FFT should be used.
- This means that we can recursively use the six-step FFT for each column FFT.

Performance Results

- To evaluate the implemented parallel 1-D FFT, we compared
  - Recursive six-step FFT-based parallel FFT
  - Six-step FFT-based parallel FFT
- Target machine: K computer
  - 82944 nodes, 16 GB per node, 128 GFlops per node, 1.27 PB total main memory, communication bandwidth 5 GB/s per node in each direction, and 10.6 PFlops peak performance.
  - We used 1 node to 8192 nodes.
  - A Tofu-optimized Message Passing Interface based on the Open MPI library was used.

Performance of Parallel 1-D FFTs on the K compuer, N=2^28×number of nodes





Breakdown of Execution Time in Recursive Six-Step FFT on the K computer, N=2^28×number of nodes
Time (sec)
30
25
20
15
10
5
0
Communication
Computation
1
4
16
64
256
1024
4096
Number of nodes

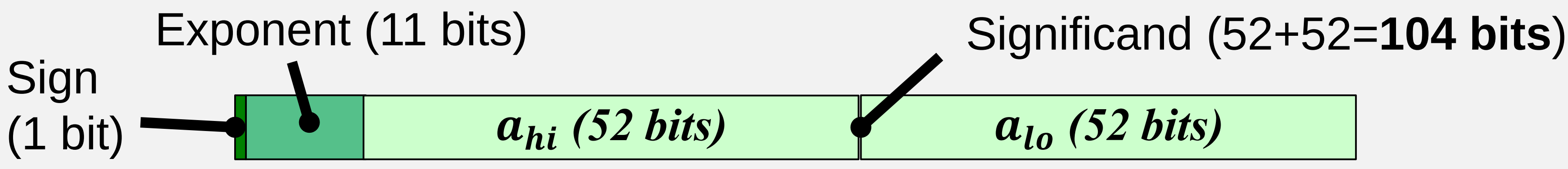
High Precision Arithmetic Operations

- Demand for high precision arithmetic operations
  - To compute ill-conditioned problems
  - Long-time and large-scale simulation: an accumulation of round-off error may become more serious problem
- Double-double (DD) type quadruple precision arithmetic libraries
  - DDFUN90 [Bailey], QD [Bailey et al.]
- Multiple precision arithmetic libraries
  - The GNU multiple precision arithmetic library (GMP)
  - MPFUN90 [Bailey], ARPREC [Bailey et al.]
- Extended precision BLAS
  - CPU: XBLAS [Li et al.], MBLAS [Nakata]
  - GPU: MBLAS (NVIDIA GPUs) [Nakata], Quadruple precision GEMM (AMD GPUs) [Nakasato 2011], Triple and quadruple precision AXPY, GEMV and GEMM (NVIDIA GPUs) [Mukunoki 2012]

Triple and Quadruple Precision Formats

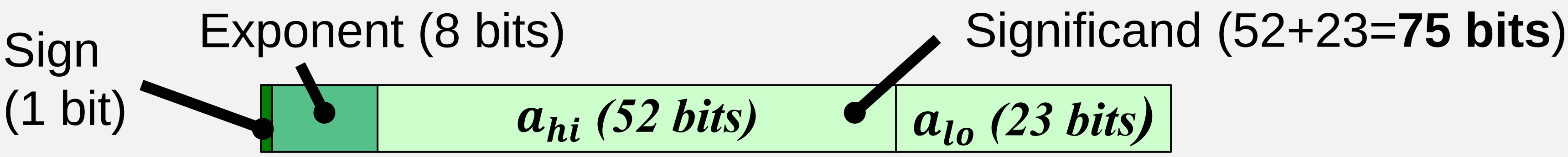
- DD (Double-Double) type quadruple precision represents one quadruple precision value $\boldsymbol{a}$ using two double precision values $\boldsymbol{a_{hi}}$ and $\boldsymbol{a_{lo}}$:

$$\boldsymbol{a} = \boldsymbol{a_{hi}} + \boldsymbol{a_{lo}}, \text{ where } |\boldsymbol{a_{lo}}| \leq \mathbf{0.5ulp}(\boldsymbol{a_{hi}})$$



- D+S (Double+Single) type triple precision represents one triple precision value $\boldsymbol{a}$ using one double precision value $\boldsymbol{a_{hi}}$ and one single precision value $\boldsymbol{a_{lo}}$:

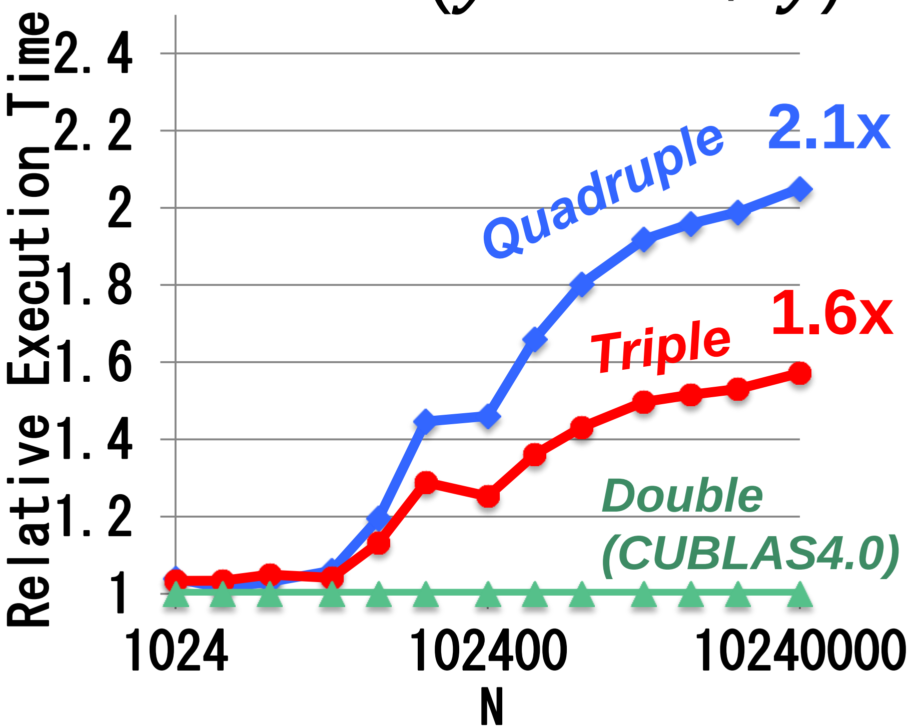
$$\boldsymbol{a} = \boldsymbol{a_{hi}} + \boldsymbol{a_{lo}}, \text{ where } |\boldsymbol{a_{lo}}| \leq \mathbf{0.5ulp}(\boldsymbol{a_{hi}})$$



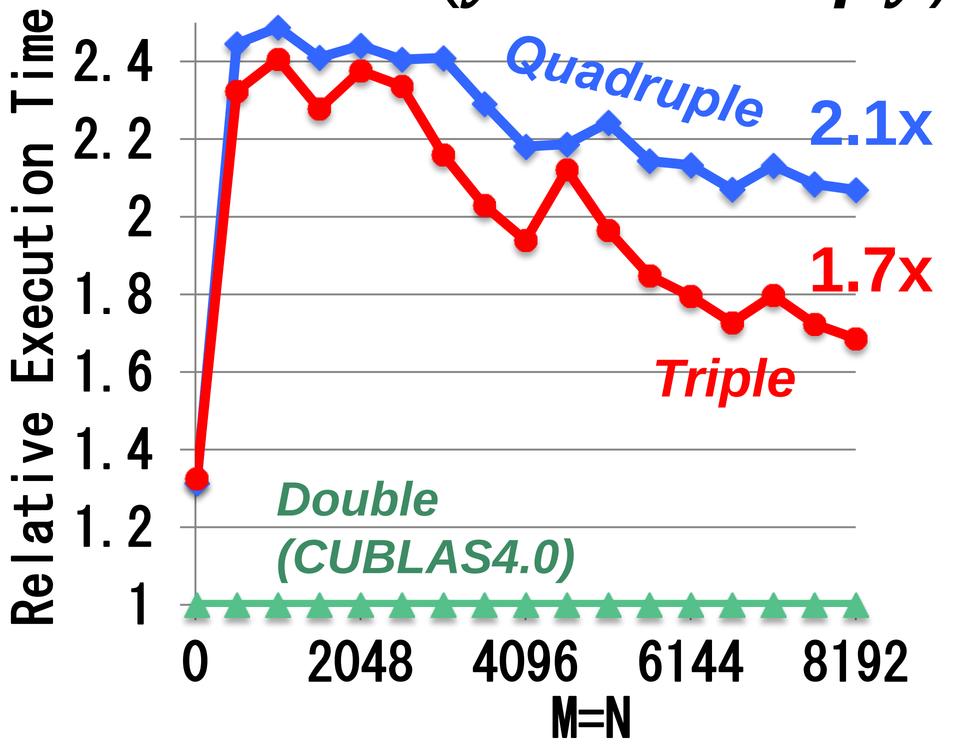
† Exponent is 8 bits: size of exponent depends on lower part's exponent

Relative Execution Time on Tesla C2050

AXPY ($y = \alpha x + y$)

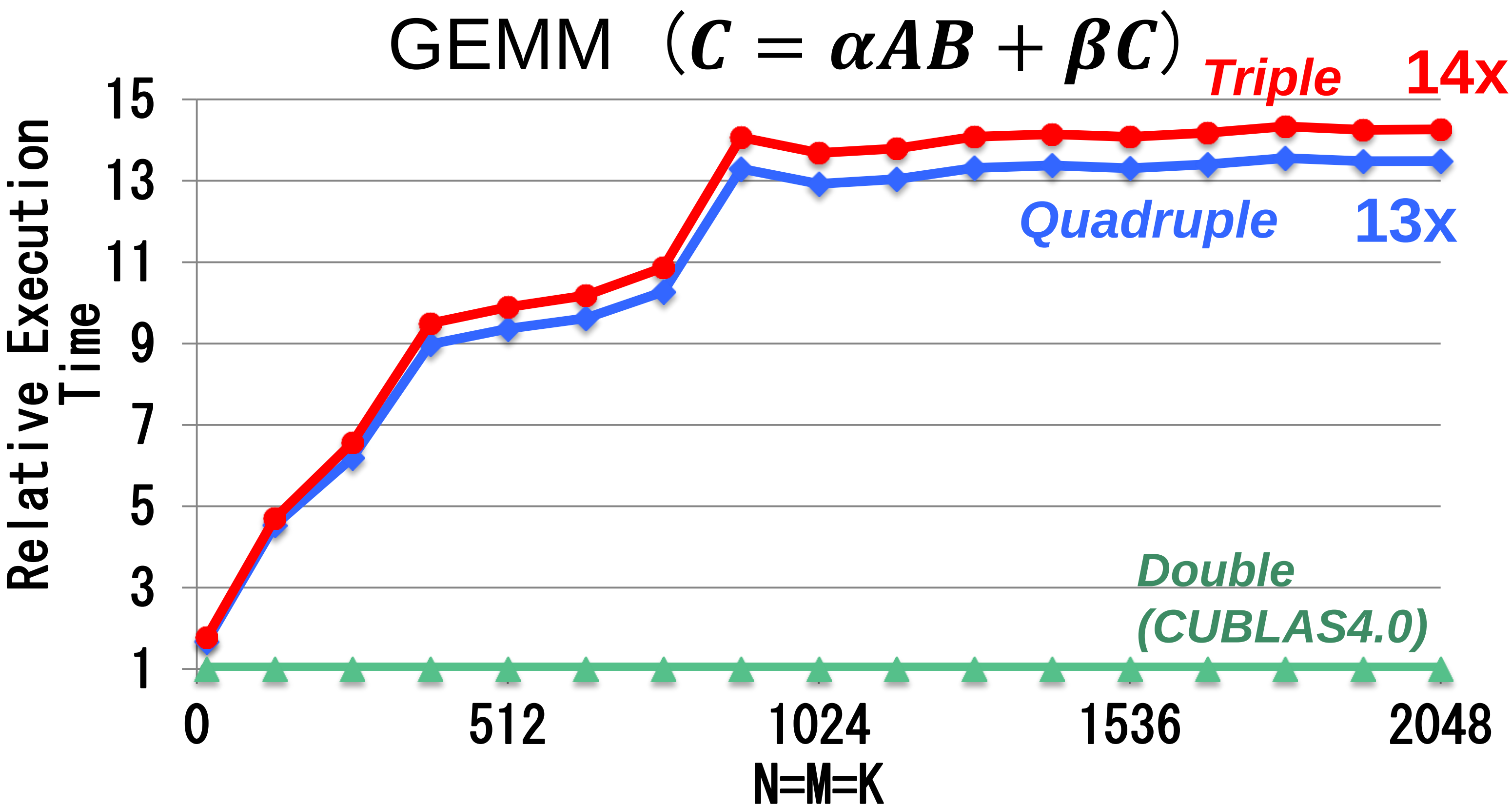


GEMV ($y = \alpha A x + \beta y$)



- Computation cost of triple and quadruple precision subroutines is 20x more than double precision subroutines in theory.
- But only 1.6-1.7x (triple) and 2.1x (quadruple) of double in practice.
- Triple and quadruple precision AXPY and GEMV are memory-bound on the GPU (evident from Bytes/Flop ratios of GPU and subroutines).

Relative Execution Time on Tesla C2050

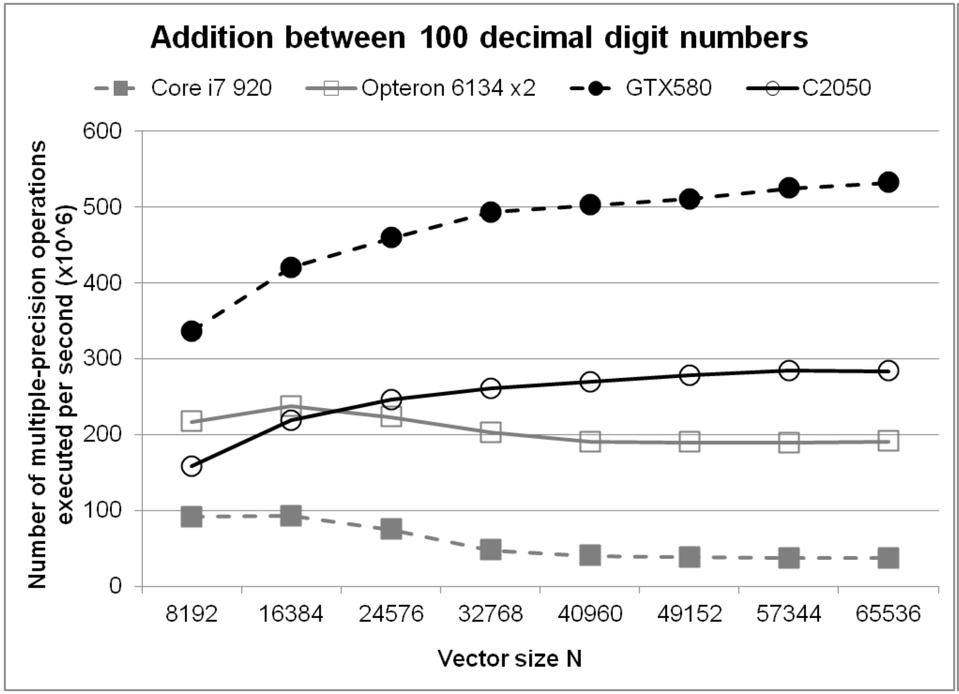


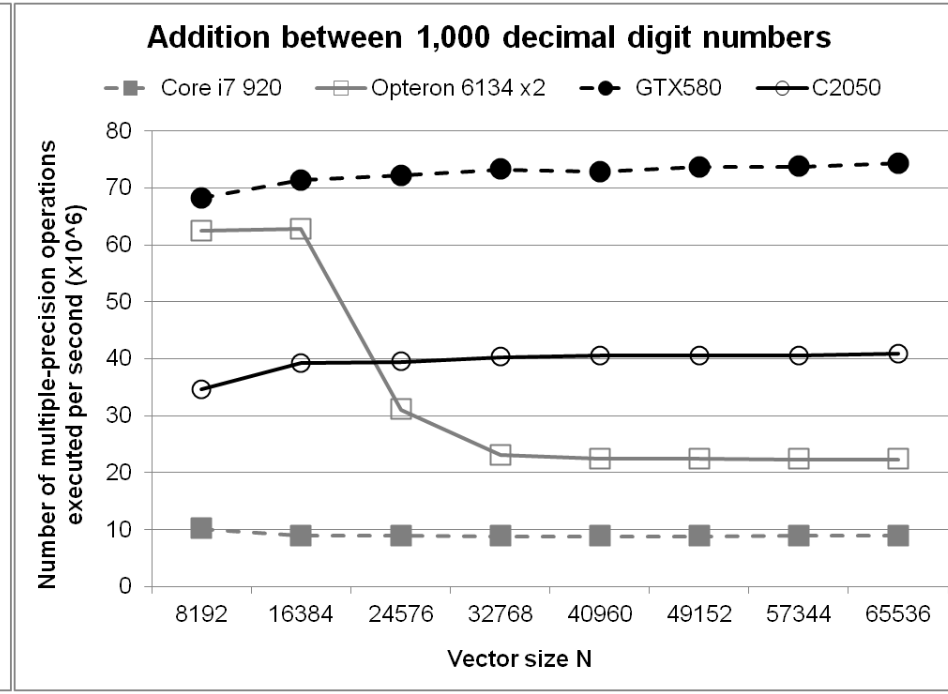
- GEMM is compute-bound in all precision on the GPU.
- Computation cost of DD-type operations is 20x more than double precision in theory, but only 13x slower in practice.

Overview of CUMP

- CUMP is a free library for arbitrary precision arithmetic on CUDA, operating on floating point numbers.
- It is based on the GMP, and its functions have a GMP-like regular interface.
- Three arithmetic operations (addition, subtraction, and multiplication) are currently available.
- Available at http://www.hpcs.cs.tsukuba.ac.jp/~nakayama/cump/

Performance Results for Elementwise Addition

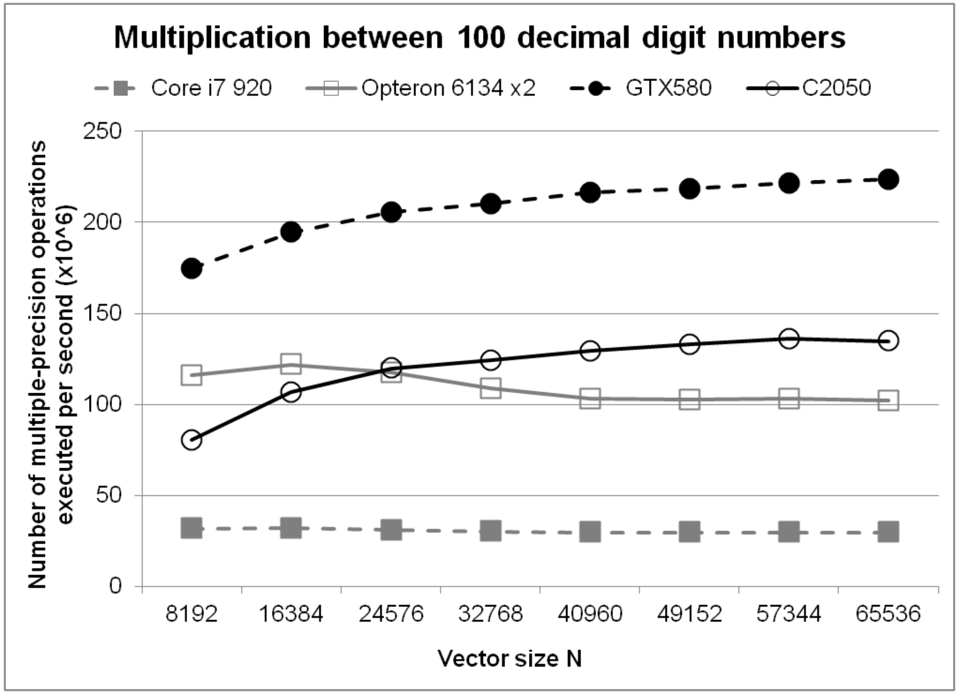


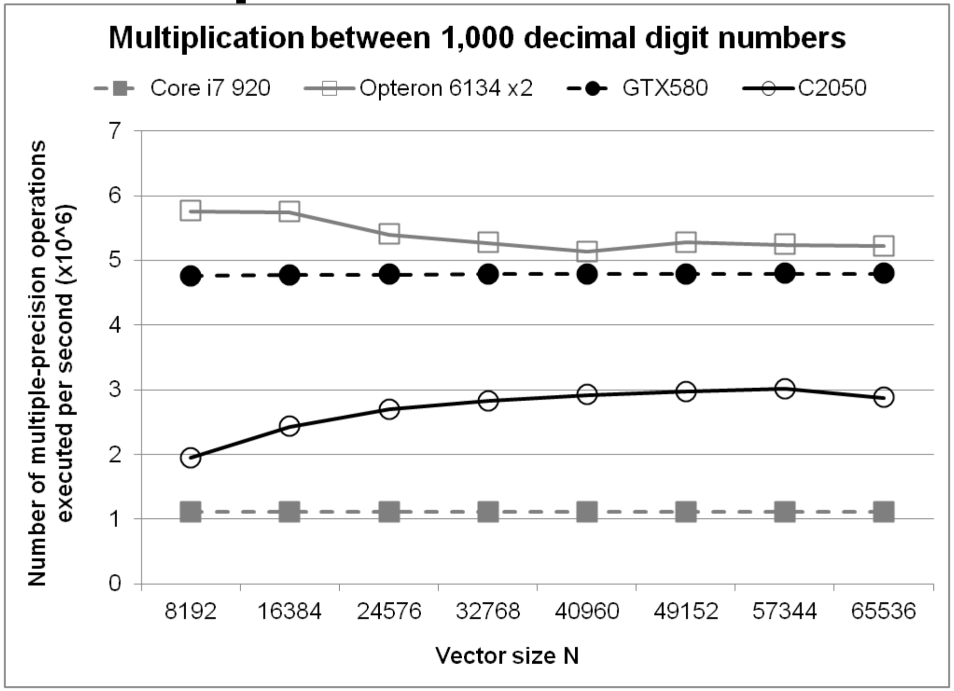


- For vector size N >= 24576, CUMP on GPUs (GTX580 and C2050) is faster than GMP on CPUs (Core i7 920 and Opteron 6134 x 2).

† Graphs courtesy of http://www.hpcs.cs.tsukuba.ac.jp/~nakayama/cump/

Performance Results for Elementwise Multiplication





- For 1,000 decimal digit numbers, GMP on CPU (Opteron 6134 x 2) is faster than CUMP on GPUs.
- CUMP does not support fast multiplication algorithms (e.g., Karatsuba, Toom-Cook and FFT).

† Graphs courtesy of http://www.hpcs.cs.tsukuba.ac.jp/~nakayama/cump/

Summary (1/2)

- We briefly introduced the FFTE library and performance results of parallel 1-D FFT on the K computer.
- The performance of the recursive six-step FFT-based parallel FFT remains at a high level even for larger problem sizes due to the recursive approach and the cache blocking.
- Global FFT on the K computer (82,944 nodes) achieved first place (205.9 TFlops) in the 2012 HPC Challenge Class 1 Awards.

Summary (2/2)

- High precision arithmetic operations will become increasingly necessary for emerging Exa-scale computing era.
- Accelerators (GPUs and MICs, etc.) are a good candidate for high precision arithmetic operations.
- Triple precision is useful for memory-bound operations, in cases where quadruple precision is not required, but double precision is not sufficient.