Performance Evaluation and Analysis of A64FX many-core Processor for the Fiber Miniapp Suite

Miwako Tsuji

Programming Environment Research Team

RIKEN Center for Computational Science

Note

- Our experiments had been conducted during the early access program of Fugaku. The results can differ from the ones obtained in the official operation period

Introduction

- Supercomputer Fugaku
  - has started full operations in March 2021
  - No.1 of Top 500 in Jun and Nov 2020
  - uses A64FX Arm based processors
- The performance evaluation of the A64FX using Fiber Miniapp suite
  - The fiber Miniapp suite is a set of mini-applications extracted from real-applications on K-computer
  - developed by the application feasibility study project of post-K (Fugaku) computer
  - includes some of the target applications of the supercomputer Fugaku
- Performance analysis and tuning examples
- The results of the performance evaluations are available for a wide-ragne of systems, such as K/FX10/FX100/OPF/BlueWaters/HA-PACS/<u>SKL/TX2</u>
  - “A new sustained system performance metric for scientific performance evaluation”
  - DOI: 10.1007/s11227-020-03545-y

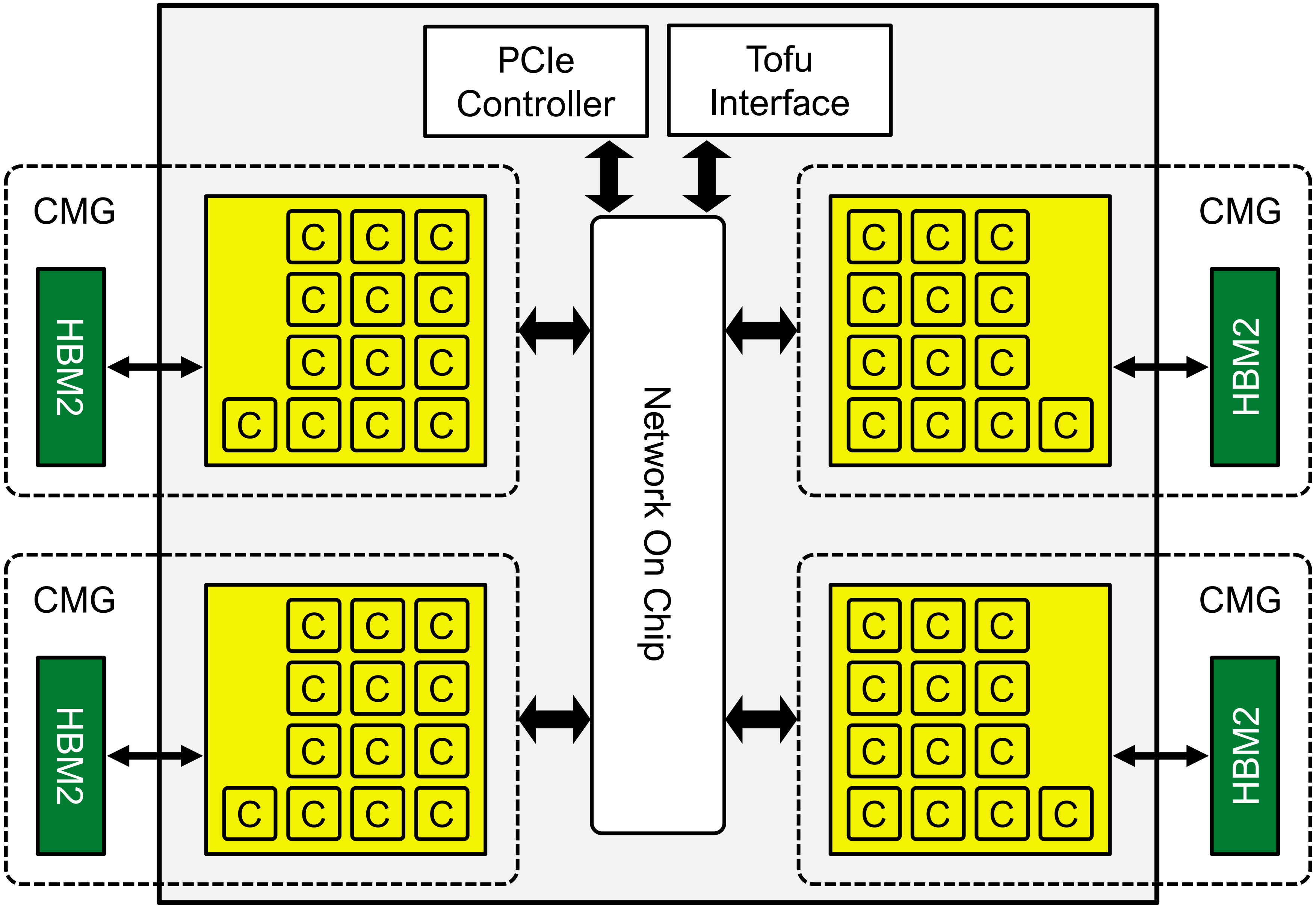
A64FX

- A processor developed by Fujitsu based on Armv8.2-A Scalable Vector Extension (SVE)
- Supercomputer Fugaku, and its consumer versions such as FX700, FX1000

A64FX: Node Overview

C: Core
CMG: Core Memory Group
HBM: High Bandwidth Memory

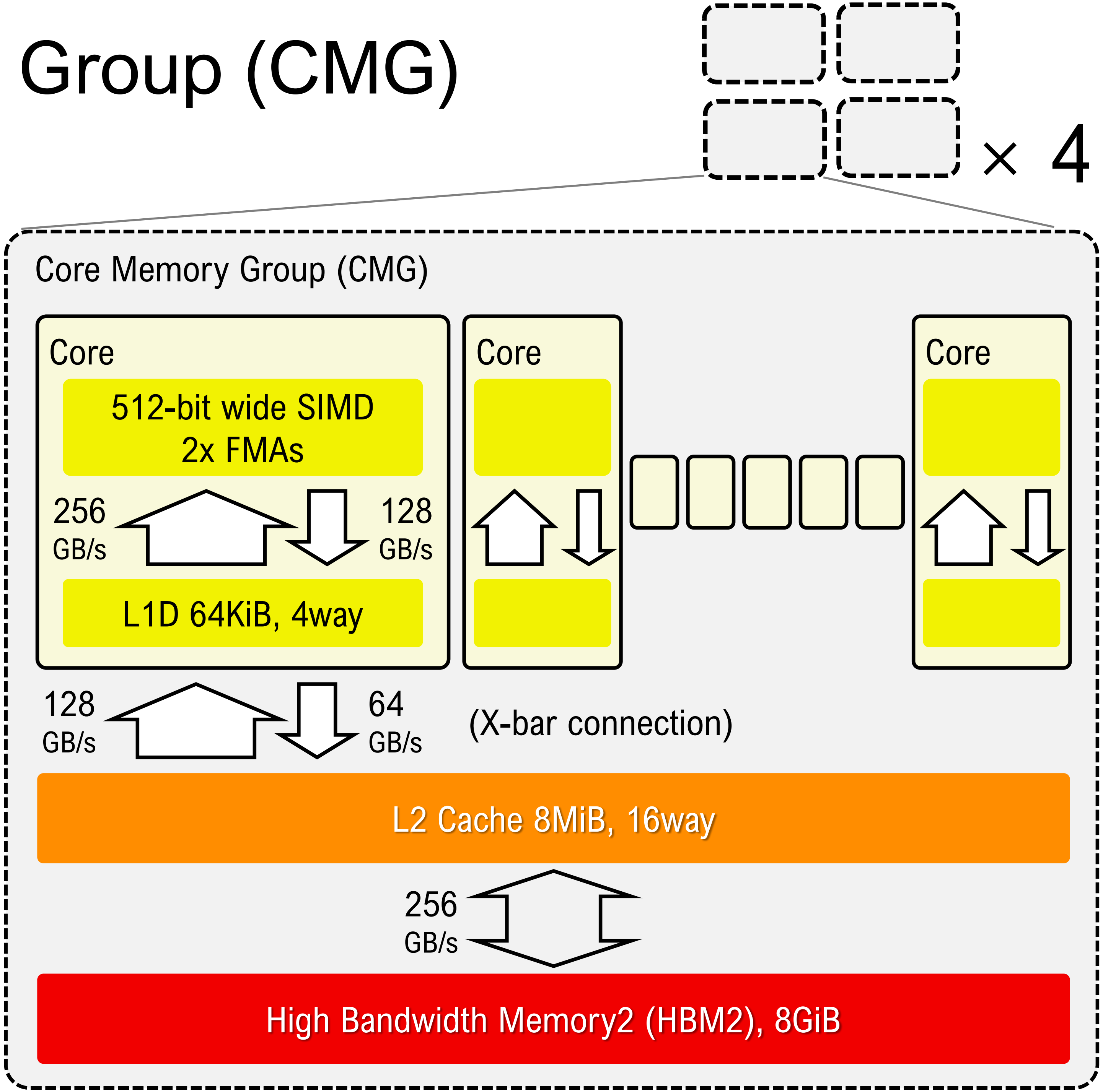
- 4 CMG in a Node
- 12+1 cores in a CMG
  - an assistant core in each CMG
- FP64/FP32/FP16
- HBM2 32GiB/Node
  - 8GiB/CMG
  - 1024 GB/s
- Leading-edge Si-technology (7nm FinFET), low power logic design (approx. 15 GF/W (dgemm)),



A64FX: Core Memory Group (CMG)

- 12+1 core in a CMG
- 2x 512-bit wide SIMD (SVE), 4x ALUs, Predicate Operation
- 2x 512-bit wide SIMD load or 512-bit wide SIMD store
- L1D cache (/core) : 64 KiB, 4 ways, “Combined Gather” on L1
- L2D cache (/CMG): 8MiB
    - X-bar connection in a CMG
- 4 CMGs support cache coherency by ccNUMA with on-chip directory (256 GB/s x 2 for inter-CMGs)
- Memory controller for HBM2

※ L1/L2 cache bandwidths are for 2.0GHz



Affinity and Allocation: Thread Affinity

- FLIB_CPU_AFFINITY (mainly used in this presentation)
  - specify core-ids for thread 0, 1, 2, … explicitly
    - FLIB_CPU_AFFINITY=”12, 14, 16, …, 58”
  - specify the period of core-ids and its interval
    - FLIB_CPU_AFFINITY=“12-59:**2**”
- FLIB_CPUBIND (I never used..)
  - chip_pack: allocate threads in a single CMG as possible
  - chip_unpack: allocate threads in different CMGs
  - off: don’t use FLIB_CPUBIND, but others such as OMP_*
- GOMP_*, OMP_* can be used (“socket” indicates CMG)
  - OMP_PLACES=“sockets” may assign threads in a core

- Some environmental variables are available only for a binary linked with Fujitsu’s OpenMP library

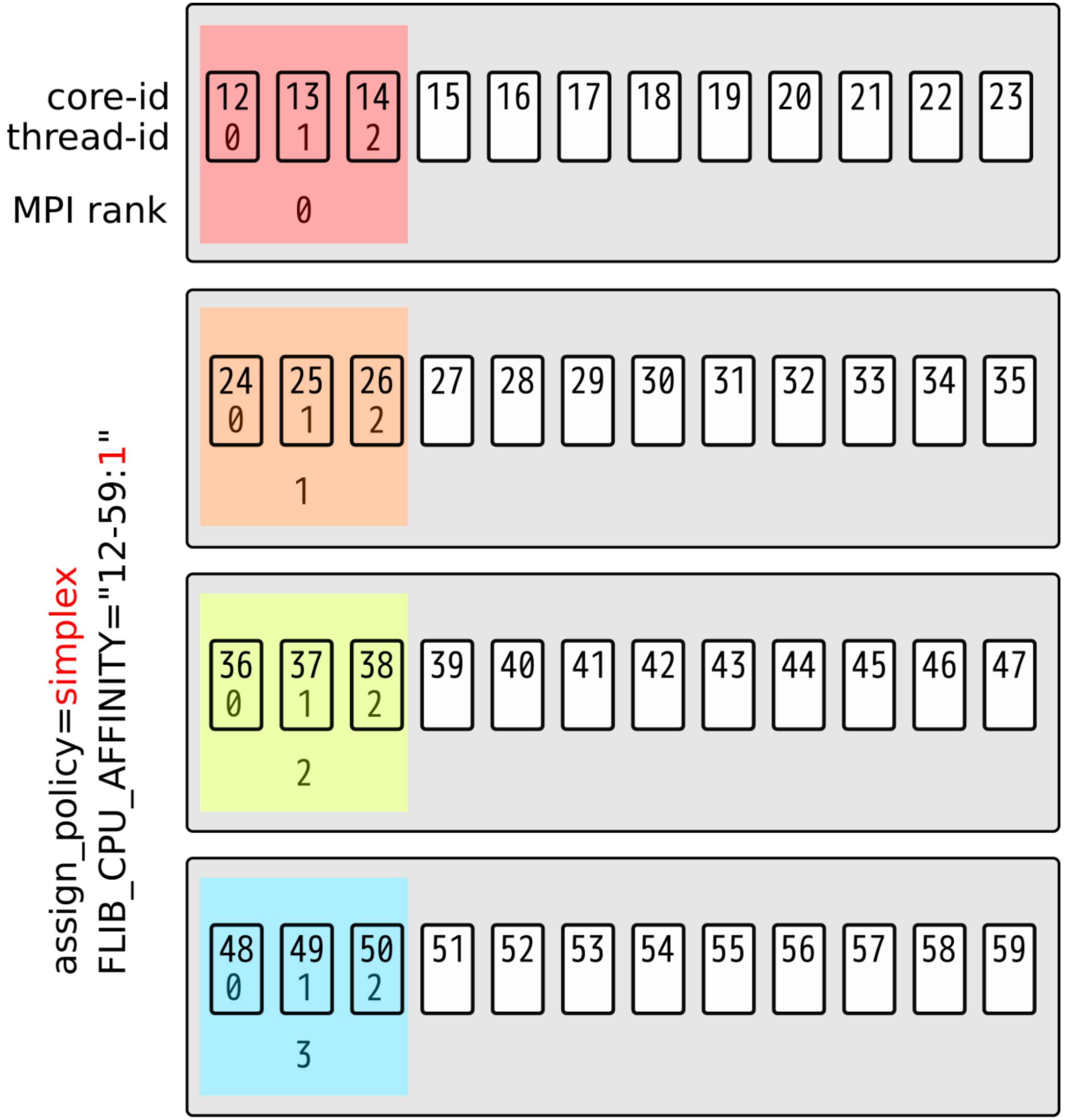
Core ID numbers are :
{12, 13, …, 23},
{24, 25, …, 35},
{36, 37, …, 47},
{48, 49, …, 59}
NOT start from 0

Affinity and Allocation: Process allocation

- MPI processes can be allocated to CMGs by using the environmental variable "**OMPI_MCA_plm_ple_numanode_assign_policy**"
  - simplex: The processes are allocated to the CMG exclusively
  - share_cyclic: The processes are allocated to the CMG with sharing with other processes. The processes are sequentially allocated in **different** CMGs
  - share_band: The processes are allocated to the CMG with sharing with other processes. The processes are sequentially allocated in a **same** CMG

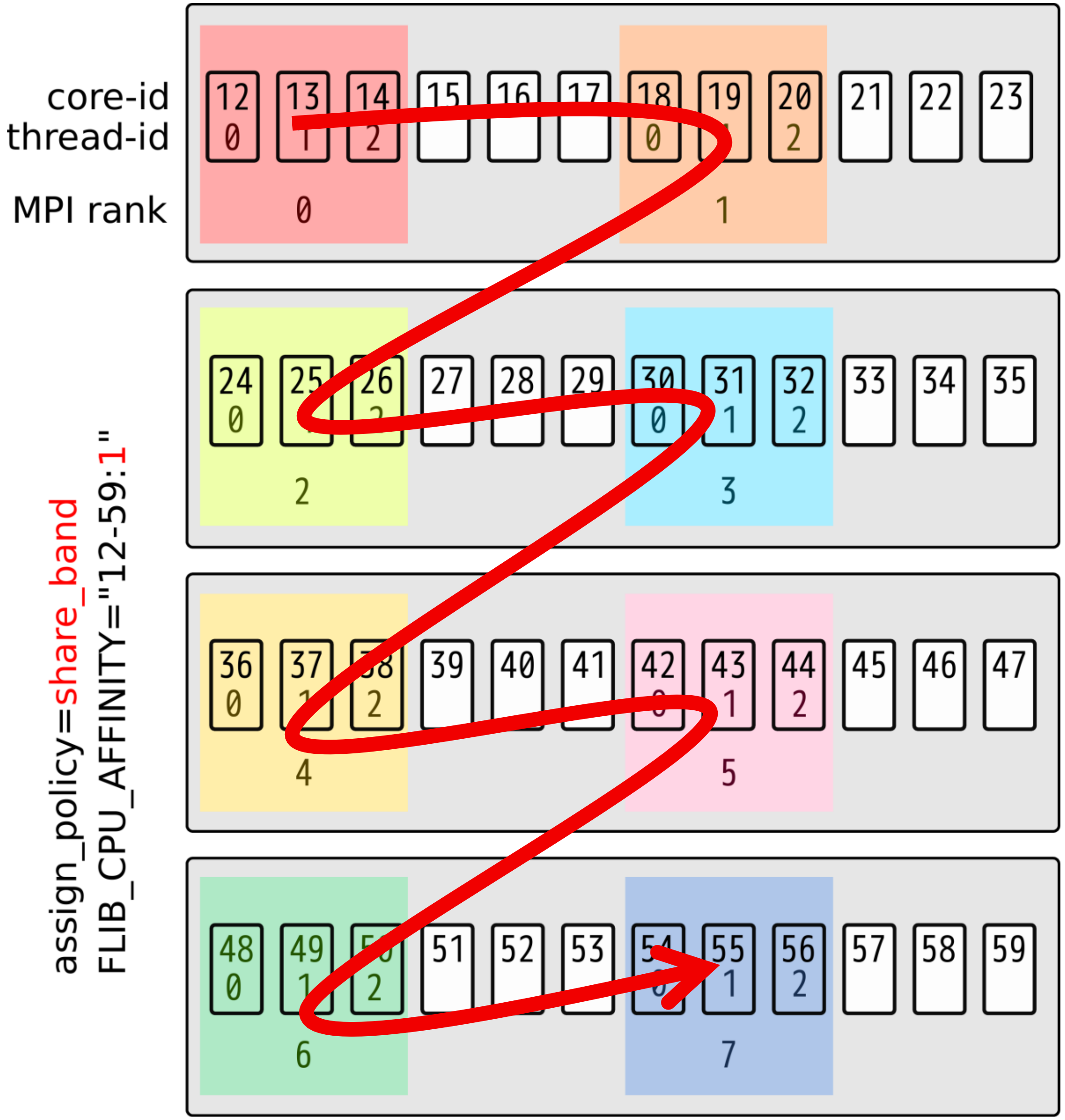
Affinity and Allocation: Example

of MPI Processes=4
OMP_NUM_THREADS=3
allocation_policy=simplex
FLIB_CPU_AFFINITY="12-59:1"



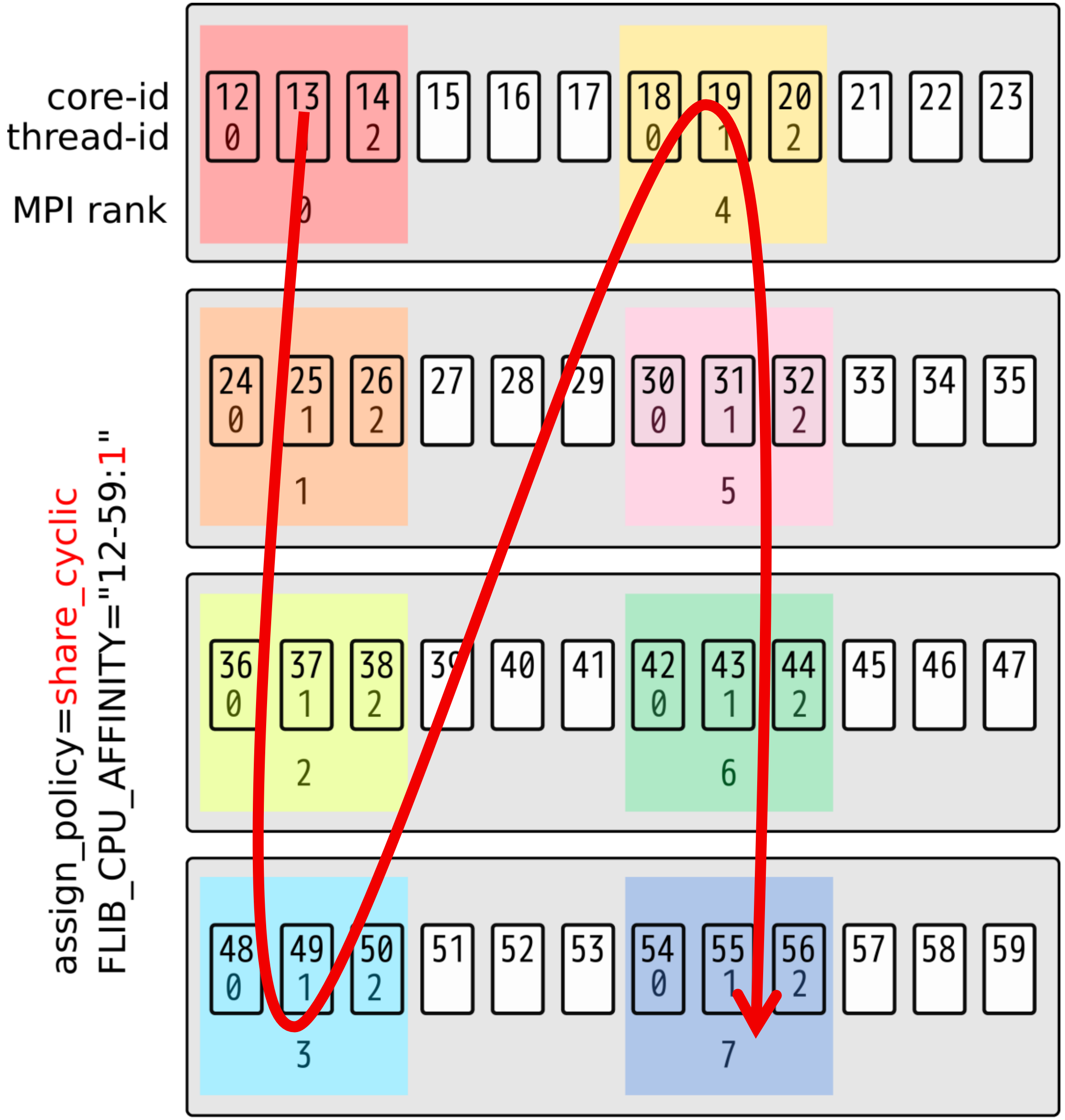
Affinity and Allocation: Example

of MPI Processes=8
OMP_NUM_THREADS=3
allocation_policy=share_band
FLIB_CPU_AFFINITY="12-59:1"



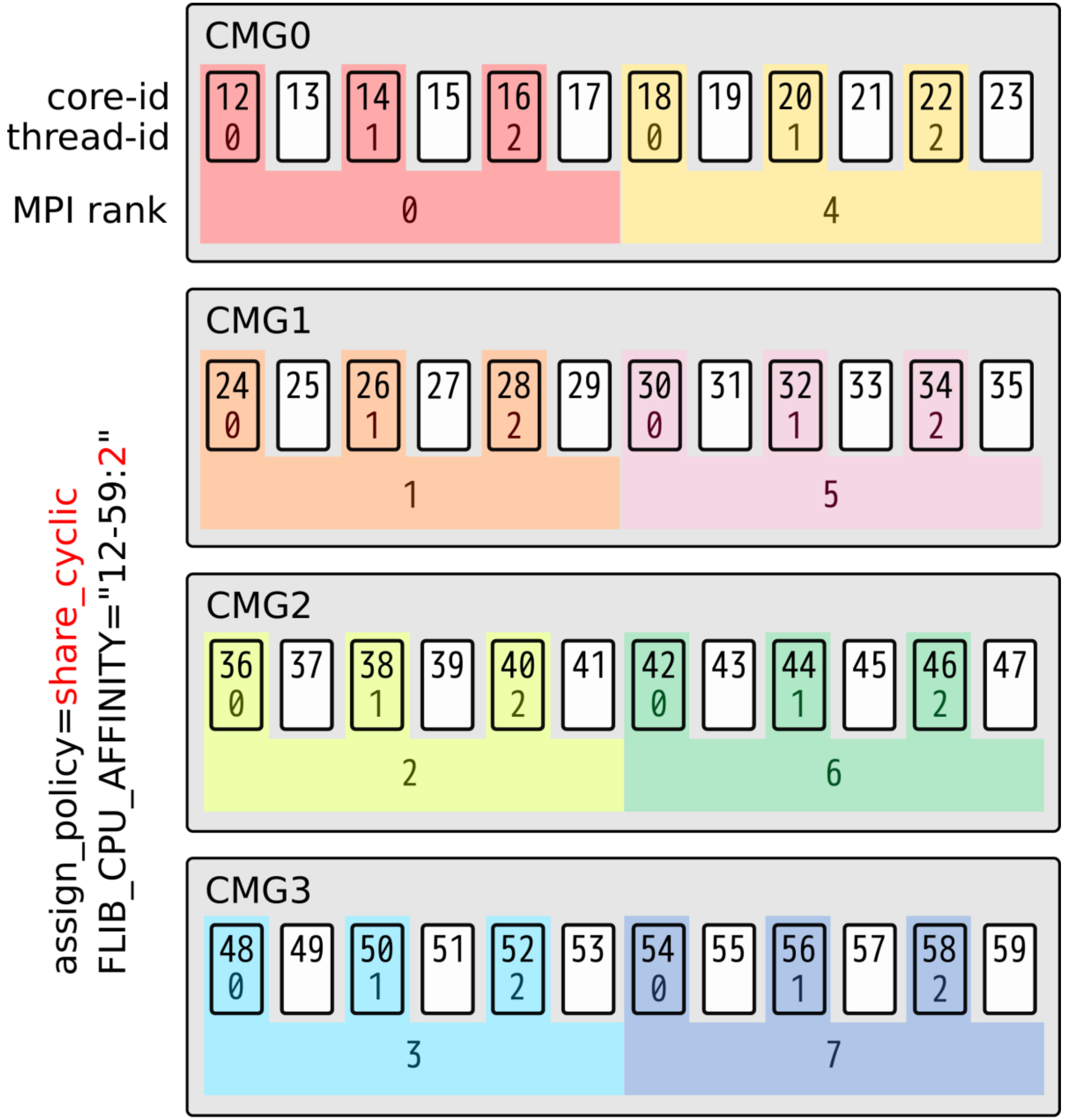
Affinity and Allocation: Example

of MPI Processes=8
OMP_NUM_THREADS=3
allocation_policy=share_cyclic
FLIB_CPU_AFFINITY="12-59:1"



Affinity and Allocation: Example

of MPI Processes=8
OMP_NUM_THREADS=3
allocation_policy=share_cyclic
FLIB_CPU_AFFINITY="12-59:**2**"



Fiber Miniapp Suite

- A suite of mini apps derived from the full-scale applications for the future computational science challenges
- Originally developed and used on high-end machines such as K computer

Application	Area	Characteristics
CCS-QCD	Quantum chromodynamics	Structured grid Monte Carlo
FFVC	Thermo-fluid analysis	3 dimensional cavity flow
NICAM-DC	Climate	Structured grid stencil
mVMC	Material Science	Many variable variational Monte Carlo
~~NGS-Analyzer~~	~~Genome sequence analysis~~	~~Multi task work flow~~
NTChem	Quantum chemistry	Molecular orbital method
FFB	Thermo-fluid analyses	Finite element method, unstructured grid

Experimental Environments

	TX2 partition @CEA	SKL partition @CEA	A64FX (富岳)
CPU	Marvell ThunderX2	Intel Xeon Platinum 8168	Fujitsu A64FX
# of cores / socket	32	24	48
# of socket / node	2	2	1
SMT / core	2	2	1
CPU GHz	2.2 GHz	2.7 GHz	2.0 GHz
Peak performance / node	1124.4 GFlops	4147.2 GFlops	3072.0 GFlops
Memory	DDR4	DDR4	HBM2
Capacity / node	256 GB	192 GB	32 GB
Bandwidth / node	340 GB/s	255 GB/s?	1024 GB/ss
# of nodes	28	22	158,976
Compiler	arm-compiler 19.0.0	Intel/17.0.6.256	Fujitsu compiler 4.3.0a
MPI library	openmpi 2.0.4	mpi/openmpi/2.0.4	(tcsds-1.2.28a)

Experimental Environments

	TX2 partition @CEA	SKL partition @CEA	A64FX (富岳)
CPU	Marvell ThunderX2	Intel Xeon Platinum 8168	Fujitsu A64FX
# of cores / socket	32	24	48
# of socket / node	2	2	1
SMT / core			1
CPU GHz			2.0 GHz
Peak performance / node	1124.4		72.0 GFlops
Memory			HBM2
Capacity / node	256 GB	192 GB	32 GB
Bandwidth / node	340 GB/s	255 GB/s?	1024 GB/ss
# of nodes	28	22	158,976
Compiler	arm-compiler 19.0.0	Intel/17.0.6.256	Fujitsu compiler 4.3.0a
MPI library	openmpi 2.0.4	mpi/openmpi/2.0.4	(tcsds-1.2.28a)

Note that
a node of A64FX is **a** processor,
but **two** processors for TX2/SKL

Experimental Environments

- All codes are “as-is”
- Fujitsu/Intel/Arm compilers
- General compile options have been used, such as –Kfast, -O3, …

- To compare different systems, the best results over different numbers of processes, threads, affinity policies have been adopted
  - The number of nodes can be different due to the memory size restriction and other reasons
  - The numbers of processors in a node differ
    - 2 SKL/TX2 processors in a node, a single A64FX processor in a node

CCS-QCD

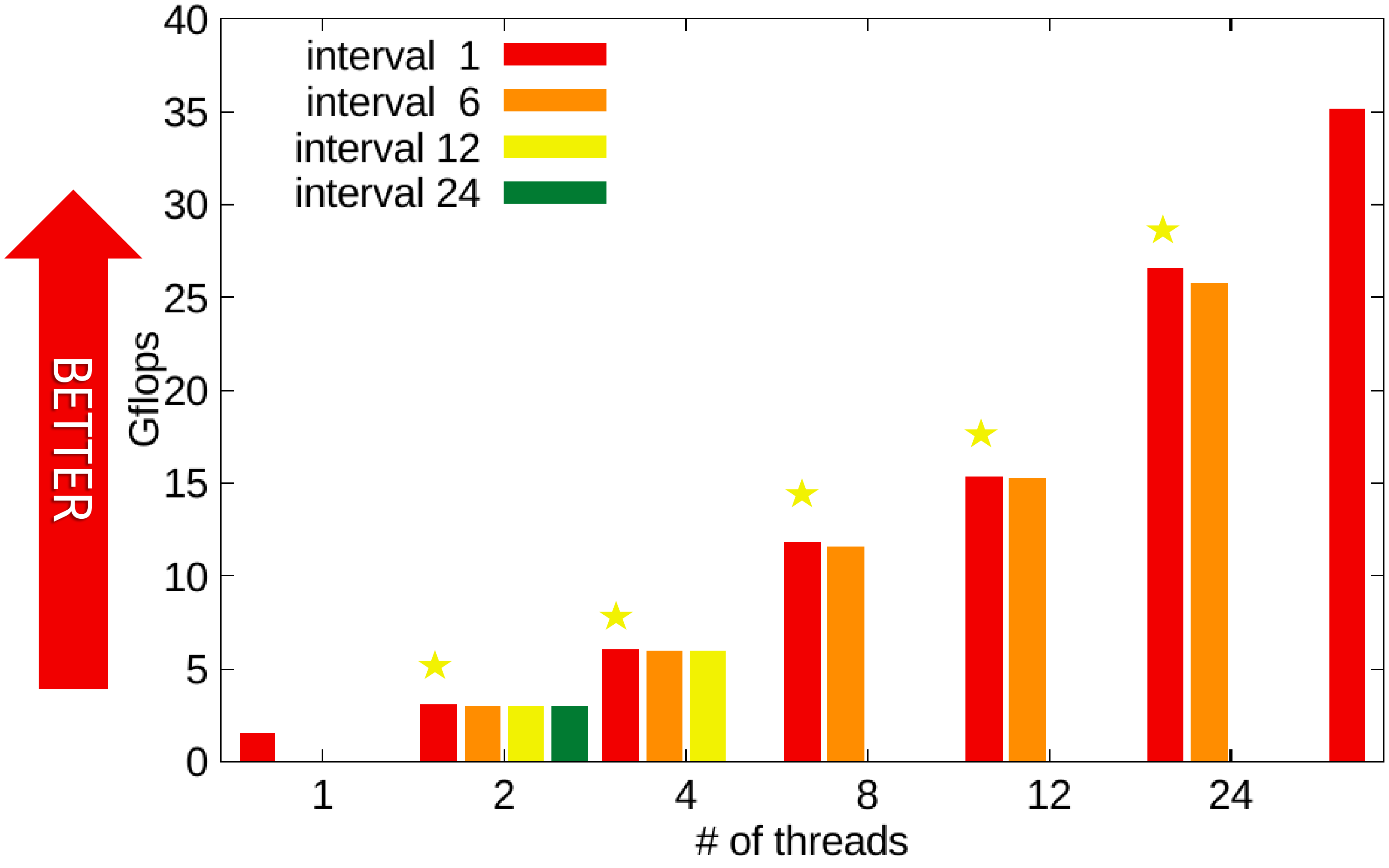
- This program benchmarks the performance of a linear equation solver with a large sparse coefficient matrix appearing in a lattice QCD problem.
- Memory bound
  - F/B 0.64 (measured in FX10 cluster, 2014*)

	class1	class2
Lattice	8x8x8x32	32x32x32x32
# of MPI processes	1	64 (4x4x4)
A64FX nodes	1 (48cores)	8 (384cores) / 16 (768cores)

*Kobayashi et.al
Swopp2014

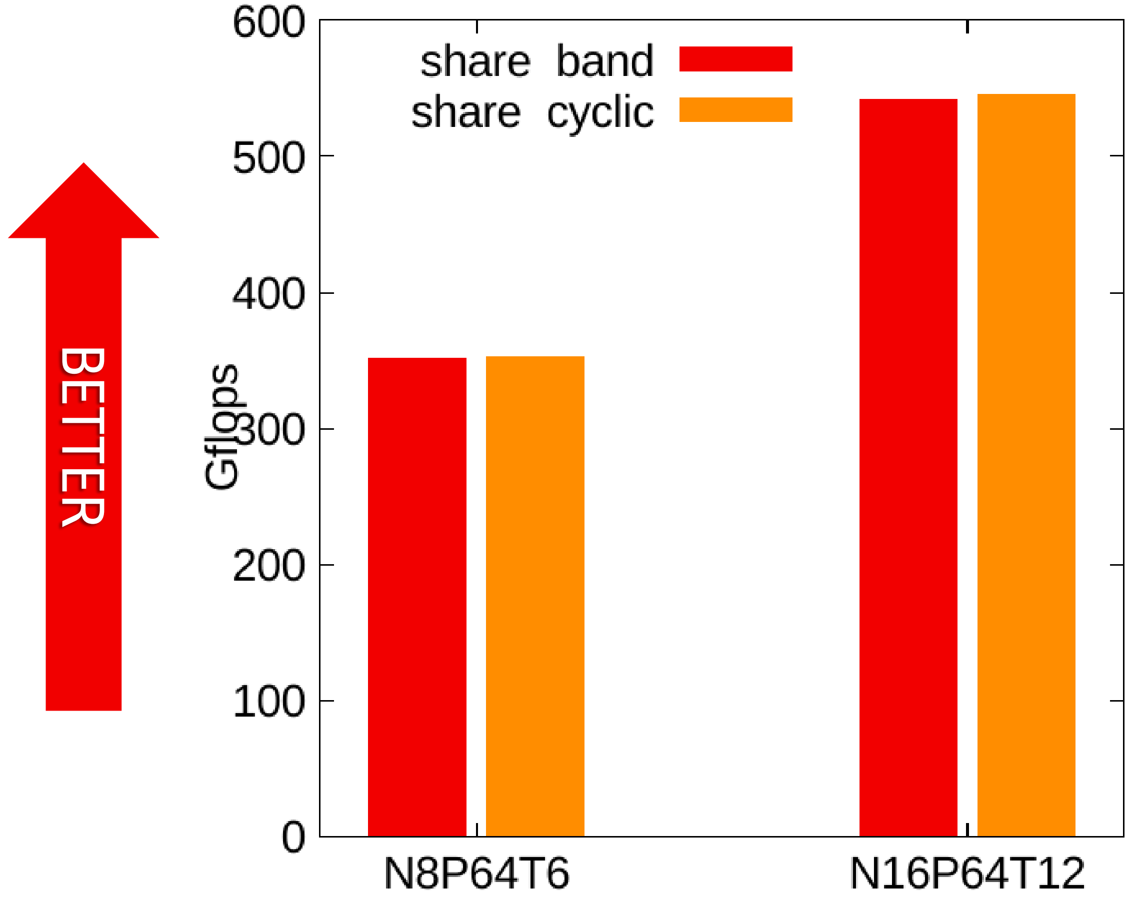
CCS-QCD Class1 1-node

- Class1, 1MPI, 1-48threads, different thread intervals
- “Compact” thread allocation is always better
- Good scalability up to 48 cores
  - First touch
  - export XOS_MMM_L_PAGING_POLICY=demand:demand:demand



CCS-QCD Class2 8, 16 nodes

- 8 Nodes, 64 MPI Proesses, 6 Threads
- 16 Nodes, 64 MPI Processes, 12 Threads
- Thread interval is always "**:1**"
- MPI Process affinity is set to
  - share_band
  - share_cyclic <= Better



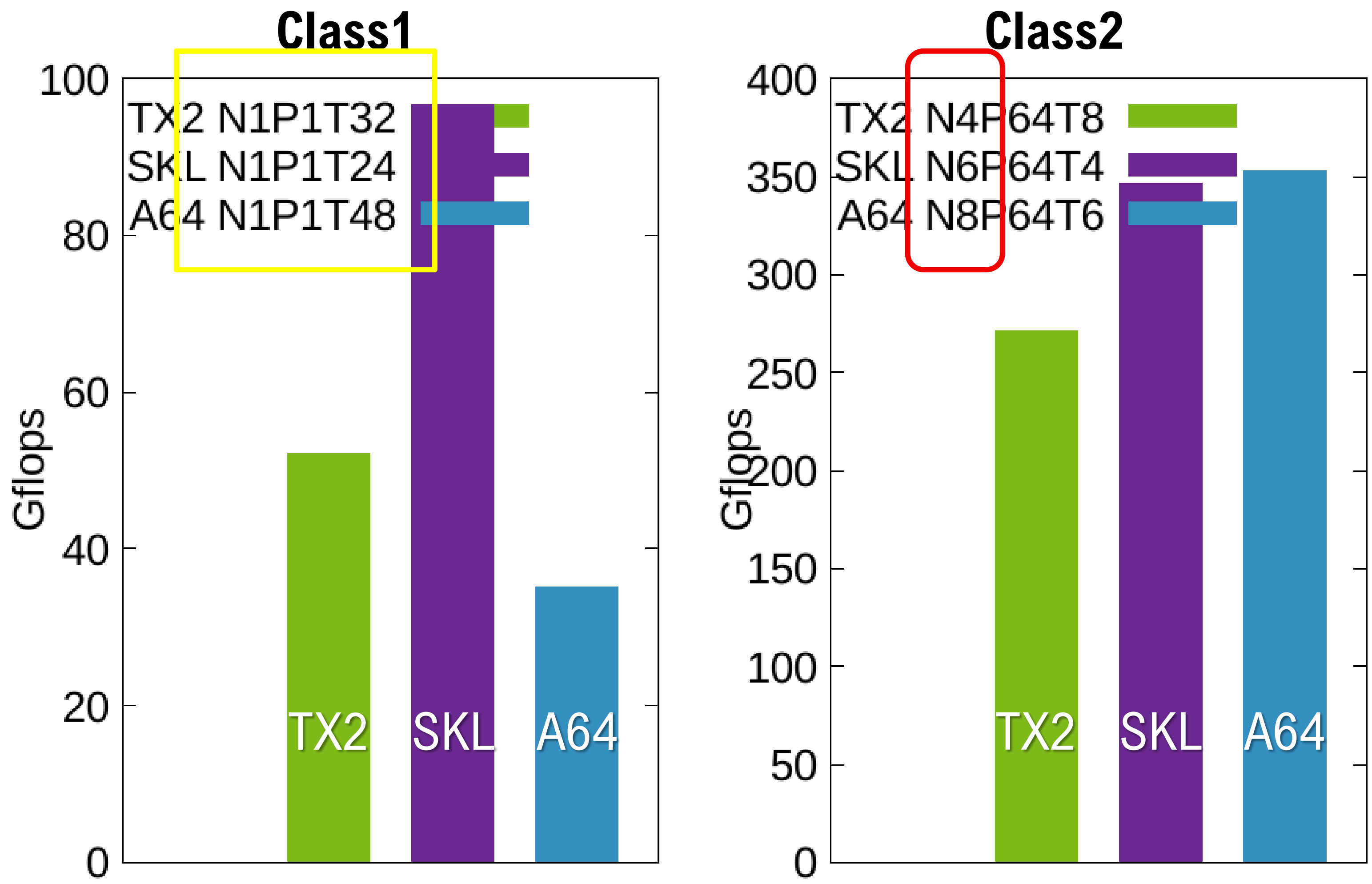
CCS-QCD class1 and class2

- For the Class1, SKL (**single socket**), TX2 (**single socket**) are better than A64FX
- For the Class2, SKL (6nodes, 12sockets) and A64FX (8nodes, 8scokets) are comparable

- The performance of A64FX is not good for class1

- For class2, SLK and A64FX are comparable

※ The number of nodes, and frequencies are different



CCS-QCD: Performance Analysis

Class1 12threads	% of SIMD	% of SVE (of SIMD)	fp op wait (sec)	total (sec)
as-is	40.07	24.16	0.240	0.636

- Only 9.6 % (40.07*24.16) of operations are SIMDized for SVE

Hand unrolled loop has bee rerolled

```
do ix=1,NX; do iy=1,NY; do iz=1,NZ
..
do itb=1-ieoxyz,NTH-ieoxyz
  ic=1
  yt(ic,3) =
  + 2.0d0*(ue_t(ic,1,itb,iz,iy,ix,4)*
      yo_t(1,3,itb+itbup,iz,iy,ix)
     +ue_t(ic,2,itb, iz,iy,ix,4)*
      ...
  ic=2
```

```
do ix=1,NX; do iy=1,NY; do iz=1,NZ
..
do itb=1-ieoxyz,NTH-ieoxyz
  do ic=1,3
   yt(ic,3) =
   + 2.0d0*(ue_t(ic,1,itb,iz,iy,ix,4)*
     yo_t(1,3,itb+itbup,iz,iy,ix)
    +ue_t(ic,2,itb, iz,iy,ix,4)*
     ...
  end do
```

- 30% of exec time is fp operation wait
    - Some options which affect instruction scheduling have been investigated
    - -Kassume=shortloop: control optimization assuming the loop is “short”
    - -Kswp_policy=small: control optimization assuming a small loop such as low register pressure
    - -Kswp_policy=large: control optimization assuming a large loop

CCS-QCD: Performance Analysis

Class1 12threads	% of SIMD	% of SVE (of SIMD)	fo op wait (sec)	total (sec)
as-is	40.07	24.16	0.240	0.636
tuned (best case)	41.82	**41.82**	0.182	0.566

	(sec)	-	-Kassume =shortloop	-Kswp_policy =small	-Kswp_policy =large
class1	as-is	$\boxed{.2805}$	.2522	.2855	.2652
	tuned	.2656	$\boxed{\textbf{.2390}}$	.2706	.2663
class2		1.798	1.694	1.802	1.794
	tuned	1.777	**1.613**	1.790	1.774

15% improved

※ 4 loops can be re-rolled, but only 1 has done in this experiment, since re-rolling other 3 loops induced invalid results ☹

swp_policy option does not improve the performance
The default (swp_policy=auto) should be appropriate

FFVC

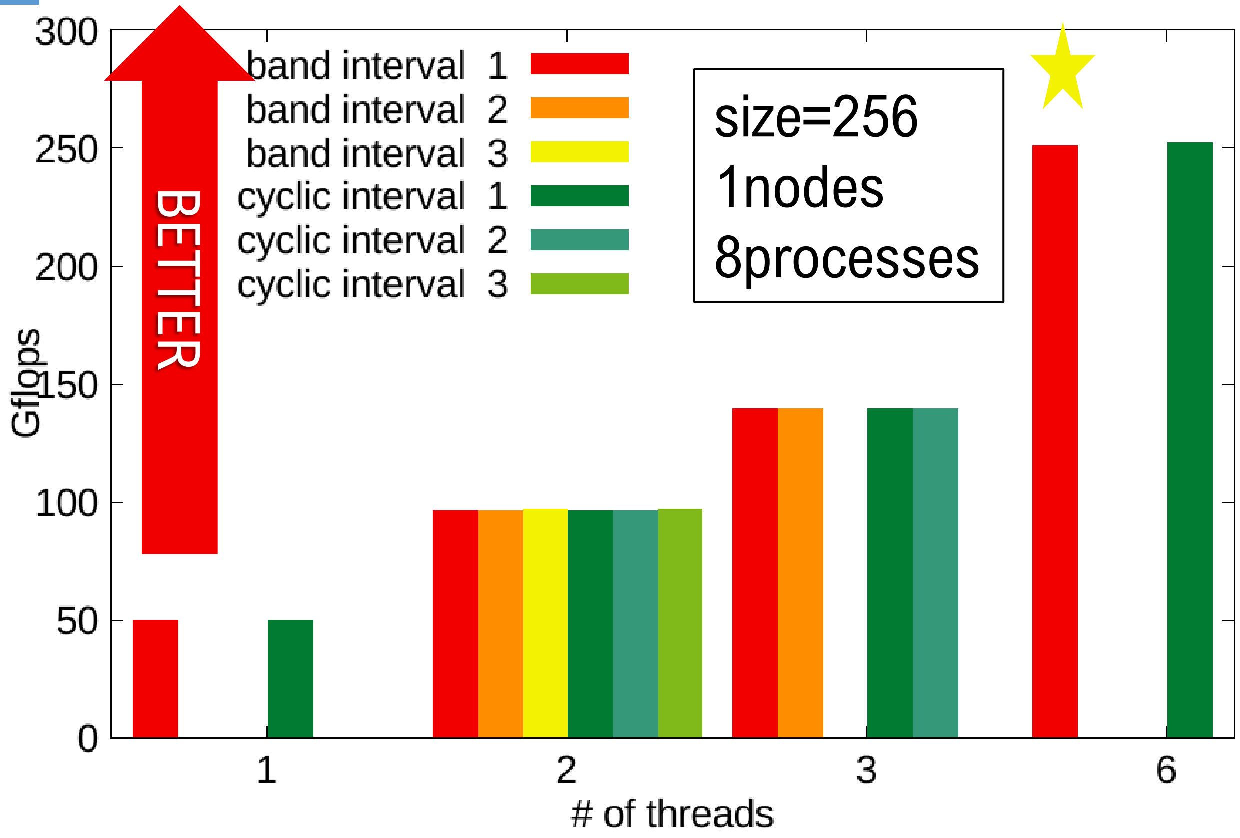
- 3D unsteady thermal flow of the incompressible fluid
  - solves the 3D incompressible Navier-Stokes equation in the even-spaced grid point orthogonal space using finite volume method
  - 1st order explicit Euler scheme for time integration
  - 3rd order MUSCL scheme for convection term
  - Strided memory access dual-colored SOR scheme for iterative solver
  - 1.47 F/B *

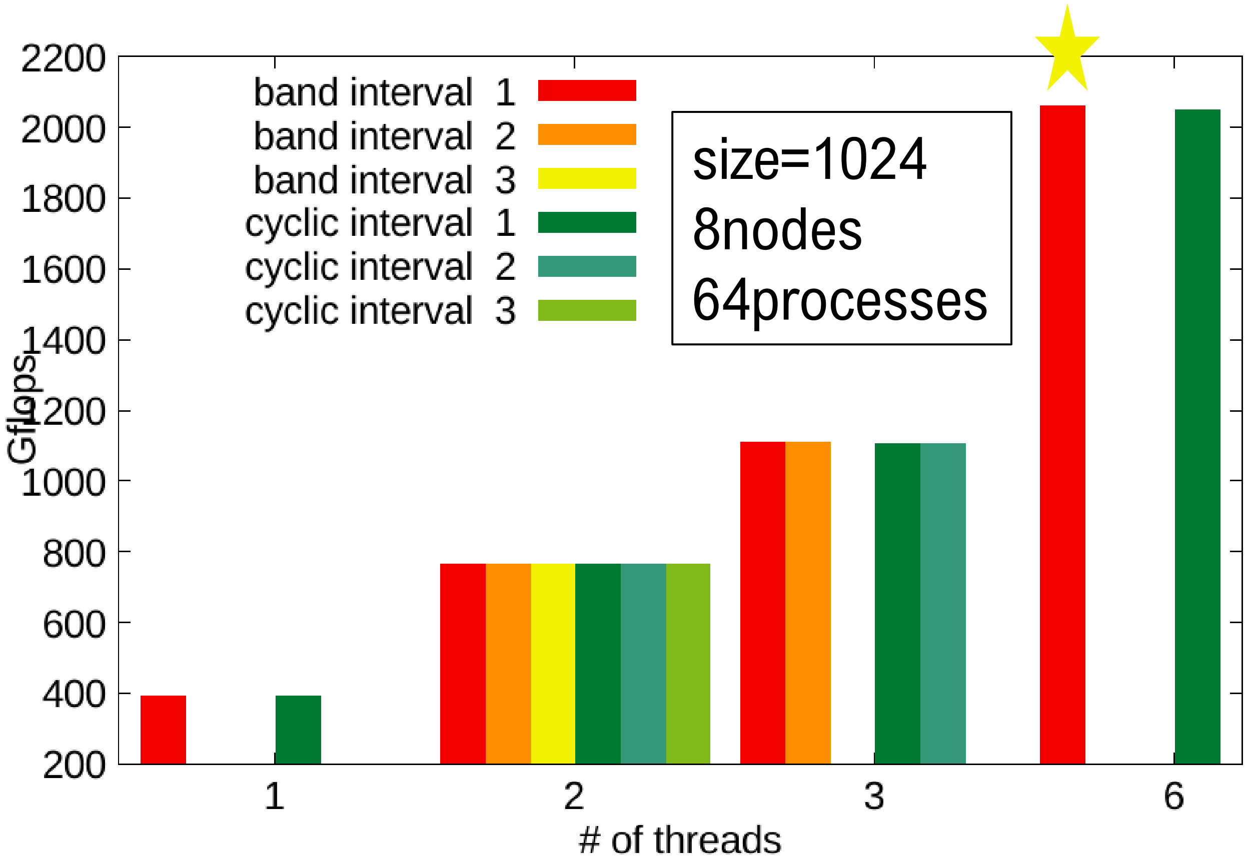
	256	1024
Lattice	256x256x256	1024x1024x1024
# of MPI processes	8	64
# of A64FX nodes	1 (48cores)	8 (384cores)

*Kobayashi et.al. Swopp2014

FFVC 256/1024

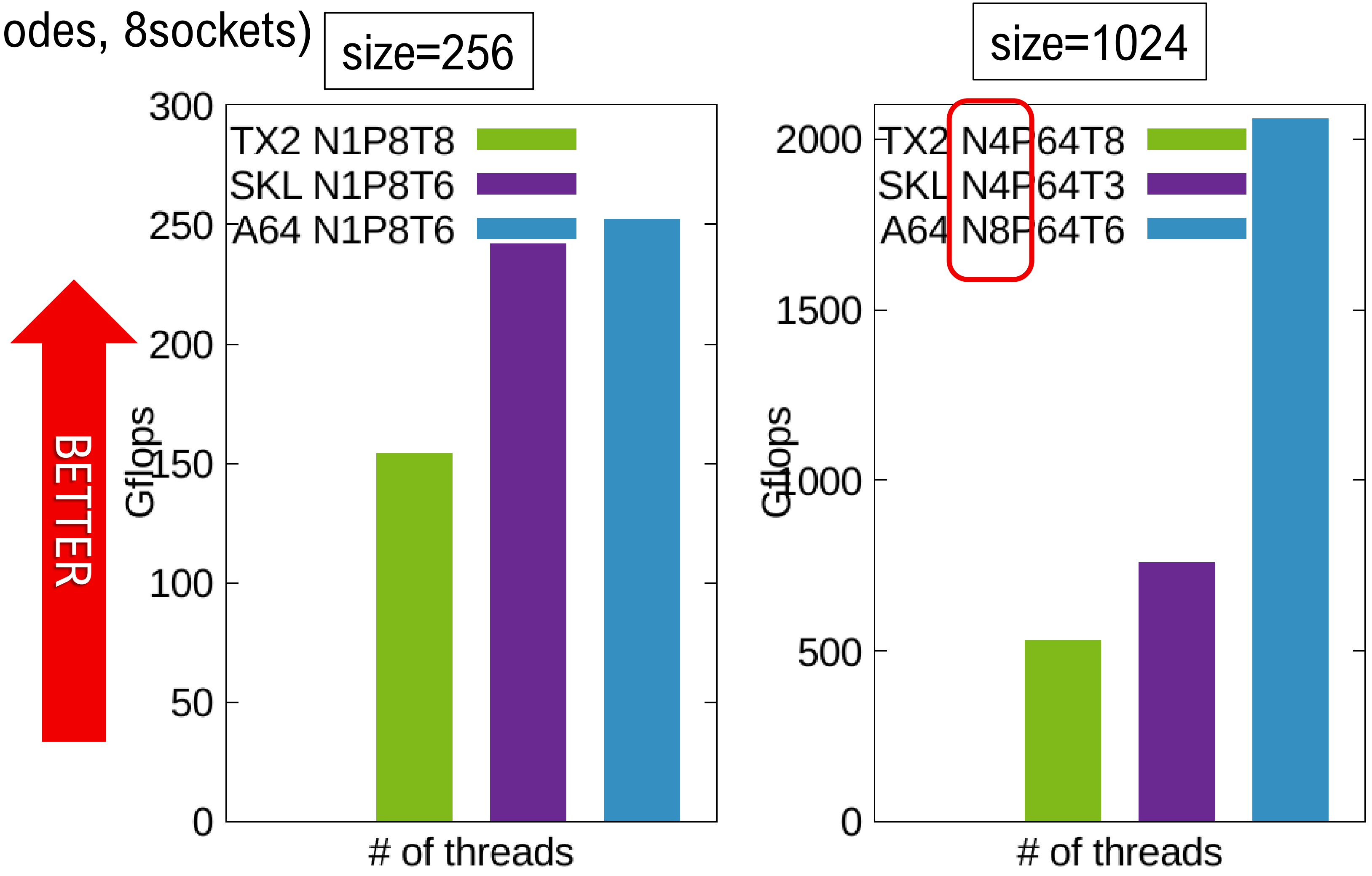
- Proceess assignment policy does not affect performance
  - share_band for MPI is slightly better than share_cyclic
- thread intervals do not affect the performance
- Good scalability up to 6 cores, i.e. the case all cores are used





FFVC Comparison

- A64FX shows better performance than TX2/SKL
- For1024 problem, A64FX (8nodes, 8sockets) is 2.7 times better than SKL (4nodes, 8sockets)



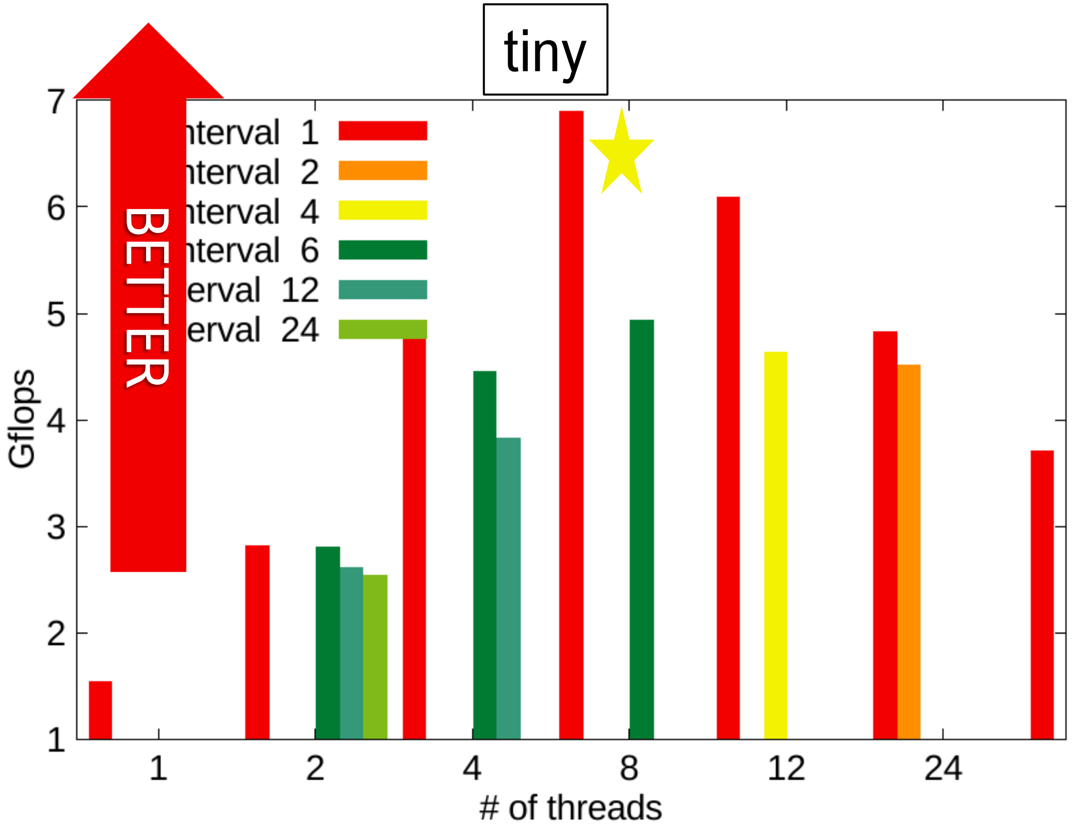
mVMC

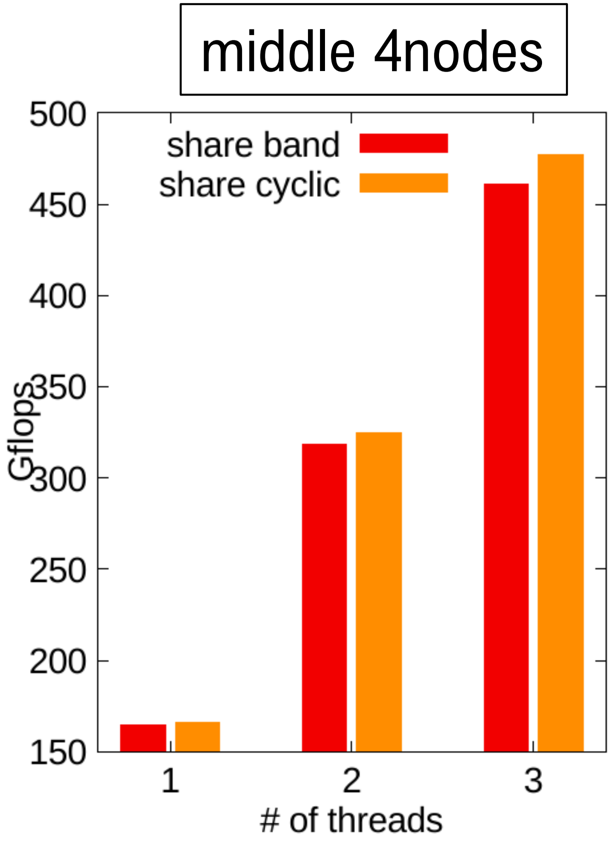
- mVMC analyzes the physical characteristics of the strongly correlated electron systems, by configuring the variational wave functions closely representing the ground state of such electron systems. It executes the Monte Carlo sampling of the real space configuration of the electrons.

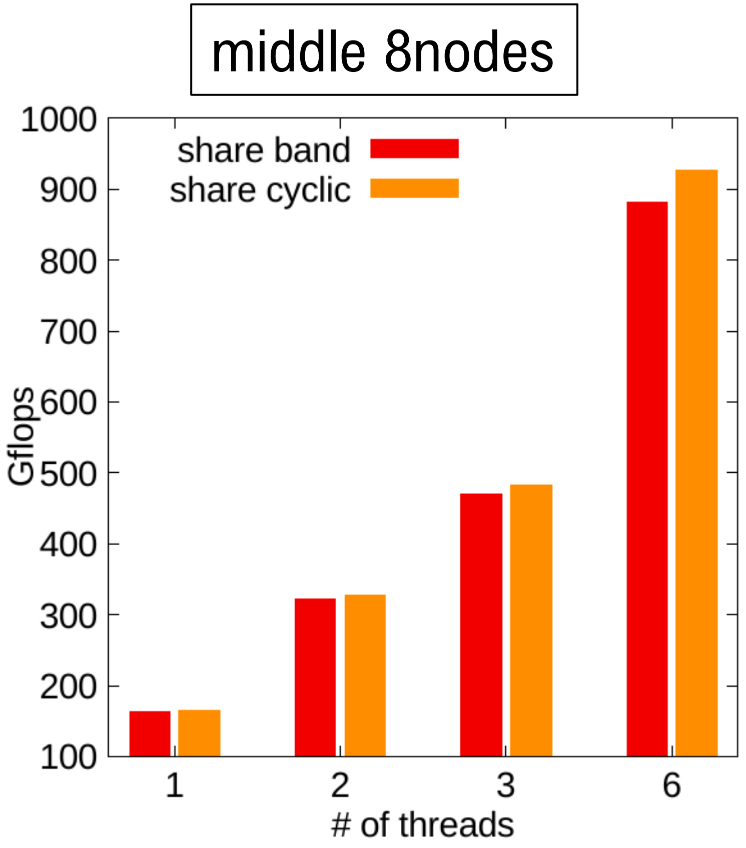
	tiny	middle
NSplitsize	1	4
# of MPI processes	1	64
A64FX nodes	1 (48cores)	4,(384cores)

mVMC

- For tiny, performance is saturated up to 8 threads
- Performance is sensitive to the thread intervals, “close” allocation is the best
- For MPI, share_cyclic is better
    - Threads should be closer, processes should be cyclic





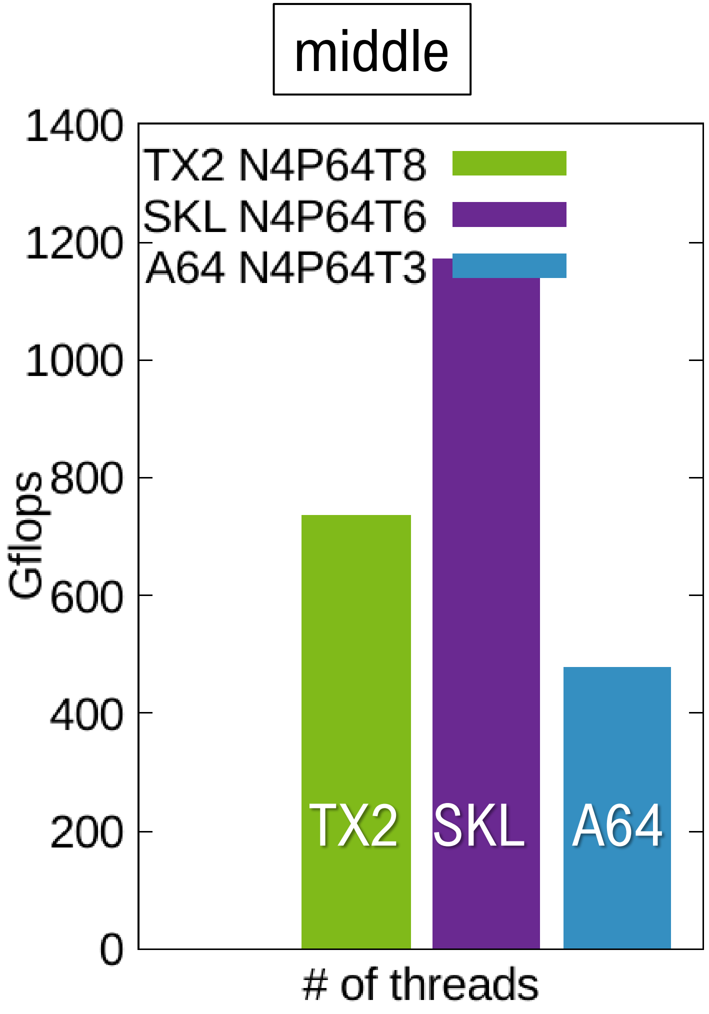


mVMC Comparison

- A64FX is the worse
  - Store Fetch Interlock (SFI)
    - SIMD Load instruction must wait the preceding store instruction of a same address
      even if it is masked and there is no real access
  - No Hyperthreading

- There is no "* **restrict**" keyword to ensure that there is no other pointer pointing to the same object in the original source code
- Adding the "* restrict" keyword improve the performance drastically

(sec)	exec(sec)	% of SIMD	% of SVE of SIMD	SFI rate
tiny as-is	3.686	21.47	92.89	0.15
tiny tuned	2.983	27.47	98.05	0.04
middle as-is	661.8	Unavailable (may be bugs in profiler …)		
middle tuned	**225.6**			



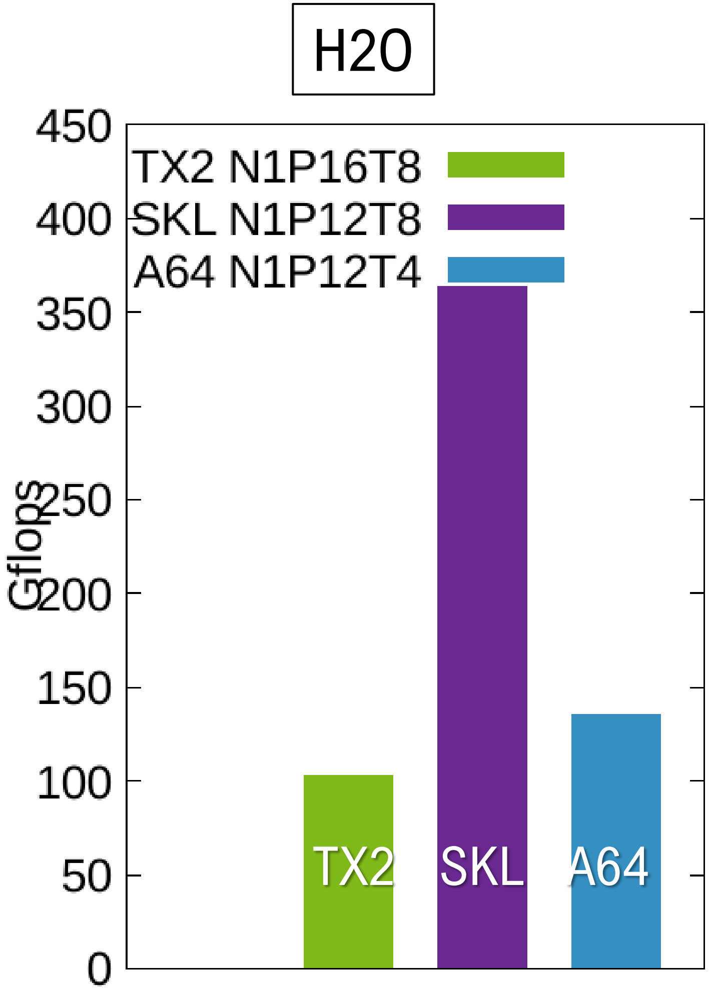
NTChem

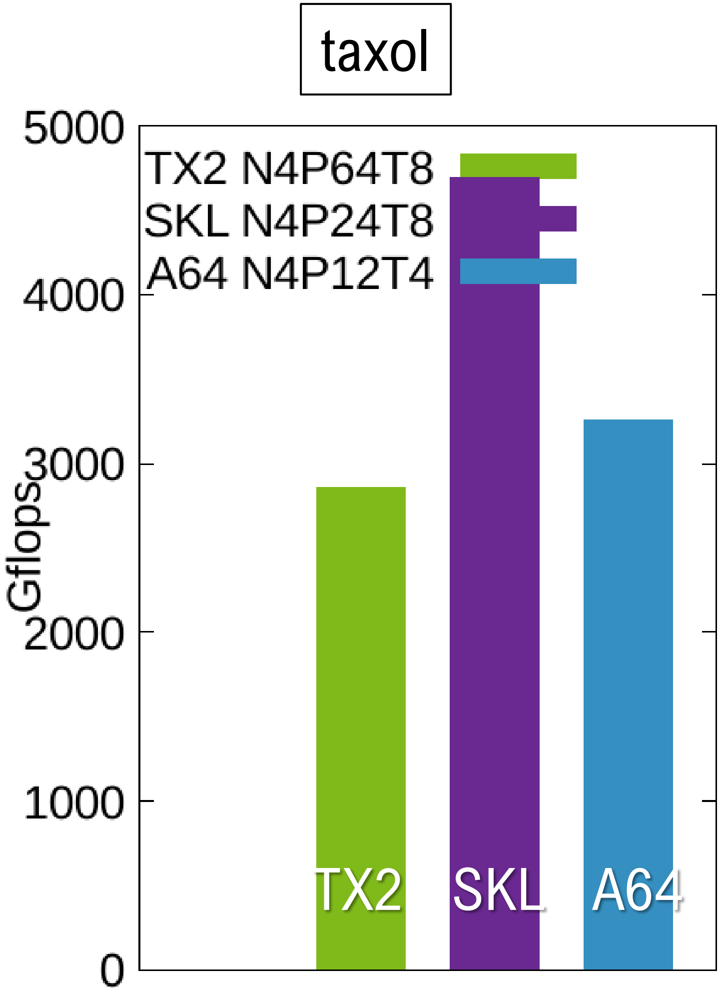
- First principle electronic structure calculation
- Dense matrix operations (**dgemm**)
  - BLAS/LAPACK are required

	h2o	taxol
# of nodes	1 (48cores)	4 (192cores)
A64FX # of MPI processes	12	48
A64FX OpenMP threads	4	4
SKL BLAS Library	SSL2BLAMP	SSL2BLAMP

NTChem Comparison

- SKL is the best for both
- A64FX is comparable with SKL for taxol considering the difference of peak performance (2.0GHz vs 2.7 GHx)
- The performance of A64FX for H2O is poor

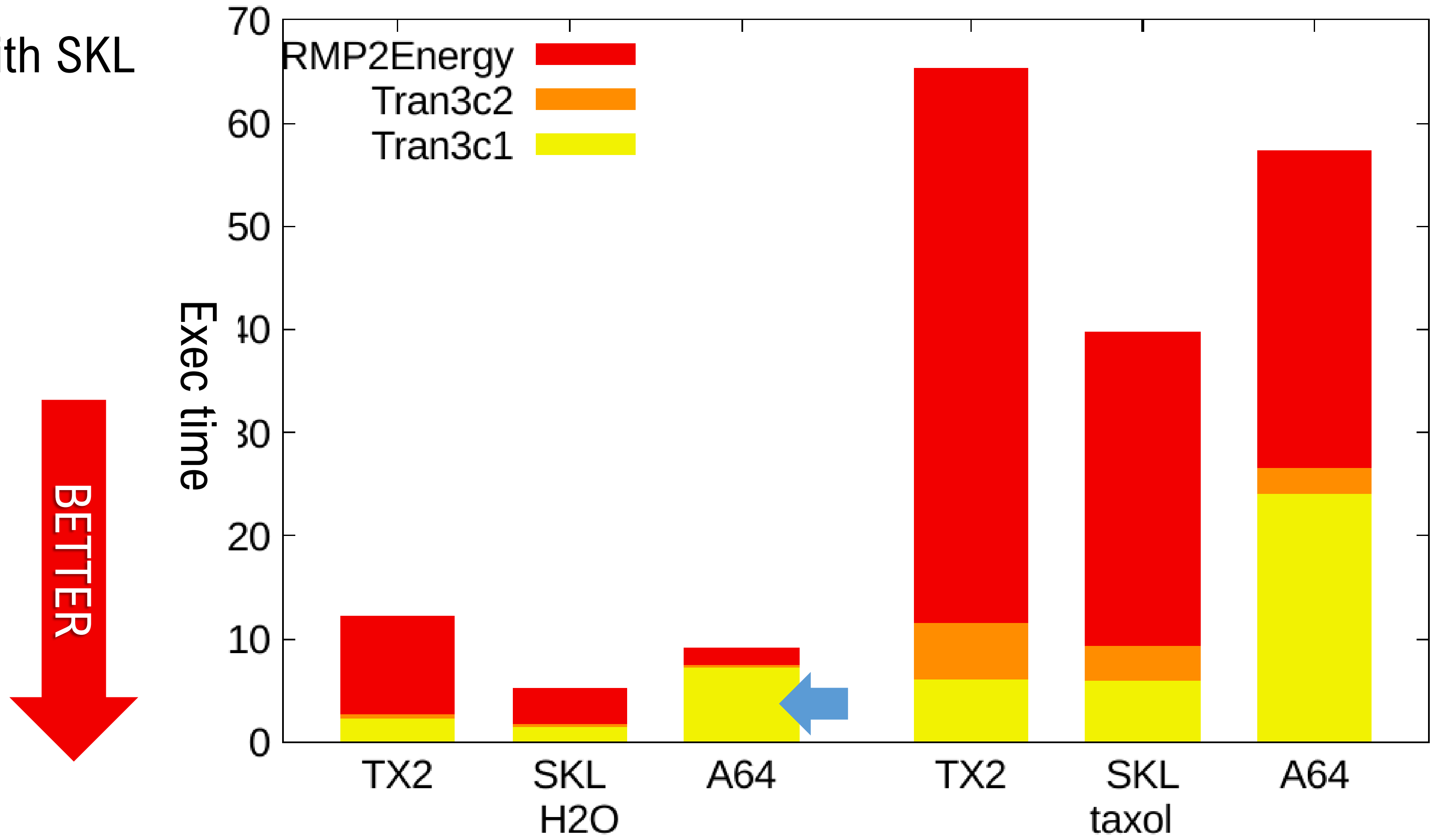




NTChem breakdown

- NTChem consists of three kernels -- RMP2Energy, Tran3c2, Tran3c1
- Large Tran3c1 ratio in A64FX, especially in the H2O problem

- A64FX is comparable with SKL or better than SKO for RMP2Energy kernel



NTChem: Tuning of the tran3c1

```
DO IJ = 1, NIJCar
   KL = 0
   DO K = 1, NKSph
      DO L = 1, NLSph
         KL = KL + 1
         Dum = Zero
         KLC = 0
         DO KC = 1, NKCar
            DO LC = 1, NLCar
              KLC = KLC + 1
              Dum = Dum + SphCoef(KC,K,IAnglC) *
              SphCoef(LC,L,IAnglD) * ERI_Array(NIJC+KLC)
            END DO
         END DO
         ERI_Array_Temp(NIJ+KL) = Dum
      END DO
   END DO
   NIJ  = NIJ  + NKLSph
   NIJC = NIJC + NKLCar
END DO
```

This loop is SIMDized and software pipelined.
※ Only the innermost loop will be SIMDized

However, since **NLCar is almost always 1,** there is no actual SIMD instruction and software pipeline effect in the code shown here

NTChem: Tuning of the tran3c1

(sec)	Total	Tran3c1
H2O as-is	11.15	8.25
H2O tuned	10.20	**7.03**
taxol as-is	57.28	24.09
taxol tuned	47.81	**16.17**

```
IF(NLCar==1) then
DO IJ = 1, NIJCar
   KL = 0
   DO K = 1, NKSph
      DO L = 1, NLSph
         KL = KL + 1
         Dum = Zero
         KLC = 0
         DO KC = 1, NKCar
            DO LC = 1, NLCar
              KLC = KLC + 1
              Dum = Dum + SphCoef(KC,K,IAnglC) *
              SphCoef(1 ,L,IAnglD) * ERI_Array(NIJC+KLC)
            END DO
         END DO
         ERI_Array_Temp(NIJ+KL) = Dum
      END DO
   END DO
   NIJ  = NIJ  + NKLSph
   NIJC = NIJC + NKLCar
END DO
```

SIMD and SWP

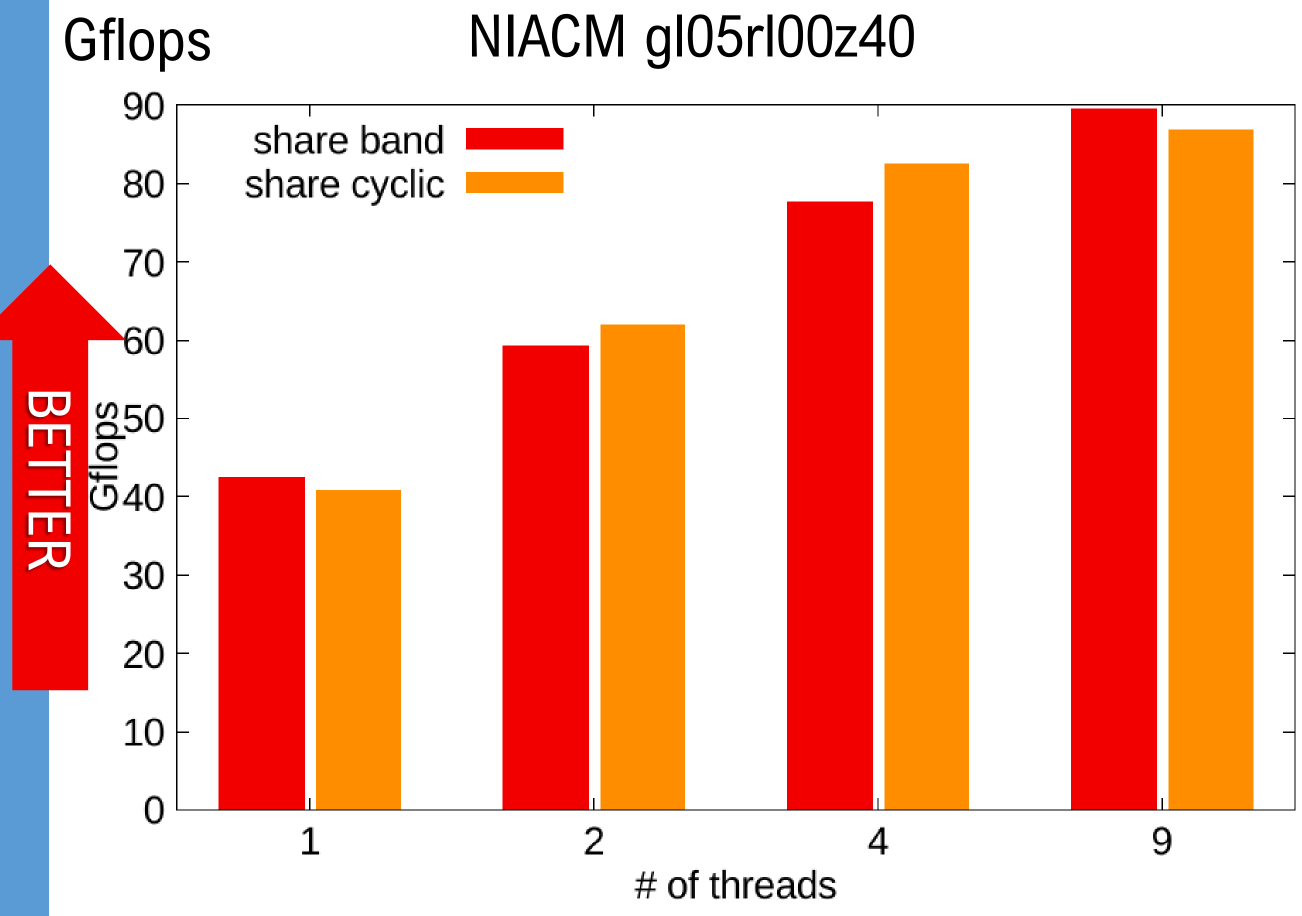
NICAM-DC

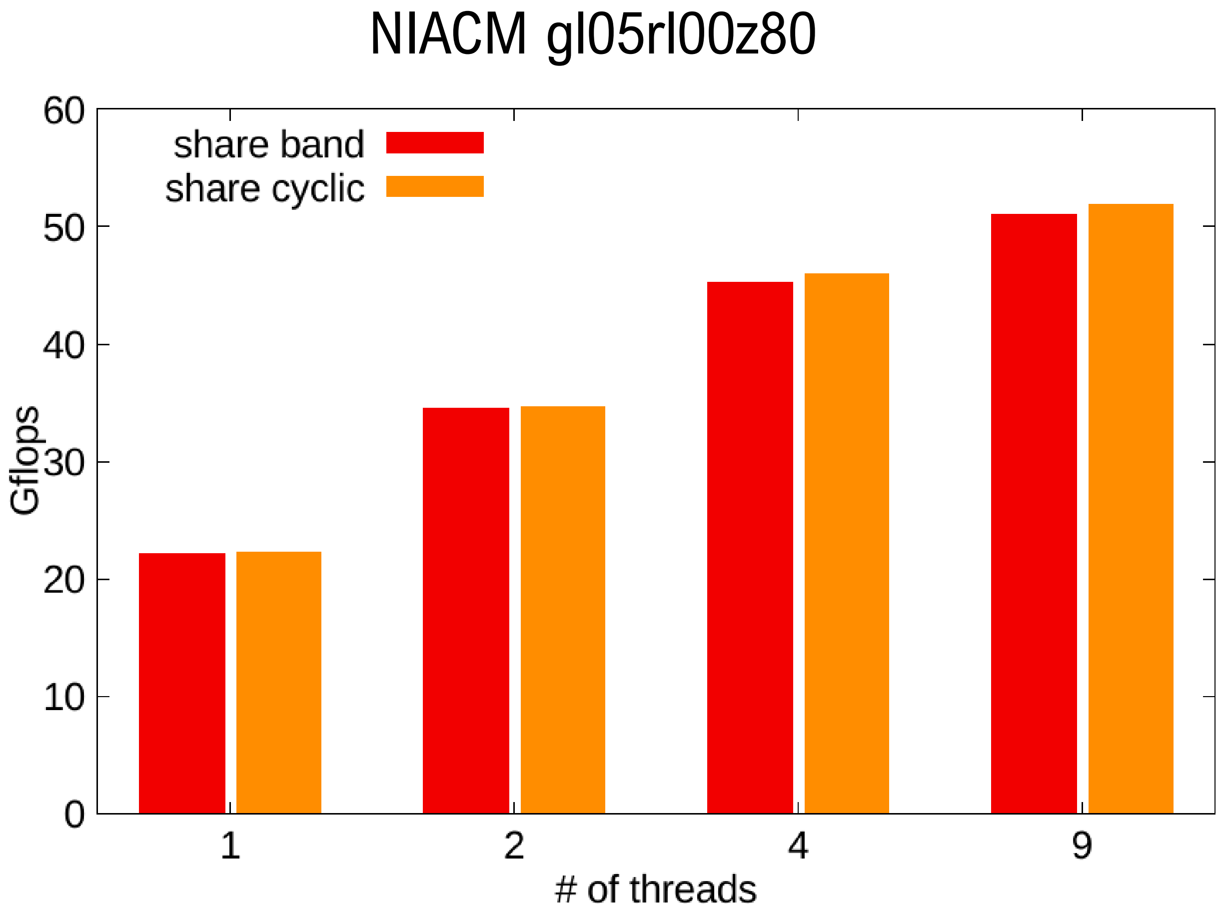
- Non-hydrostatic ICosahedral Atmospheric Model (NICAM)
- Atmospheric general circulation model reproducing the unsteady baroclinic oscillation
- No OpenMP directive
  - **-Kparallel** option, which parallelizes loops for threads automatically, has been used
  - The auto parallelizations in SKL, TX2 had not worked well…

	NIACM gl05rl00z40	NIACM gl05rl00z80
# of MPI processes	10	5
A64FX Nodes	2 (96)	1 (48 cores)
A64FX OpenMP threadss	1, 2, 4, 9	1, 2, 4, 9

NICAM-DC

- Good scaling up to 4 threads

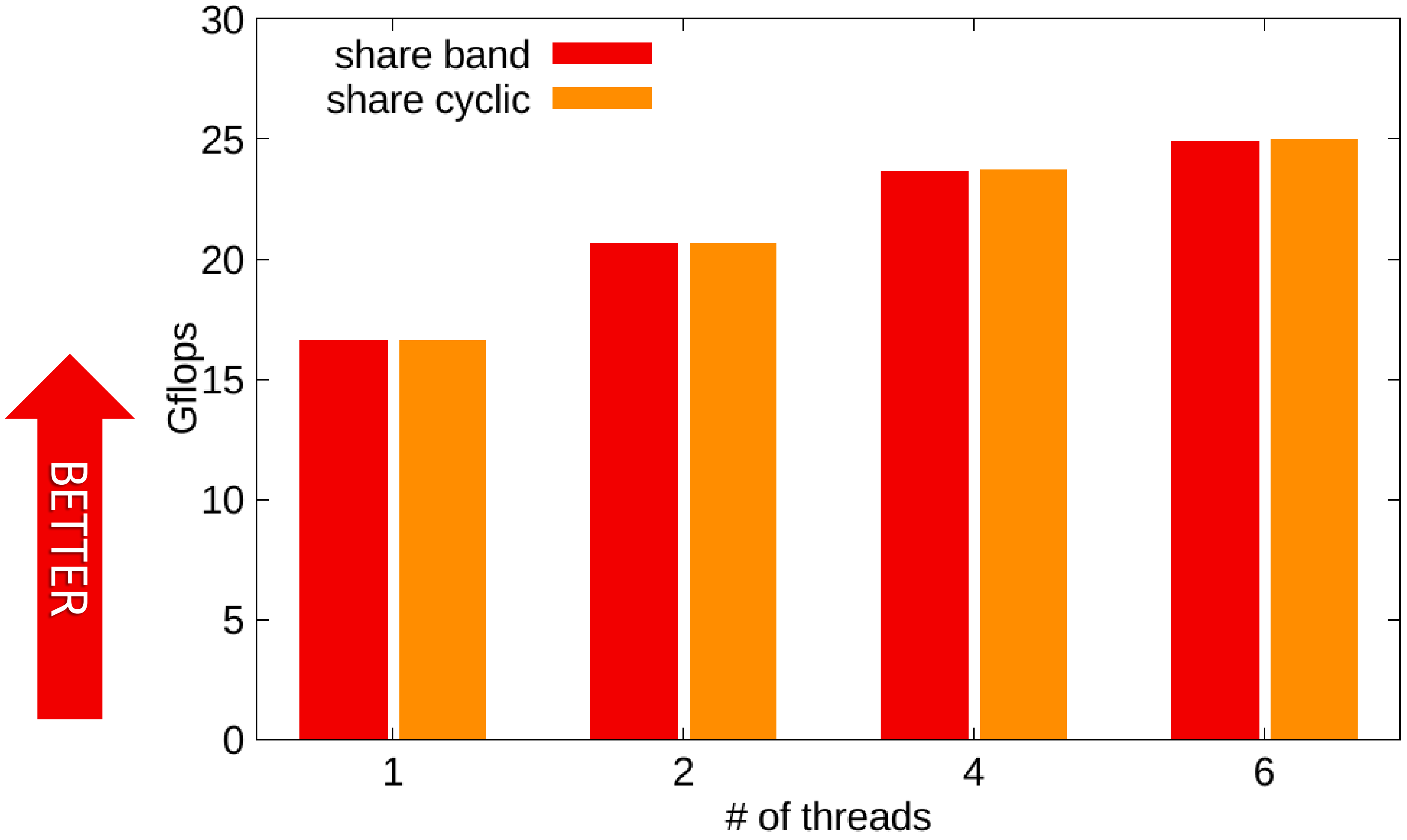




FFB

- FrontFlow/blue (FFB)
  - a general purpose thermal flow simulation program
  - solves the unsteady incompressible Navier-Stokes equations by finite element method
  - Dataset sample
  - No OpenMP directive

    -Kparallel
    - 1node
    - 8MPI
    - 1,2,4,6 OpenMP



Summary

- The performance evaluation of A64FX using Fiber mini-apps suite
- Process allocation strategy does not affect the performance in most cases
- Performance analysis
  - poor performance comes from poor % of SVE instructions
  - wait time of floating point operations
- SIMD and software pipelining are important to improve the performance
  - re-roll the hand unrolled loop
  - avoid (may be) short loop whose number of iteration is specified by a variable
  - investigate compiler options regarding instruction scheduling