Charm++ workshop, Nov. 18, 2021

# Challenges of Programming models for The Supercomputer “Fugaku” and Beyond


## Mitsuhisa Sato

**Team Leader of Programming Environment Research Team**

**Deputy Director, RIKEN Center for Computational Science (R-CCS)**

**Professor (Cooperative Graduate School Program), University of Tsukuba**



**Thanks to Tetsuya Odajima and Yuetsu Kodama, … and many project members**

**FLAGSHIP 2020 project, R-CCS**

# Outline of my talk

- **Co-design of A64FX processor for “Fugaku” in FLAGSHIP 2020 project**
  - Design target and KPIs, and co-design
  - Overview of A64FX processor
    - A64FX was developed by Fujitsu and RIKEN, and the first processor equipped with Arm SVE.
- **The Performance results of A64FX processor**
  - UK benchmark and LULESH, Open-source HPC software, SPEC® benchmark
  - A64FX performance characteristics & performance tuning, Power consumption
- **System software overview of “Fugaku”**
- **Challenges of programming models for Fugaku and beyond**
- **Concluding remarks**

# KPIs on Fugaku development in FLAGSHIP 2020 project

## 3 KPIs (key performance indicator) were defined as the design target for Fugaku development

- **1. Extreme Power-Efficient System**
  - Maximum performance under Power consumption of 30 - 40MW (for system)
- **2. Effective performance of target applications**
  - It is expected to exceed 100 times higher than the K computer's performance in some applications

# Target Application's Performance

- **Performance Targets**
  - 100 times faster than K for some applications (tuning included)
  - 30 to 40 MW power consumption

https://postk-web.r-ccs.riken.jp/perf.html

## Predicted Performance of 9 Target Applications

*As of 2019/05/14*

| Area | Priority Issue | Performance Speedup over K | Application | Brief description |
|---|---|---|---|---|
| Health and longevity | 1. Innovative computing infrastructure for drug discovery | **x125+** | **GENESIS** | MD for proteins |
| | 2. Personalized and preventive medicine using big data | **x8+** | **Genomon** | Genome processing (Genome alignment) |
| Disaster prevention and Environment | 3. Integrated simulation systems induced by earthquake and tsunami | **x45+** | **GAMERA** | Earthquake simulator (FEM in unstructured & structured grid) |
| | 4. Meteorological and global environmental prediction using big data | **x120+** | **NICAM+ LETKF** | Weather prediction system using Big data (structured grid stencil & ensemble Kalman filter) |
| Energy issue | 5. New technologies for energy creation, conversion / storage, and use | **x40+** | **NTChem** | Molecular electronic (structure calculation) |
| | 6. Accelerated development of innovative clean energy systems | **x35+** | **Adventure** | Computational Mechanics System for Large Scale Analysis and Design (unstructured grid) |
| Industrial competitiveness enhancement | 7. Creation of new functional devices and high-performance materials | **x30+** | **RSDFT** | Ab-initio program (density functional theory) |
| | 8. Development of innovative design and production processes | **x25+** | **FFB** | Large Eddy Simulation (unstructured grid) |
| Basic science | 9. Elucidation of the fundamental laws and evolution of the universe | **x25+** | **LQCD** | Lattice QCD simulation (structured grid Monte Carlo) |

# KPIs on Fugaku development in FLAGSHIP 2020 project

## 3 KPIs (key performance indicator) were defined as the design target for Fugaku development

- **1. Extreme Power-Efficient System**
  - Maximum performance under Power consumption of 30 - 40MW (for system)
- **2. Effective performance of target applications**
  - It is expected to exceed 100 times higher than the K computer's performance in some applications
- **3. Ease-of-use system for wide-range of users**

# Codesign of “Fugaku”

## 3 Design Targets:

- **1. Extreme Power-Efficient System**
  - Maximum performance under Power consumption of 30 - 40MW (for system)
- **2. Effective performance of target applications**
  - It is expected to exceed 100 times higher than the K computer’s performance in some applications
- **3. Ease-of-use system for wide-range of users**

Cool (Low-power) technology is important!!

Codesign

Codesign to meet these 3 design targets

### Technologies and Architectural Parameters to be determined

- **Basic Architecture Design (by Feasibility Studies)**
  - Manycore approach, O3 cores, some parameters on chip configuration and SIMD
- **Instruction Set Architecture and SIMD Instructions**
  - Fujitsu collaborated with Arm, contributing to the design of the SVE as a lead partner
- **Chip configuration**
- **Memory technology**
  - DDR, HBM, HMC …
- **Cache structure**
- **Out of order (O3) resources**
- **Enhancement for Target Applications**
- **Interconnect between Nodes**
  - SerDes, topologies “Tofu” or other network?

Charm++ workshop

- ✓ The number of cores in a CMG
- ✓ The number of CMGs in a chip
- ✓ How to connect cores to shared L2 in a CMG
- ✓ The number of ways, the size, and throughputs of the L1
- ✓ and L2 caches
- ✓ The topology of network-on-chip to connect CMGs
- ✓ The die size of the chip
- ✓ The number of chips in a node

# Supercomputer “Fugaku” and A64FX processor

- **Ultra-scale “general-purpose” manycore system: 158,976 nodes (1 processor/node, total 7.6 M cores, theoretical peek 537PFLOPS (DP))**
- **Arm-based manycore processor: Fujitsu A64FX (Armv8.2-A SVE 512bit SIMD, #core 48 + 2/4, 3TF@2.0GHz, boost to 2.2GHz)**
  - 12 cores in a cluster of cores called CMG, connected to L2 and HBM memory chips
- **Advanced Memory technology: HBM2 32 GiB, 1024 GB/s bandwidth, packaged in CPU chip**
- **Scalable Interconnect: ToFu-D interconnect**

- Standard programing model is OpenMP-MPI hybrid programming. running each MPI process on a NUMA node (CMG).
- 48 threads OpenMP is also supported.

CMG(Core-Memory-Group): NUMA node
12+1 core

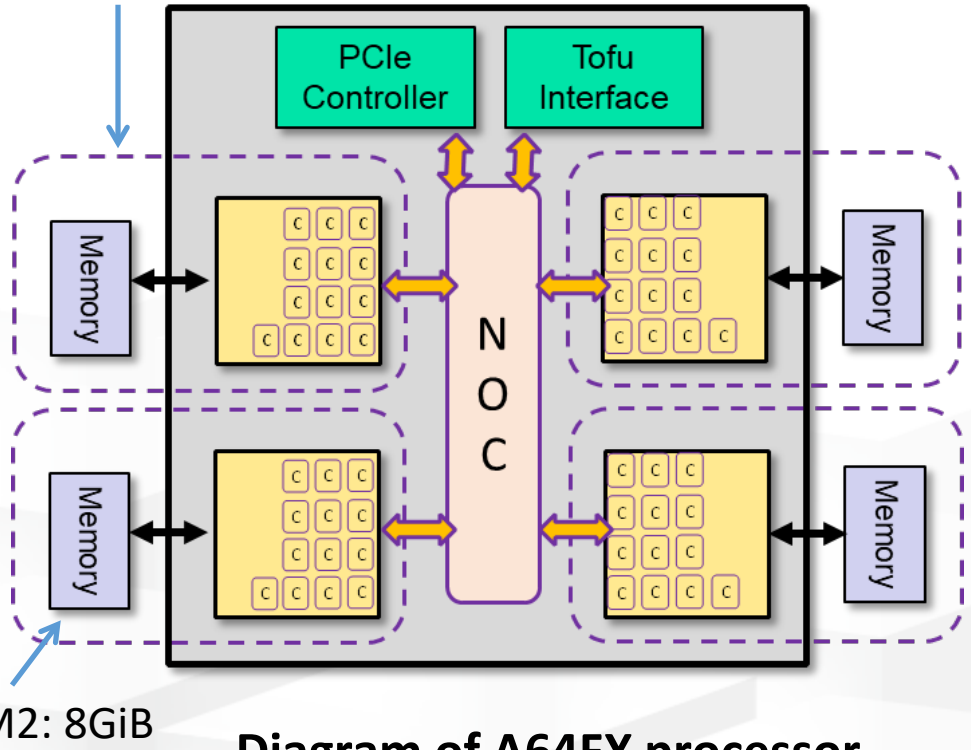
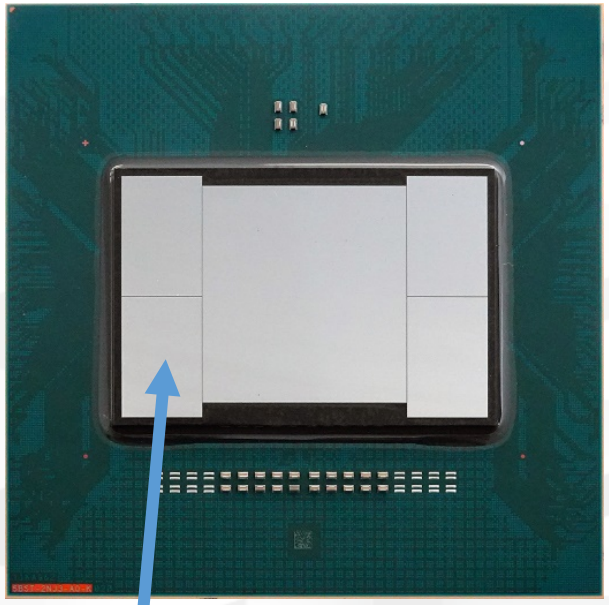
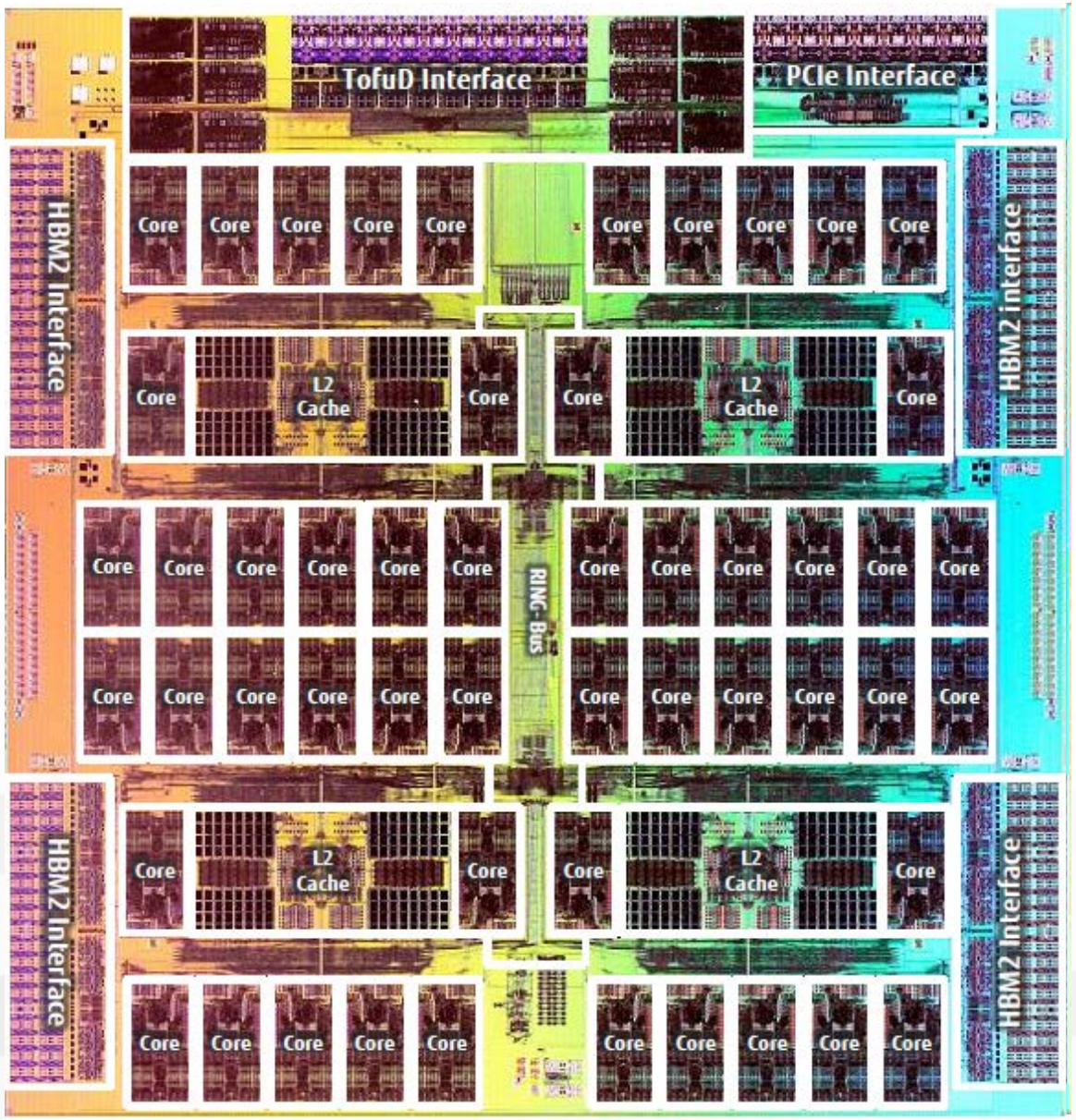



**Diagram of A64FX processor**

# Die Photograph of A64FX processor

- TSMC 7nm FinFET
- 400 mm^2
- HBM2 chips are mounted on Si-interposer connected by TSMC CoWoS technology

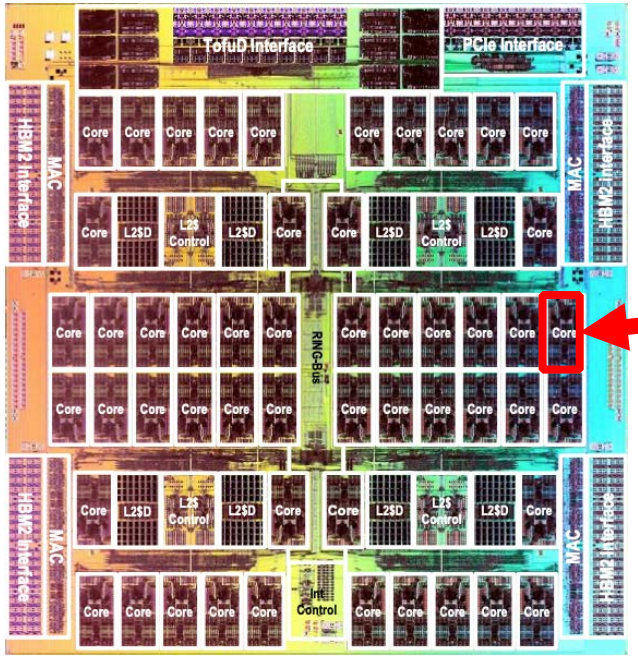


# Comparison of Die-size

- **A64FX: 52 cores (48 cores), 400 mm² die size (8.3 mm²/core), 7nm FinFET process (TSMC)**
- **Xeon Skylake: 20 tiles (5x4), 18 cores, ~485 mm² die size (estimated) (26.9 mm²/core), 14 nm process (Intel)**
- **A64FX core is more than 3 times smaller per core.**

A64FX:
400 mm²
(20 x 20)




https://www.fujitsu.com/jp/solutions/business-technology/tc/catalog/ff2019-post-k-computer-development.pdf

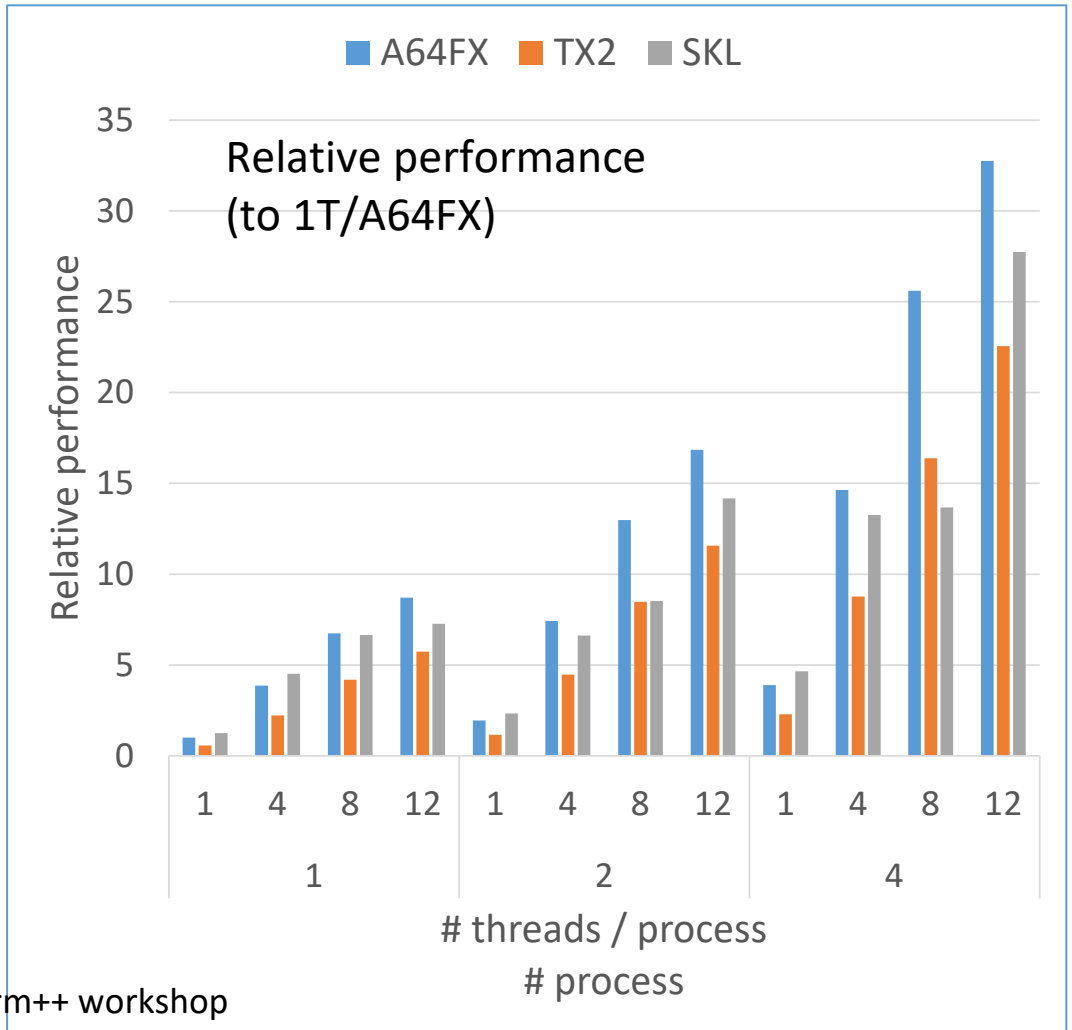
https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(server)

Xeon Skylake, High Core Count:
4 x 5 tiles, 18 cores, 2 tiles used for memory interface
485 mm² (22 x 22)

# Benchmark result of CloverLeaf

Taken form UK benchmarks:
A hydrodynamics mini-app to solve the compressible Euler equations in 2D, using an explicit, second-order method

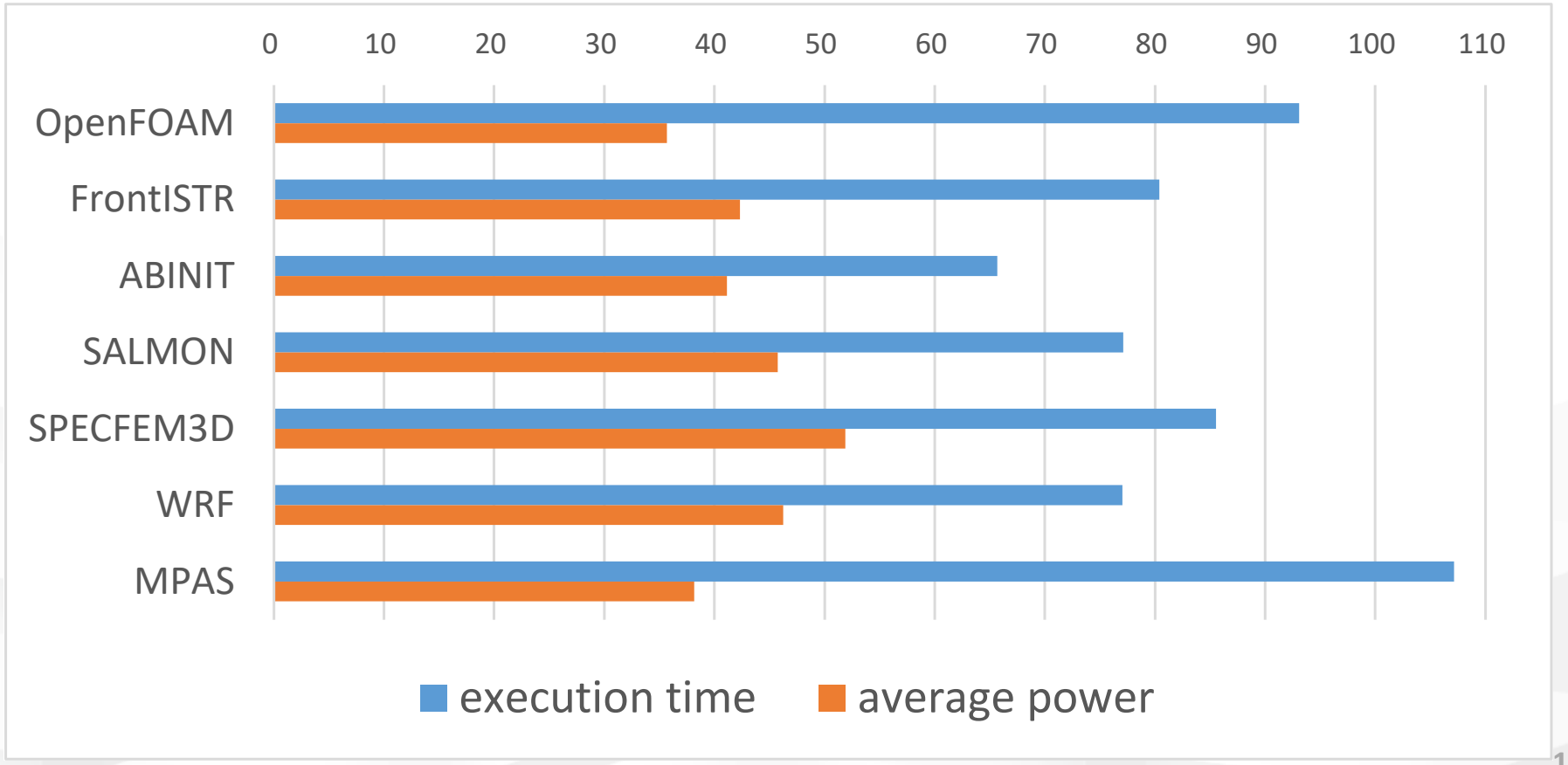
- Comparison with two nodes of TX2 (dual) and Skylake (dual)
- Good scalability by increasing the number of threads within CMG.
- The performance of one A64FX is comparable (better) to that of two nodes (4 sockets) of Skylake

# Performance and Power-efficiency of HPC OSS

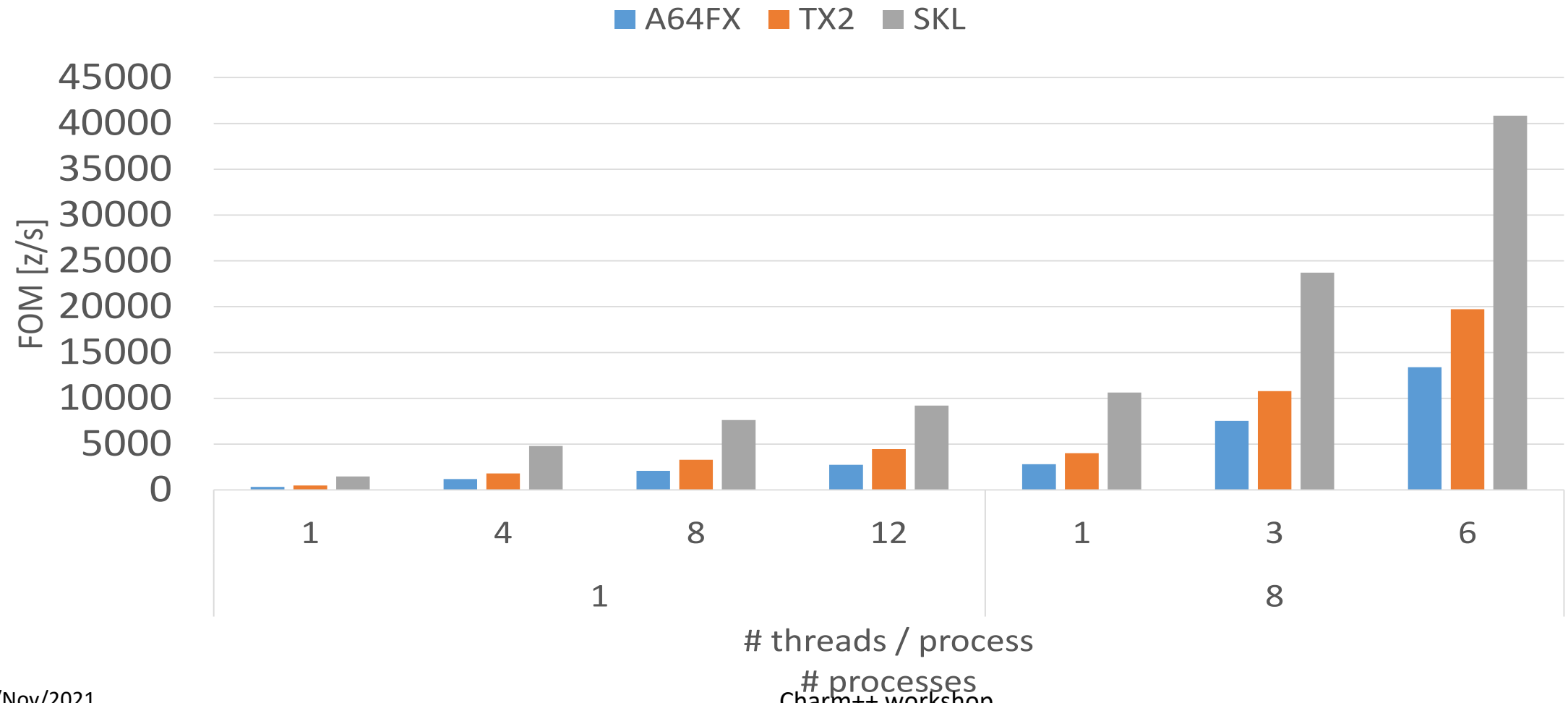
- **Several Open-source software were already ported and evaluated.**
- **Evaluation using one chip A64FX and dual chips of Xeon.**
- **The almost same performance to dual sockets of Xeon with half of power consumption.**

Performance and power efficiency of open-source applications
(results are shown in %, relative to Intel Xeon Platinum 8268 (Cascadelake, 2.90 GHz, 24 cores/socket) (dual sockets))



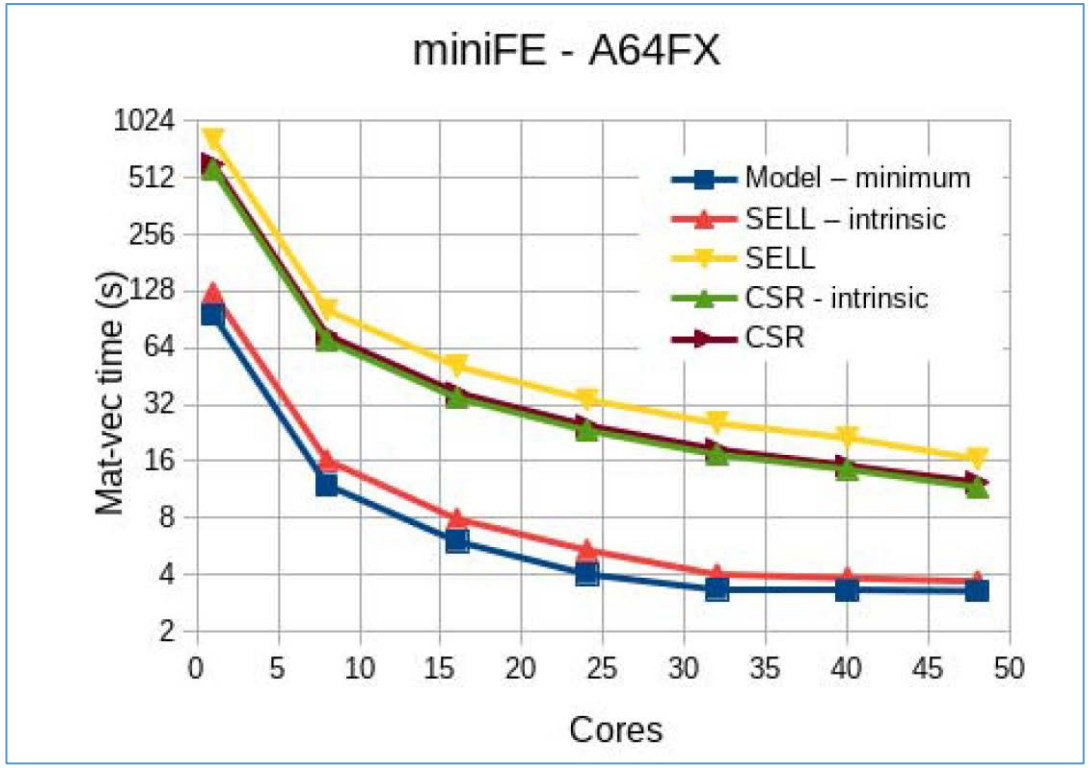
# LULESH

- A64FX performance is less than Thx2 and Intel one
- We found low vectorization (SIMD (SVE) instructions ratio is a few %)
- We need more code tuning for more vectorization using SIMD



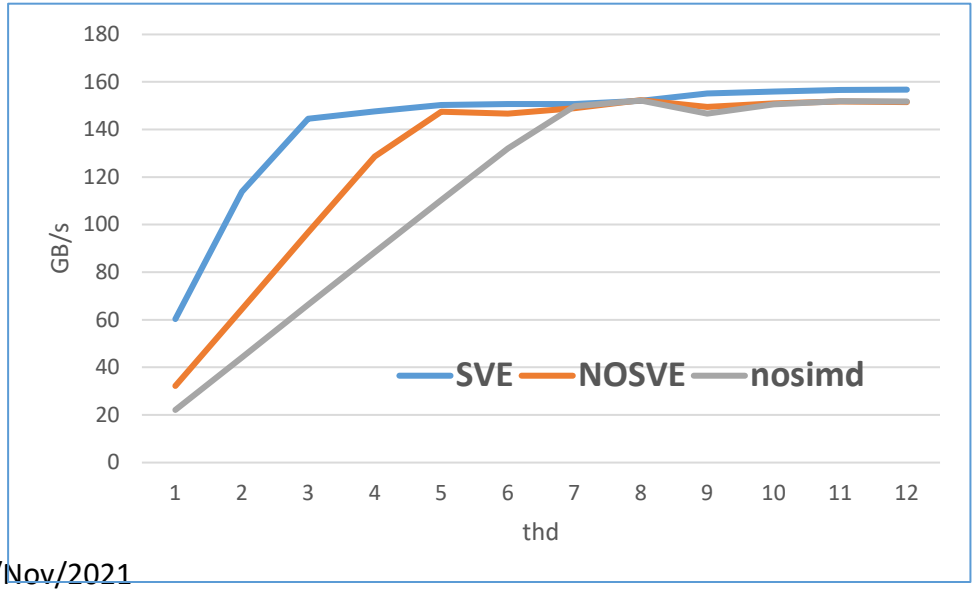
# How to improve the performance of sparse-matrix code

- **Storage format is important:**
  - Sliced ELLPACK format shows significantly better performance than CSR, but only when it is vectorized manually using intrinsics.”
  - CSR is not good even with manual vectorizing.
- **Vectorizing with SVE is important to get memory bandwidth.**




B. Brank, S. Nassyr, F. Pouyan and D. Pleiter, "Porting Applications to Arm-based Processors," EAHPC Workshop, *IEEE CLUSTER* 2020 , Kobe, Japan, 2020, pp. 559-566, doi: 10.1109/CLUSTER49012.2020.00079.

**Memory bandwidth with hardware prefetch**



# SPEC CPU® 2017 integer Speed

- **The performance of A64fX is about ¼ performance of Xeon in single thread.**
  - Fugaku uses normal mode (2.0GHz) with Fujitsu compiler tcsds-1.2.30a. For c and c++, clang mode is used.
  - Xeon is Cisco UCS B200 M5 (Platinum 8168(Skylake), 2.7GHz, 24core x 2 chip, turbo on) with icc 18.0.2.

https://www.spec.org/cpu2017/results/res2018q2/cpu2017-20180529-06367.txt

  - Reference machine is UltraSPARC-IV+(2.1GHz, 2cores x 4 chip)

- **The reason for the low single thread integer performance of A64FX is that**
  - the SIMD rate is low in SPEC CPU/int and
  - the frequency and the O3 resource are limited for the throughput-oriented architecture of A64FX.

| | Lang | Threads | A64FX | Xeon |
|---|---|---|---|---|
| 600.perlbench_s | C | 1 | 1.20 | 6.20 |
| 602.gcc_s | C | 1 | 2.63 | 9.57 |
| 605.mcf_s | C | 1 | 3.42 | 11.2 |
| 620.omnetpp_s | C++ | 1 | 1.26 | 7.31 |
| 623.xalancbmk_s | C++ | 1 | 1.61 | 9.46 |
| 625.x264_s | C | 1 | 2.06 | 11.6 |
| 631.deepsjeng_s | C++ | 1 | 1.37 | 5.17 |
| 641.leela_s | C++ | 1 | 1.26 | 4.36 |
| 648.exchange2_s | F90 | 1 | 1.42 | 13.2 |
| 657.xz_s | C/OpenMP | 48 | 8.52 | 23.5 |
| SPECspeed®2017_int_base | | | 1.98 | 9.07 |

```
COPTIMIZE       = -Nclang -Ofast -mcpu=a64fx+sve -ffj-no-fp-relaxed -ffj-eval-concurrent -fsave-optimization-record -fopenmp -Nlst=t -Koptmsg=2
CXXOPTIMIZE     = -Nclang -Ofast -mcpu=a64fx+sve -ffj-no-fp-relaxed -ffj-eval-concurrent -fsave-optimization-record -fopenmp -Nlst=t -Koptmsg=2
FOPTIMIZE       = -Kfast,openmp -Nlst=t -Koptmsg=2
```

# SPEC OMP® 2012

- **The performance of A64FX using 48 thread is about 65% performance of Xeon using 56 thread (28 cores).**
  - Fugaku uses normal mode (2.0GHz) with Fujitsu compiler tcsds-1.2.30a. For c and c++, clang mode is used.
  - Xeon is Cisco C240 M5 (Platinum 8280(Cascade Lake), 2.7GHz, 28core x 1chip, hyperthread on (56threads), turbo on) with icc 19.0.1.

https://www.spec.org/omp2012/results/res2019q2/omp2012-20190313-00172.txt

  - Reference machine is Sun Fire X4140 (AMD Opteron 2384, 2.7GHz 4core x 2chips)

- **For some programs (swim and mgrid), A64FX brings extremely good performance due to HBM2.**
- **For 350.md, performance improvement has been confirmed by source code tuning, and we hope that it will be applied by improving the compiler.**

| | Lang | Threads | A64FX | Xeon |
|---|---|---|---|---|
| 350.md | F | 48 | 2.63 | 62.6 |
| 351.bwaves | F | 48 | 15.5 | 11.2 |
| 352.nab | C | 48 | 3.00 | 12.9 |
| 357.bt331 | F | 48 | 5.82 | 16.0 |
| 358.botsalgn | C | 48 | 5.22 | 10.5 |
| 359.botsspar | C | 48 | 3.07 | 6.83 |
| 360.ilbdc | F | 48 | 7.69 | 8.25 |
| 362.fma3d | F | 48 | 4.28 | 11.3 |
| 363.swim | F | 48 | 53.1 | 8.38 |
| 367.imagick | C | 48 | 12.2 | 13.6 |
| 370.mgrid331 | F | 48 | 32.6 | 7.46 |
| 371.applu331 | F | 48 | 8.88 | 14.4 |
| 372.smithwa | C | 48 | 12.8 | 11.8 |
| 376.Kdtree | C++ | 48 | 3.22 | 9.24 |
| SPECompG_base2012 | | | 7.77 | 12.0 |

# Summary of A64FX performance characteristics

- **For core-to-core comparison in intspeed, integer performance is ¼ of Xeon**
- **For chip-to-chip comparison in SPEC OMP, 48 threads performance of one chip is 65% to one chip of recent high-end Xeon (Cascade Lake)**
  - NOTE: Performance of memory-intensive benchmarks is extremely good in A64FX thanks to HBM.
- **For some scientific workload, the almost same performance to dual sockets of Xeon with half of power consumption (UK benchmark and HPC OSS)**
- **High SIMD rate is important to get performance**
  - Need to tune memory access pattern
  - We found many benchmark programs are not well-vecterized.
- **Power efficiency of A64FX is very good (double efficiency than Xeon?)**

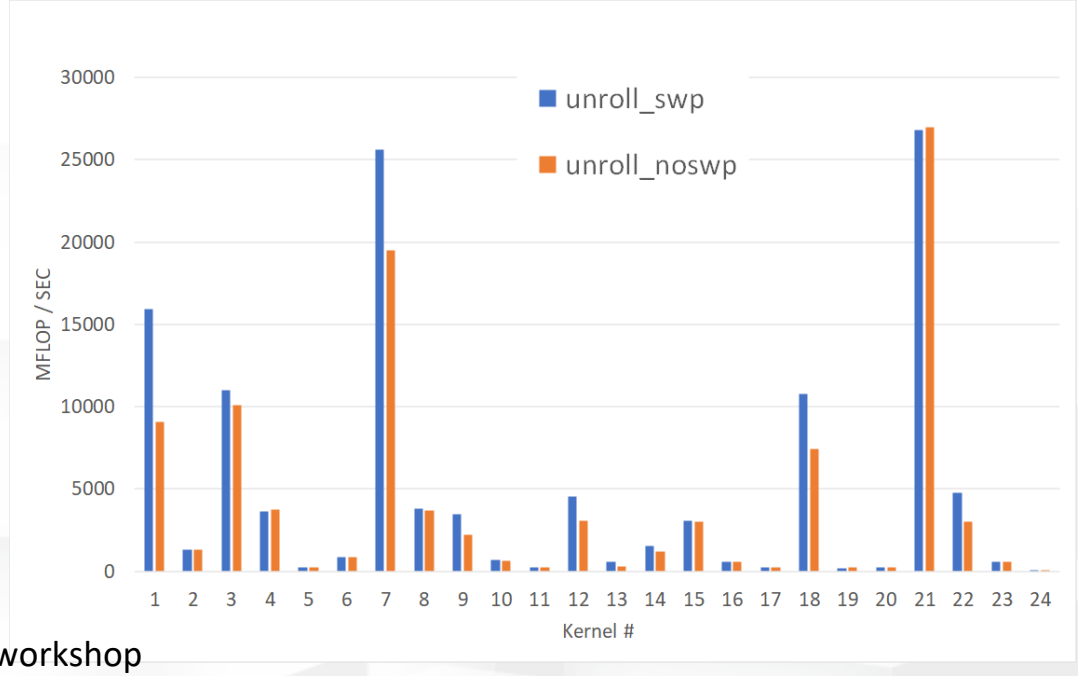
# Performance Tuning for A64FX processor

- **HPC-oriented design**
  - Small core ⇒ Less O3 resources
  - (Relatively) Long pipeline
    - 9 cycles for floating point operations
    - Core has only L1 cache
  - High-throughput, but long-latency
  - Pipeline often stalls
    for loops having complex body.

| | A64FX | Skylake |
|---|---|---|
| ReOrder Buffer | 128 entries | 224 entries |
| Reservation Station | 60 (=10x2+20x2) entries | 97 entries |
| Physical Vector Register | 128 (=32 + 96) entries | 168 entries |
| Load Buffer | 40 entries | 72 entries |
| Store Buffer | 24 entries | 56 entries |

A64FX : https://github.com/fujitsu/A64FX
Skylake : https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(server)

- **Compiler optimization (Fujitsu compiler)**
  - SWP: software pipelining
    - ～ 20% speedup in Livermore Kernels
  - Automatic and Manual loop fissions



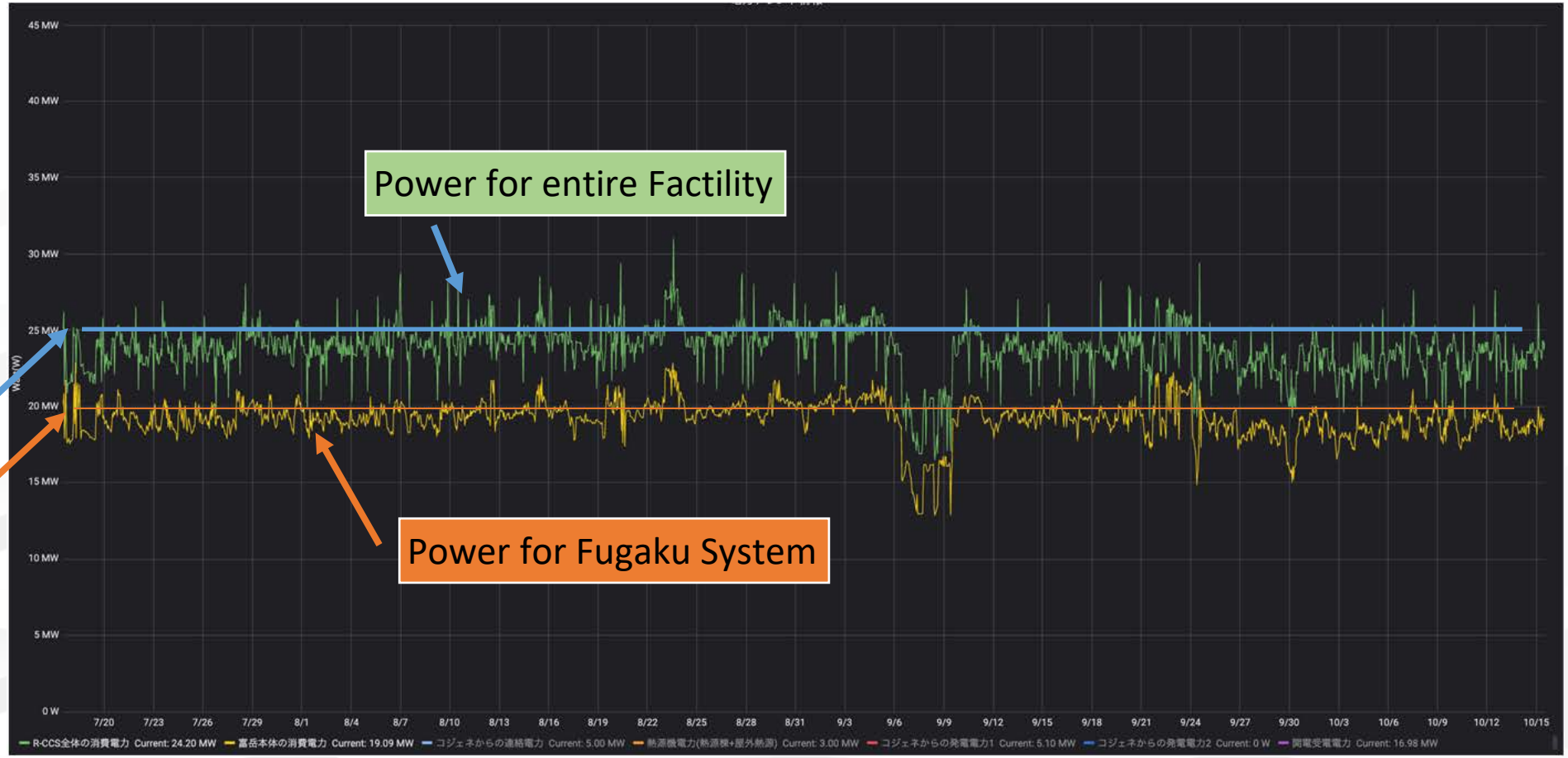

**Performance improvement by SWP in Livermore Kernels by Fujitsu compiler**

# Power consumption of Fugaku systems

- **Actual Power consumption of Fugaku is about 20MW**
  - Less than that we estimated at design time (30MW-40MW)

Power profile of Fugaku
(from July to Sep, 2021)

Utilization ratio was around 70-80%



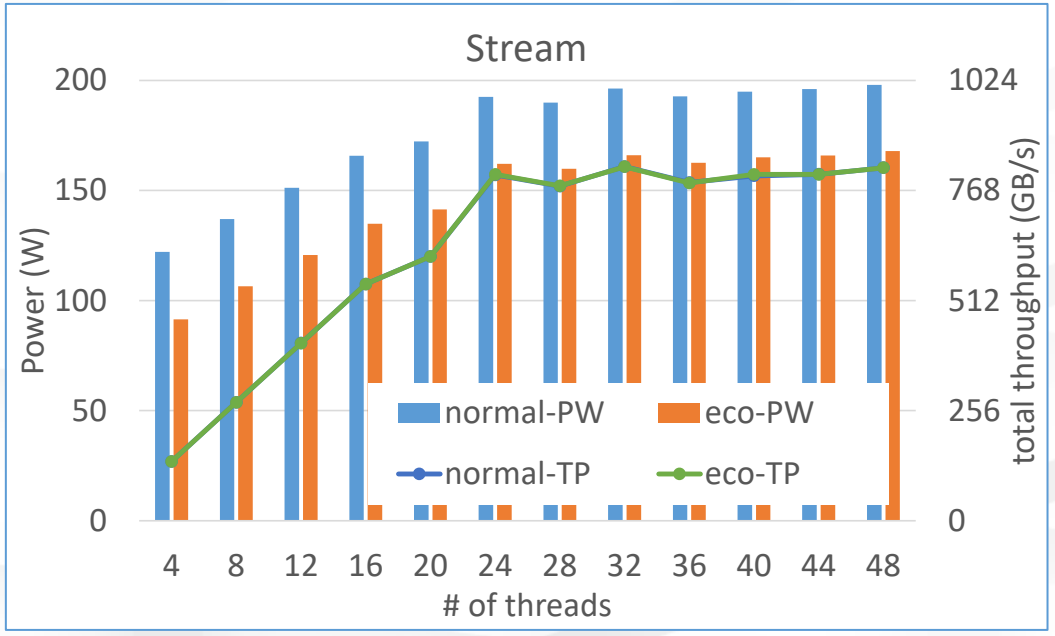

25 MW

20 MW

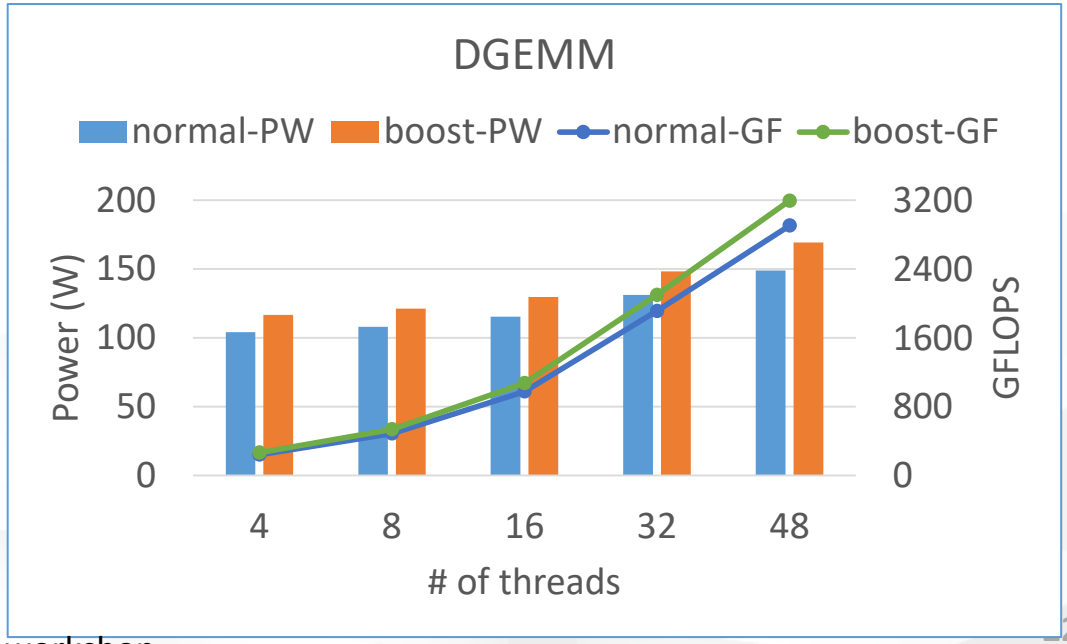
Note: “Power for entire Faciltiy” includes the power for cooling systems

# A64FX power mode: Boost mode (2.2GHz) & Eco mode (1 SIMD pipeline)

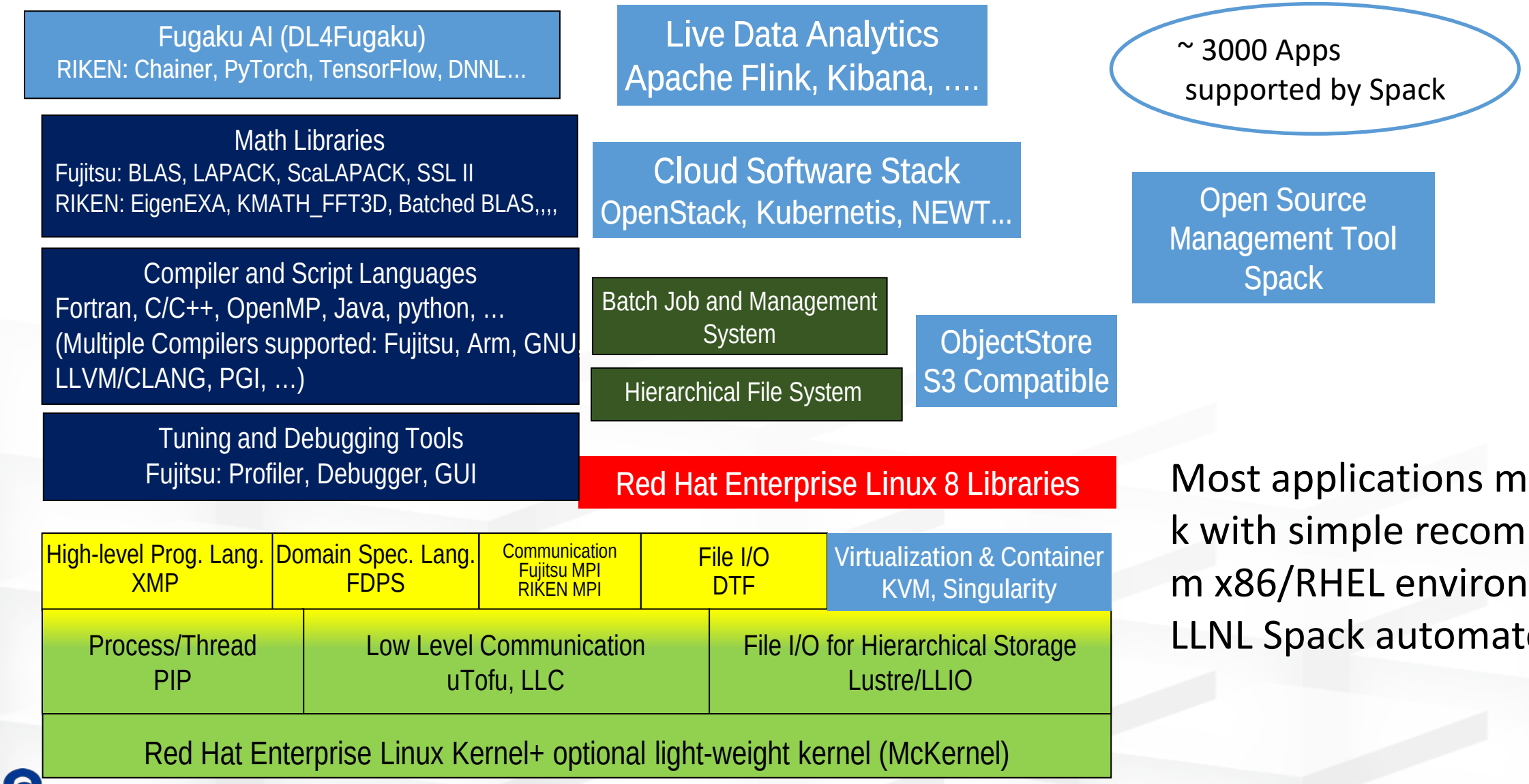
- **Power & Performance of STREAM using Eco mode**
  - The performance is almost the same as that in normal mode (24 threads hits 80% of peak memory bandwidth
  - The power increases upto 24 threads.
  - 15%-25% reduction comparing to that in normal mode.




- **Power & Performance of DGEMM (in Fujitsu Lib) using Boost mode**
  - Reach to 95% out of peak performance
  - The performance is 10% better than that in normal mode.
  - The power increases by 13.7%
  - The power-efficiency decreases by 3.3 %



# Fugaku System Software Stack




Most applications may wor
k with simple recompile fro
m x86/RHEL environment.
LLNL Spack automates this.

# System software and Programming models & languages for “Fugaku”

- Standard programming model is OpenMP (for NUMA node(CMG)) + MPI
  - Both OpenMPI (by Fujitsu) and MPICH (by Riken) are supported.
  - 4 compilers (Fujitsu, gcc, LLVM/Arm, Cray),  OpenMP 4.x is supported.
  - uTofu low-level comm. APIs for Tofu-D interconnect.
- Container and Virtual machine (KVM, Singularity, …)
- DL4Fugaku: AI framework for A64FX and Fugaku, used in Chainer, PyTorch, TensorFlow
- Many Open-source software are already ported using Spack

- System software and Programming tools, Math-Libs developed by RIKEN
  - McKernel: Light-weight Kernel enabling jitter-less environment for large-scale parallel program execution.
  - XcalableMP directive-based PGAS Language
  - FDPS: DLS for Framework for Developing Particle Simulators.
  - EigenExa: Eigen-value math library for large-scale parallel systems.

# XcalableMP(XMP) http://www.xcalablemp.org

- **What's XcalableMP (XMP for short)?**
  - A PGAS programming model and language for distributed memory , proposed by **XMP Spec WG**
  - XMP Spec WG is a special interest group to design and draft the specification of XcalableMP language. It is now organized under **PC Cluster Consortium**, Japan. Mainly active in Japan, but open for everybody.

- **Project status (as of June 2019)**
  - XMP Spec **Version 1.4** is available at XMP site. new features: mixed OpenMP and OpenACC , libraries for collective communications.
  - Reference implementation by U. Tsukuba and Riken AICS: **Version 1.3.1 (C and Fortran90)** is available for PC clusters, Cray XT and K computer. Source-to- Source compiler to code with the runtime on top of MPI and GasNet.
- **HPCC class 2 Winner 2013. 2014**

**The spec of XcalableMP 1.x is now converged.**
**We are now moving to XcalableMP 2.0 with global task-based parallel programming and PGAS**

- **Language Features**
  - Directive-based language extensions for Fortran and C for PGAS model
  - Global view programming with global-view distributed data structures for data parallelism
    - SPMD execution model as MPI
    - pragmas for data distribution of global array.
    - Work mapping constructs to map works and iteration with affinity to data explicitly.
    - Rich communication and sync directives such as "gmove" and "shadow".
    - Many concepts are inherited from HPF
  - Co-array feature of CAF is adopted as a part of the language spec for local view programming (also defined in C).

Code example

```
int array[YMAX][XMAX];

#pragma xmp nodes p(4)
#pragma xmp template t(YMAX)
#pragma xmp distribute t(block) on p
#pragma xmp align array[i][*] to t(i)

main(){
  int i, j, res;
  res = 0;

#pragma xmp loop on t(i)  reduction(+:res)
  for(i = 0; i < 10; i++)
    for(j = 0; j < 10; j++){
      array[i][j] = func(i, j);
      res += array[i][j];
    }
}
```

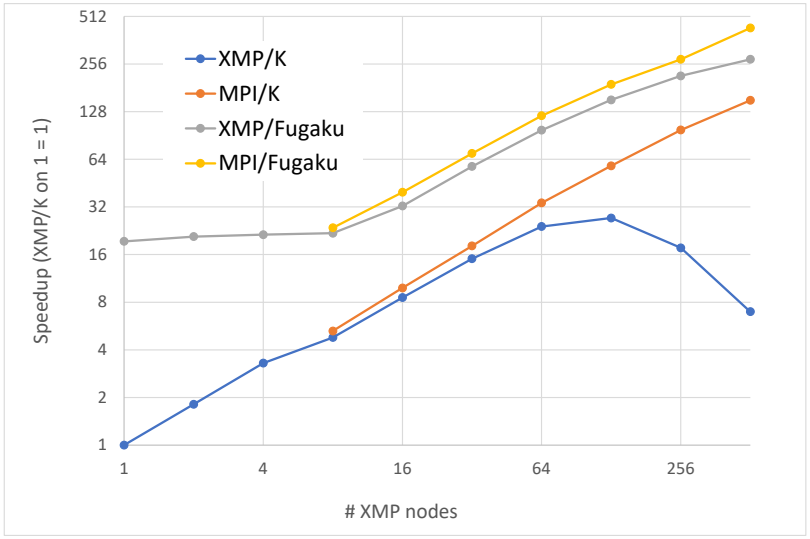
data distribution

add to the serial code : incremental parallelization

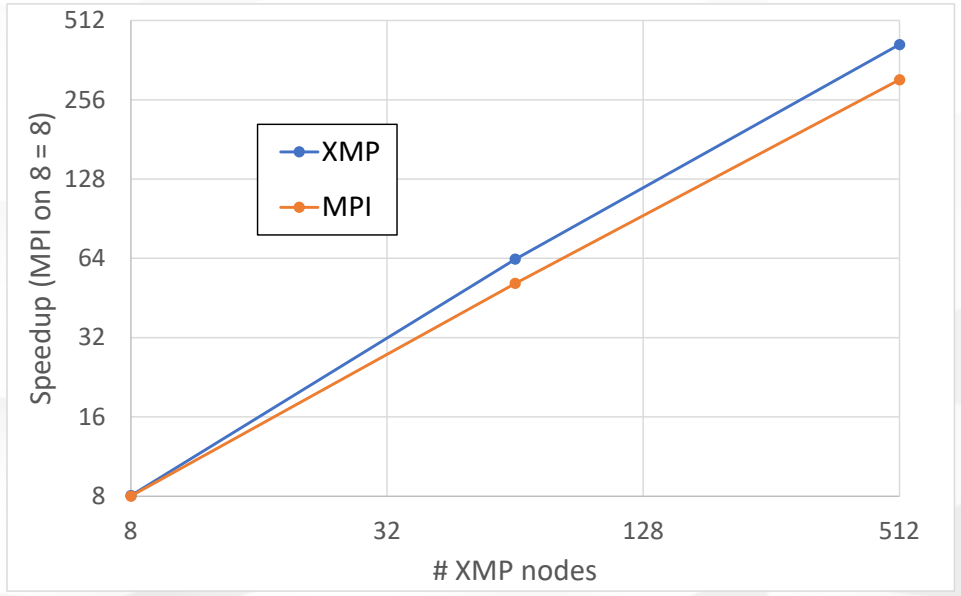
work sharing and data synchronization

# Performance of XcalableMP on Fugaku

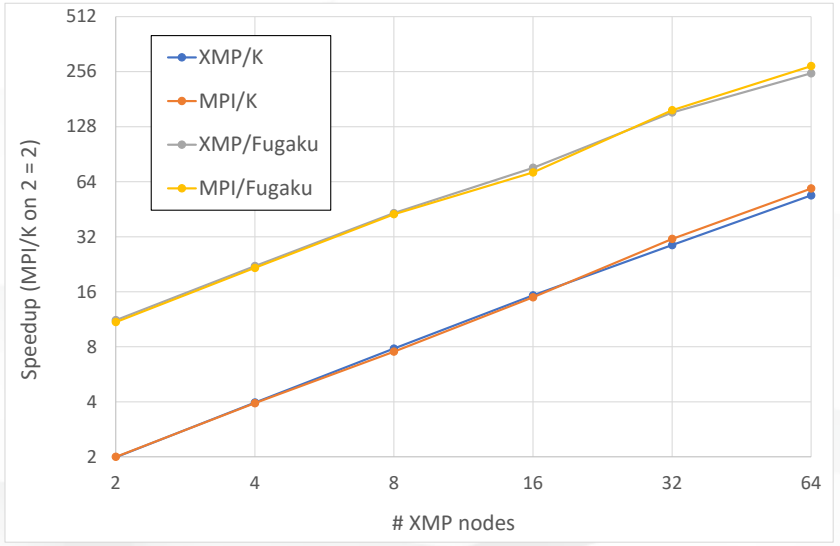
- XcalableMP was taken as a parallel programming language project for improving the productivity and performance of parallel programing.
- XcalableMP is now available on Fugaku and the performance is enhanced by the Fugaku interconnect, Tofu-D.




QCD (Local view programming, Coarray)



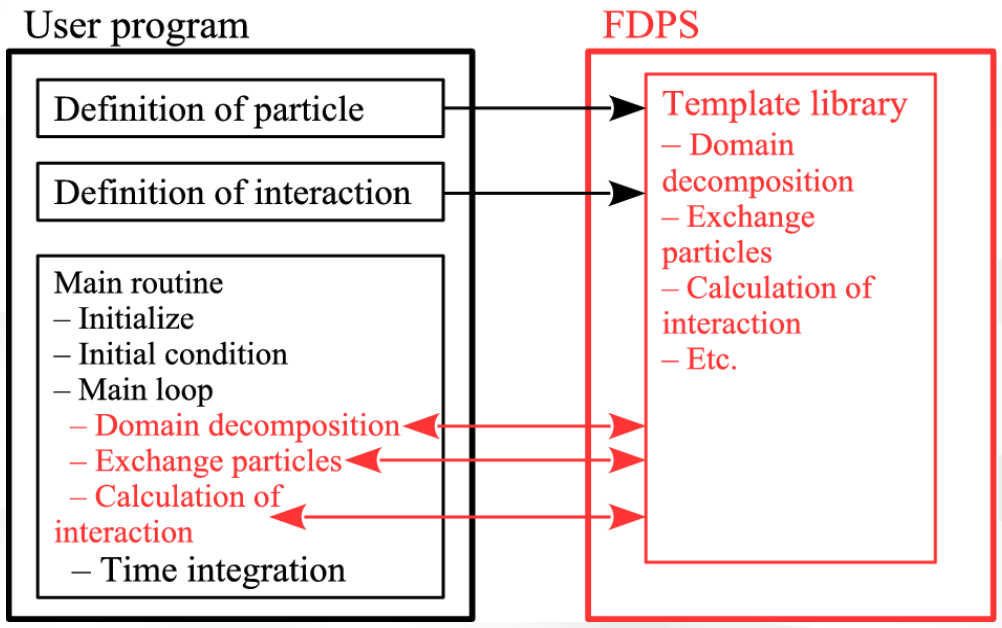

Impact-3D (global view, stencil apps)
Fusion simulation code




NT-Chem (local view programming, Coarray)

# **FDPS:** a framework for developing parallel particle simulation codes

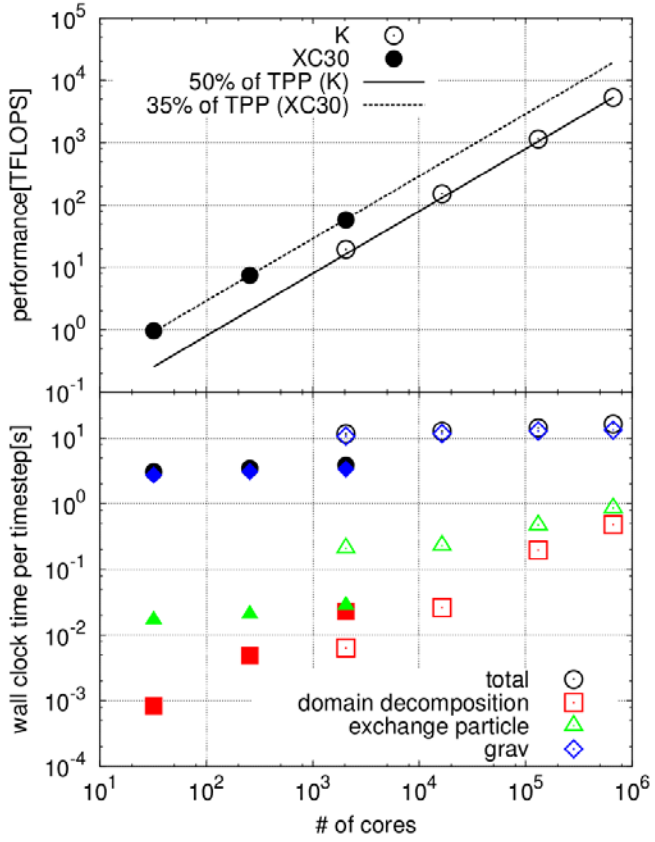
- Developed by Prof. Makino's group, R-CCS
- Basic idea: "abstract" code for
  - domain decomposition
  - particle exchange
  - parallel O(N log N) interaction calculation




Basic concept of FDPS. The user program gives the definitions of particle and interaction to FDPS, and calls ...

- Implemented as a template class library in C++.
- A single program can run on a notebook, a cluster of Intel servers, and the entire K computer, without change (Well, interaction function needs some optimization)
- Works also on GPGPUs

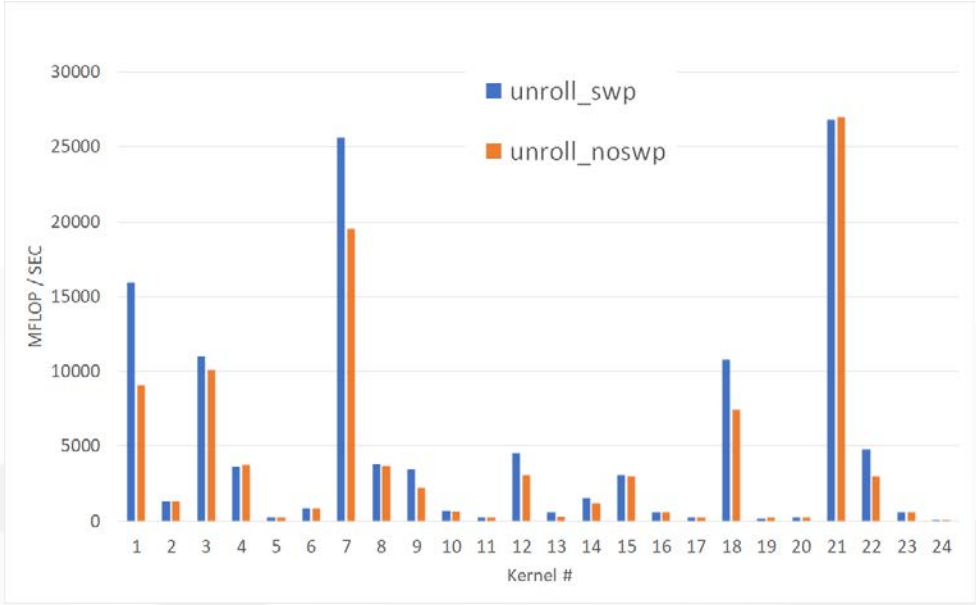
Gravitational N-body (270k/process) Weak scaling performance pretty good for up to all nodes of K computer



# Challenges of programming models for Fugaku

- **Challenges on programming for Massive parallelism**
  - Task-based programming models for “Fugaku”
    - to exploit parallelism of SIMD and manycore for A64FX, and enables overlapping comp. and comm.
  - OpenMP 4.0 task + MPI (Multithread-aware MPI)
  - XcalableMP 2.0 is being designed for task-based programming on global address space (PGAS) (in next slide)

- **How to exploit SIMD**
  - SIMD is a key for performance on A64FX
  - OpenMP SIMD directives
  - Compiler optimization (Fujitsu compiler)
    - SWP: software pipelining, loop fission, …
  - OpenCL for SVE (Arm SIMD)
- **Comm Optimization by Low-level layer, uTofu.**



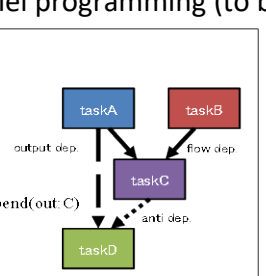

Performance improvement by SWP in Livermore Kernels by Fujitsu compiler

# Research agenda for XcalableMP 2.0

- **Task-based programming on global address space (PGAS) using the description of data distribution.**
- **XMP-API: “compiler-free” approach based on XMP concept including C++ template wrapping, as well as directive-based extension**
- **Using low-level one-sided communication layers such as UCX, designed for global task parallel run-time in multi-threaded environment.**
- **Task migration (like Charm++)**
- **Various kinds of Task offloading**
  - Offload within a node,
    - Offload to GPU, FPGA using OpenCL
    - Offload to SIMD (Integration with kokkos, oneAPI DPC++)
  - Offload outside nodes
    - Offload to cluster of accelerators
      - ESSPER project (Dr. Sano)
    - For “modular” computing

OpenMP 4.0 task-parallel programming (to be extended to XMP)

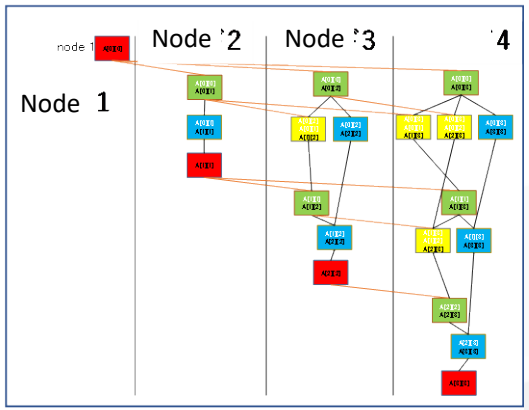
```
#pragma omp parallel
#pragma omp single
{
  int A, B, C;
#pragma omp task depend(out:A)
  A = 1;      /* taskA */
#pragma omp task depend(out:B)
  B = 2;      /* taskB */
#pragma omp task depend(in:A, B) depend(out:C)
  C = A + B;  /* taskC */
#pragma omp task depend(out:A)
  A = 3;      /* taskD */
}
```




XcalableMP data distribution

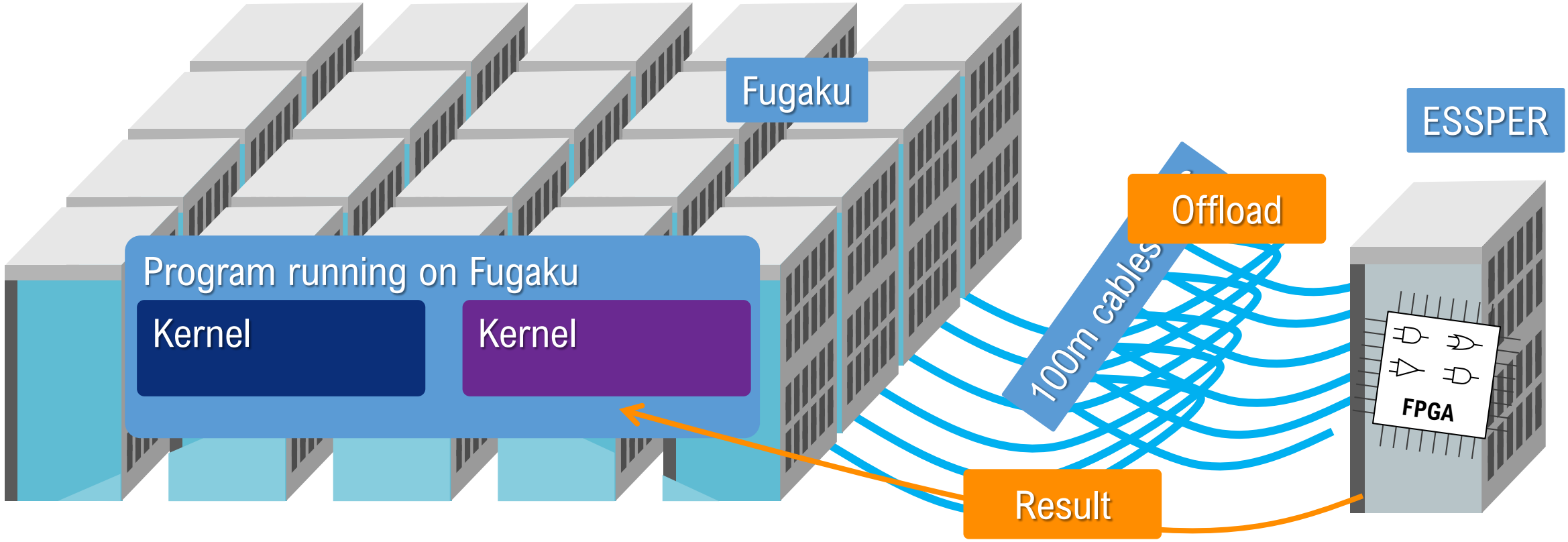
```
#pragma xmp nodes p(NUM_COLS, NUM_ROWS)
#pragma xmp template t(0:NA-1,0:NA-1)
#pragma xmp distribute t(block, block) onto p
#pragma xmp align w[i] with t(*,i)
#pragma xmp align q[i] with t(i,*)
double a[NZ];
int rowstr[NA+1], colidx[NZ];
…
```

XcalableMP global task-paralle



# ESSPER: Elastic and Scalable System for high-PErformance Reconfigurable computing

- Experimental prototype to extend existing HPC systems -- supercomputer Fugaku -- with FPGAs
- Fugaku and ESSPER, a cluster of FPGAs, are connected by 100G IB cables



# OpenMP task + Remote Procedure Call (RPC)

```
void cholesky(const int ts, const int nt, double* A[nt][nt]){
    OmniRpcRequest rq1, rq2, rq3, rq4;

#pragma omp parallel private(rq1, rq2, rq3, rq4)
#pragma omp single
    for (int k = 0; k < nt; k++) {
#pragma omp task depend(out:A[k][k])
{
        portl(ts, ts, A[k][k]);
}

        for (int i = k + 1; i < nt; i++) {
….
#pragma omp task depend(in:A[k][k]) depend(out:A[k][i] )
{
            rq3 = OmniRpcCallAsync("rpc_dgemm", ….., A[k][i])
}
        }
```

```
Define rpc_dgemm(IN int ts …. double A[ts][ld])
{
    … invoke dgemm implemented on FPGA
}
```

- OmniRPC is a grid PRC library supporting master-worker parallel programming
- A task in a thread offloads a pre-defined kernel to remote node(s)
- Remote node can consist of CPUs, GPUs, FPGAs, etc..
- **OmniRPC is to be extended** to manage the requests que from tasks to avoid blocking threads, and designed with OpenMP task management

# Programming models for beyond “Fugaku”

- **Future systems beyond Fugaku will be heterogenous parallel system with some accelerators**
- **Programming models for accelerator-based heterogenous parallel system**
  - Task-based offloading to accelerators (Fugaku has no accelerators.)
  - Task migration with accelerators
  - “MIMD-to-MIMD” offloading
    - XcalableACC: integration of XcalableMP and OpenACC
- **Programming models and support for “Modular” computing platforms**
  - Offloading with “RPC”
  - Workflow programming between systems with different characteristics
  - “smart” tasks (or, job) allocation and management  including migrations and load-balancing

# Concluding remarks

- **We have confirmed that 3 KPIs were achieved:**
  - Power-efficiency ⇒ Actually, Fugaku is running around at 20MW with 70% utilization
  - Effective Performance of applications. ⇒ Many apps are running more efficient than expected.
  - Ease-of-use ⇒ easy for porting OpenMP+MPI programs without any accelerator programming.
- **A64FX is a manycore processor designed for HPC workload.**
  - Performance tuning may be required to exploit A64FX arch and HBM ..
- **Several international collaborations are going on:**
  - DOE-MEXT collaborations
    - Arm Arch collaboration (SNL/NNSA, U. Bristol)
    - Spack for Fugaku (LLNL, going-on)
    - ECP software porting & evaluation (on-going)
  - CEA@France, A*STAR@Singapore, BSC, …