Executing high performance and energy efficient jobs on supercomputer Fugaku

May 2025

This document is based on the material from "The 1st R-CCS/RIST Joint Seminar on Advanced use of Supercomputer Fugaku and Arm computer systems, 2025/4/23".
The power consumption [Wh] per job is used as the primary metric of energy in this document.

contents

Major power knobs for Fugaku

This combination is called BOOST ECO mode

name	feature	Control point	Description, acceptable values
freq	Change frequency	CPU Socket	Control CPU frequency (unit in Hz) ・2000000000 (called NORMAL mode) ・2200000000 (called BOOST mode)
throttling_state	Memory access control	Memory	Restrict the bus use rate between the memory access controller and memory 0, 1, 2, .. , 9: 0:use rate100% (highest perf.), 1: 90%, 2: 80%, .. , 9: 10%
issue_state	Instruction issue control	CPU Core	Restrict the instruction issue rate for the computing core 0: 4 instructions 1: 2 instructions
ex_pipe_state	EXA only	CPU Core	Reduce the number of pipelines for Integer operations 0: user both of EXA and EXB pipelines 1: use EXA pipeline only
eco_state	Combination of eco mode and FLA only control	CPU Core	Reduce the number of pipelines for floating point operations. Additionally apply the further power saving feature named ECO. 0: ECO off and use both FLA and FLB. 1: ECO off and use FLA only. 2: ECO on and use FLA only. This setting is called ECO mode.
retention_state	Retention control	CPU Core	Controll the transition to a lower power state, named Retention, when there is no running process. 0: no transition 1: transition to Retention state

power knob control and report for Fugaku

- Methods to apply power knobs
  - A) Job submit option/directive (*1): applied over the job, i.e. same value every where
  - B) Call PMlib inside apps (*2) : applied at control point, Fortran/C/C++/Python API
  - C) Call PowerAPI inside apps (*3) : applied at control point, C API

- Checking power consumption
  - A) Job statistic file (*.stats): power consumption per job
  - B) Call PMlib inside apps : report per user specified section
  - C) Call PowerAPI inside apps : format your own report per user specified section
  - D) Repeat 11 fapp runs and postprocess with fapppx (*4) : Excel report per user specified section. Fortran/C/C++ API

(*1) Fugaku Users Guide 7.2. Power mode
(*2) open source PMlib tutorial
(*3) Fugaku Users Guide 7.3 Power API
(*4) Profiler User's Guide 3. advanced profiler, 4. CPU performance report

Applying power knob examples

- A) specify job control option/directive
  - Most convenient. Users Guide - Use and job execution. 7.2 Power mode
  - Note that it takes effect throughout the execution of the submit job

```
# Job submit option/directive
#PJM --rsc-list "freq=2000"
#PJM --rsc-list "eco_state=2"
#PJM --rsc-list "retention_state=0"
```

```
*.stats file displays：
AVG POWER CONSUMPTION OF NODE (IDEAL) : 142.9     (power unit in W)
ENERGY CONSUMPTION OF NODE (IDEAL) : 4.2796       (energy unit in Wh)
```

  - Since April 2025, BOOST ECO mode shown below is applied as the default setting

```
#PJM --rsc-list "freq=2200"
#PJM --rsc-list "eco_state=2”
#PJM -s
```

  - To apply the NORMAL mode setting, include the following option/directive

```
#PJM --rsc-list "freq=2000"
#PJM --rsc-list "eco_state=0”
#PJM -s
```

Applying power knob examples

- B) call PMlib API
  - Open source library. Fortran/C/C++/Python APIs. PMlib repository and tutorial
  - Detail report of computing performance and power consumption at once.
  - On Fugaku, PMlib is available as spack module

```
!cx example in Fortran program

call f_pm_initialize (1)
call f_pm_start (”label")
 .. Computing section to be measured ..
call f_pm_stop (”label")
call f_pm_report ("")
```

```
# job script example
spack load pmlib@10.0-clang
export PMLIB_REPORT=BASIC
export POWER_CHOOSER=NODE
xospastop
./a.out
```

```
Report in the standard output file:

    Report for option HWPC_CHOOSER=FLOPS is generated.
Section      |   HP_OPS    SP_OPS    DP_OPS  Total_FP  [Flops]  [%Peak]
-------------+-------------------------------------------------------
label        : 0.00e+00  0.00e+00  1.60e+11  1.60e+11  7.35e+10  9.57e+00
-------------+-------------------------------------------------------

    Report is generated for POWER_CHOOSER=NODE option.
Estimated power inside node [W]
Section      |  total | CMG+L2  MEMORY  TF+A+U | Energy[Wh]
-------------+--------+-----------------------+-----------
label        :  156.5   106.0    45.8     8.1    4.24e+00
-------------+--------+-----------------------+-----------
```

Applying power knob examples

- C) Call PowerAPI inside apps (typical source and output)　　7.3 Power API

```
#include "pwr.h"
#define MAX_P_OBJ_NODE 20
PWR_Cntxt pacntxt = NULL;
PWR_Time pa64timer[MAX_P_OBJ_NODE][2];
PWR_Obj p_obj_array[MAX_P_OBJ_NODE];
double d_time, ave_watt, w_joule[MAX_P_OBJ_NODE][2];
char p_obj_name[MAX_P_OBJ_NODE][30] = {
  "plat.node", "plat.node.cpu.cmg0.cores",  .. , "plat.node.tofuopt",
};
irc = PWR_CntxtInit(PWR_CNTXT_DEFAULT, PWR_ROLE_APP, "app", &pacntxt);
for (int i=0; i<MAX_P_OBJ_NODE; i++) {
irc = PWR_CntxtGetObjByName (pacntxt, p_obj_name[i], &p_obj_array[i]);
irc = PWR_ObjAttrGetValue (p_obj_array[i], PWR_ATTR_ENERGY, &w_joule[i][0],
&pa64timer[i][0]);
}
// .. 測定したい計算部分 ..
for (int i=0; i<MAX_P_OBJ_NODE; i++) {
irc = PWR_ObjAttrGetValue (p_obj_array[i], PWR_ATTR_ENERGY, &w_joule[i][1],
&pa64timer[i][1]);
d_time = (double)(pa64timer[i][1] - pa64timer[i][0])/1.0e9;
ave_watt = (w_joule[i][1] - w_joule[i][0]) / d_time;
printf("rank %d [%-30s] %f [W]¥n", my_rank, p_obj_name[i], (float)ave_watt);
}
```

```
rank 0 [plat.node                    ] 116.648102 [W]
rank 0 [plat.node.cpu.cmg0.cores     ] 27.563534 [W]
rank 0 [plat.node.cpu.cmg1.cores     ] 22.250008 [W]
rank 0 [plat.node.cpu.cmg2.cores     ] 22.250008 [W]
rank 0 [plat.node.cpu.cmg3.cores     ] 22.250008 [W]
rank 0 [plat.node.cpu.cmg0.l2cache   ] 4.583274 [W]
rank 0 [plat.node.cpu.cmg1.l2cache   ] 1.314807 [W]
rank 0 [plat.node.cpu.cmg2.l2cache   ] 1.312501 [W]
rank 0 [plat.node.cpu.cmg3.l2cache   ] 1.312501 [W]
rank 0 [plat.node.cpu.acores.core0   ] 0.533256 [W]
rank 0 [plat.node.cpu.acores.core1   ] 0.534289 [W]
rank 0 [plat.node.cpu.uncmg          ] 0.312500 [W]
rank 0 [plat.node.cpu.tofu           ] 5.250001 [W]
rank 0 [plat.node.mem0               ] 2.565995 [W]
rank 0 [plat.node.mem1               ] 1.750000 [W]
rank 0 [plat.node.mem2               ] 1.750000 [W]
rank 0 [plat.node.mem3               ] 1.750000 [W]
rank 0 [plat.node.pci                ] 0.000000 [W]
rank 0 [plat.node.tofuopt            ] 2.625001 [W]
```

Applying power knob examples

- D) Repeat fapp runs and produce Excel report
  - Profiler User's Guide 3. advanced profiler, 4. CPU perforr
  - 4-1 repeat fapp runs on compute node

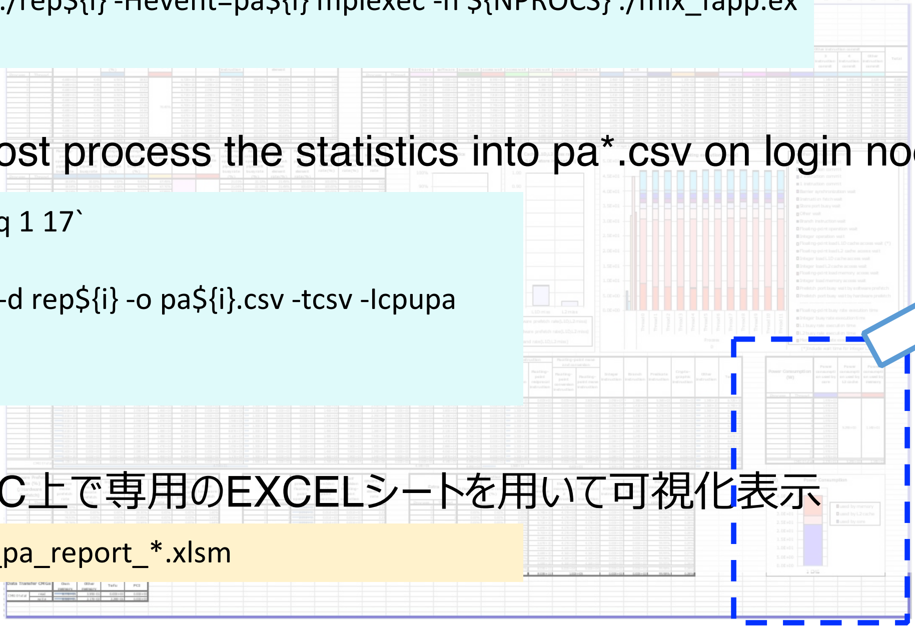
```
for i in `seq 1 17`
do
fapp -C -d ./rep${i} -Hevent=pa${i} mpiexec -n ${NPROCS} ./mix_fapp.ex
done
```

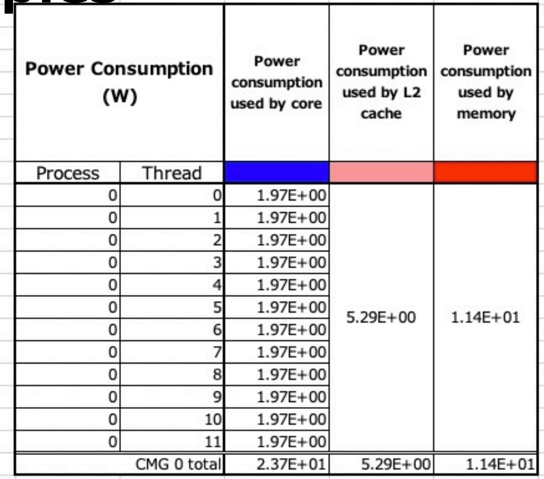
  - 4-2 post process the statistics into pa*.csv on login node

```
for i in `seq 1 17`
do
fapppx -A -d rep${i} -o pa${i}.csv -tcsv -Icpupa
done
pwd
```

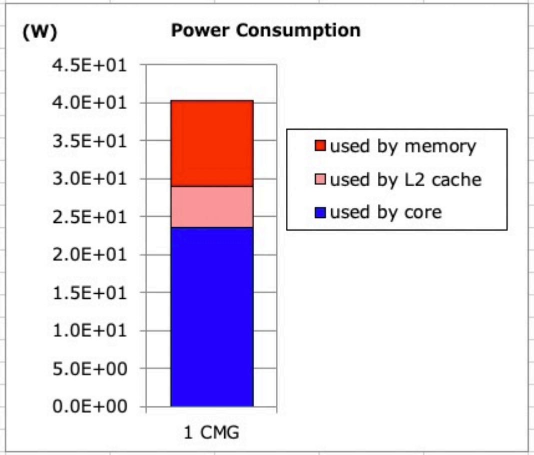
  - 4.3 PC上で専用のEXCELシートを用いて可視化表示

```
open cpu_pa_report_*.xlsm
```



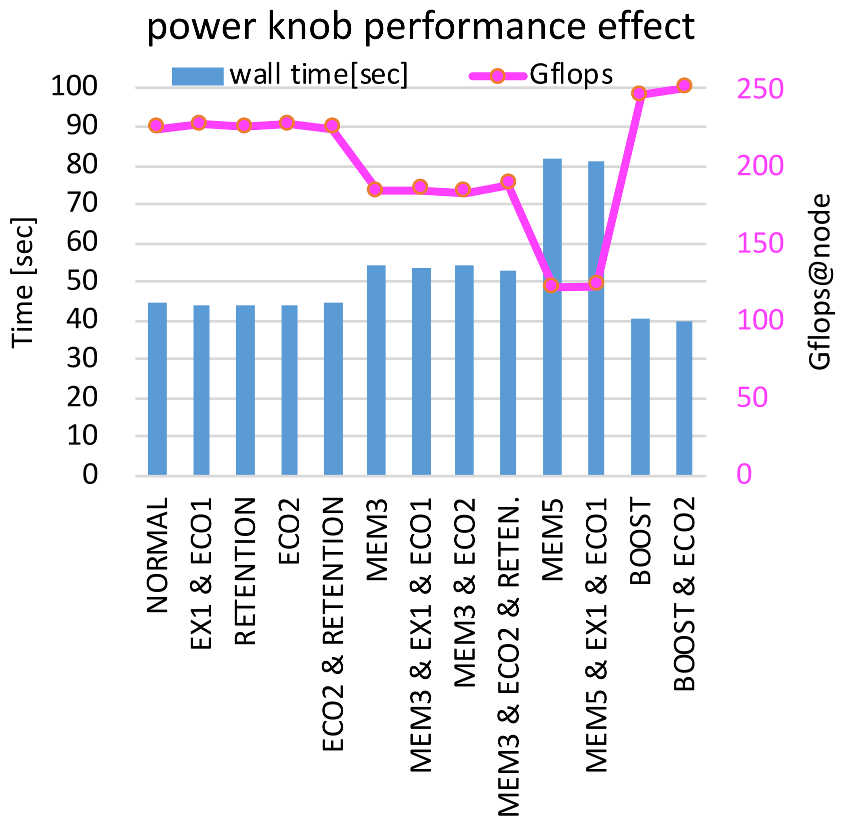


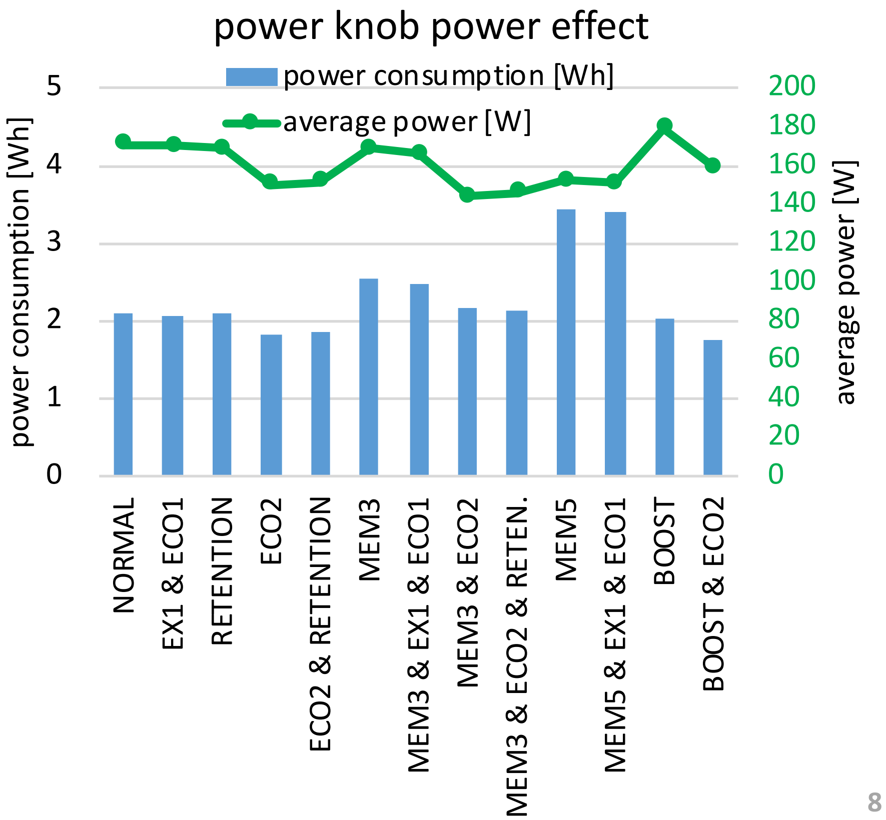
Power Consumption (W)		Power consumption used by core	Power consumption used by L2 cache	Power consumption used by memory
Process	Thread	🟦	🟥	🟥
0	0	1.97E+00	5.29E+00	1.14E+01
0	1	1.97E+00		
0	2	1.97E+00		
0	3	1.97E+00		
0	4	1.97E+00		
0	5	1.97E+00		
0	6	1.97E+00		
0	7	1.97E+00		
0	8	1.97E+00		
0	9	1.97E+00		
0	10	1.97E+00		
0	11	1.97E+00		
CMG 0 total		2.37E+01	5.29E+00	1.14E+01



Performance and power : effect of each power knob

- Examples of power knob effect for highly vectorized kernel





Data precision supported by Fugaku hardware

- The choice of the data type has a significant impact on every aspect of computing accuracy, computing performance, memory size requirement, power consumption

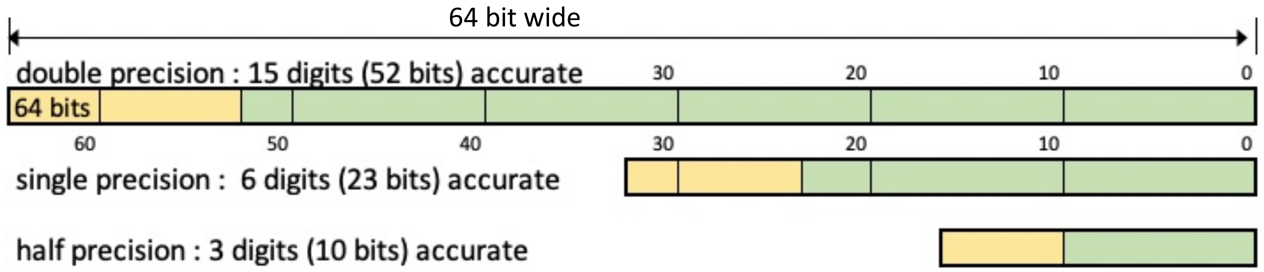
Floating point data supported by the hardware instructions

data type	double precision	single precision	half precision
bits wide	64	32	16
accuracy	15 digits	6 digits	3 digits
fortran	real(kind=8)	real(kind=4)	real(kind=2)
C/C++	double	float	_Float16
Python	float64	float32	float16

Integer data are alike :

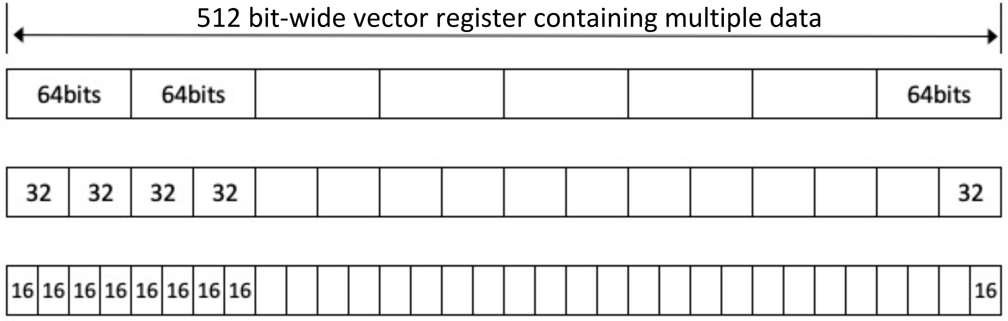
64, 32, 16, 8 bit wide

programming languages support more data types


64 bit wide
double precision : 15 digits (52 bits) accurate
64 bits
30
20
10
0
60
50
40
30
20
10
0
single precision : 6 digits (23 bits) accurate
half precision : 3 digits (10 bits) accurate

Vectorization of arithmetic operations

- Fugaku CPU supports 512 bit-wide vector operation – SVE



8 dp F.P.ops [(*1)] executed at once by SVE

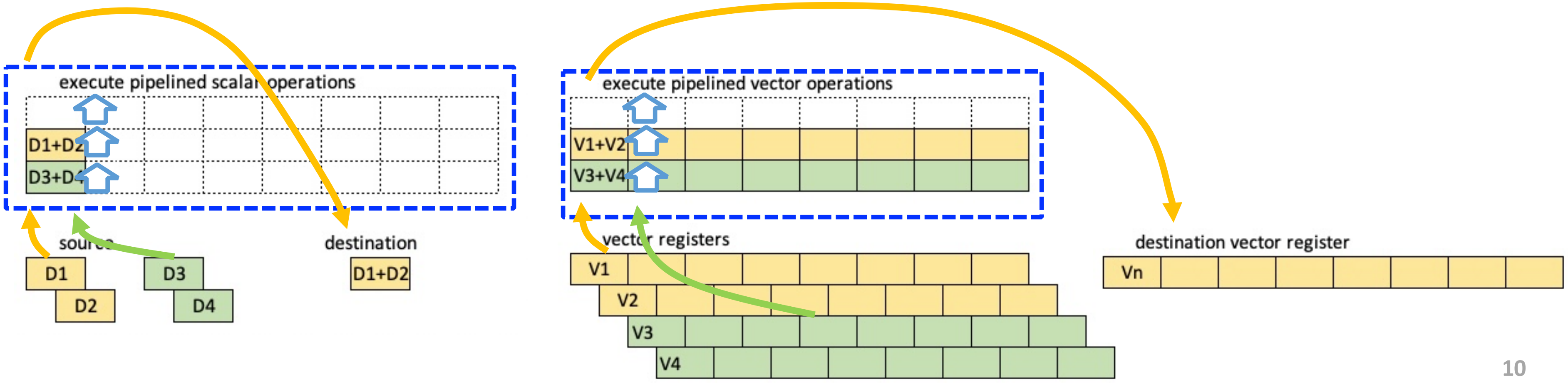
[(*1)] 16 dp F.P.ops with FMA

16 sp F.P.ops[(*2)] executed

[(*2)] 32 sp F.P.ops with FMA

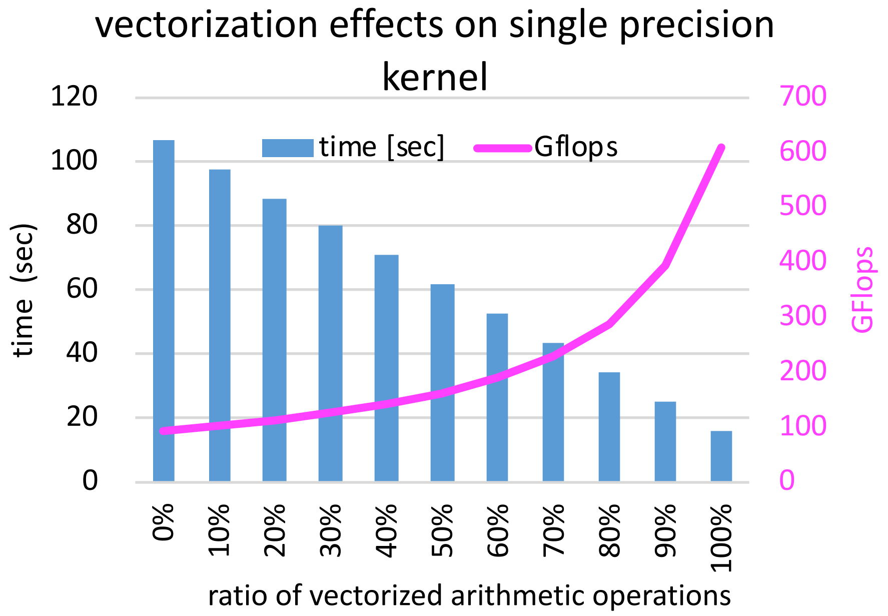
32 hp F.P.ops[(*3)] executed

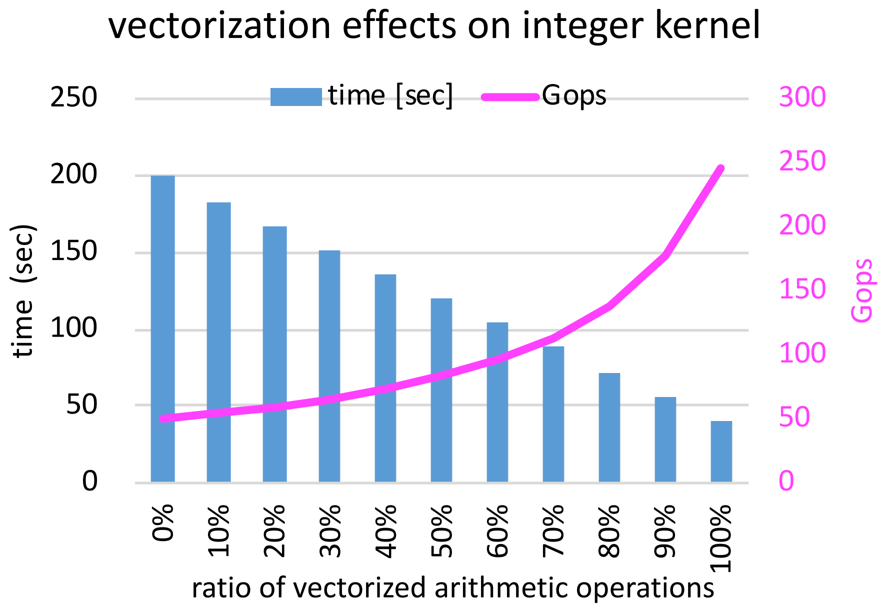
[(*3)] 64 hp F.P.ops with FMA


execute pipelined scalar operations
D1+D2
D3+D4
source
D1
D2
D3
D4
destination
D1+D2
execute pipelined vector operations
V1+V2
V3+V4
vector registers
V1
V2
V3
V4
destination vector register
Vn

Vectorization and its performance effects

- Typical performance gain of vectorized operation
  - Utilizing SVE operation is essential for achieving HPC in your application
  - Achieving high performance leads to saving energy for the job (see later slides)





case study of performance and power consumption

- Let's pickup a simple apps kernel which can be easily examined and compared from the following perspectives
  - Parameter
    - Data precision type, i.e. double precision real, single precision real, half precision
    - Vectorization effect, i.e. vectorized kernels vs. scalar kernels
    - Power knob effect, mostly NORMAL mode vs. BOOST ECO mode
  - Evaluation axis
    - computing time, computational performance
    - power consumption [Wh], average power consumption [W]
  - Assumed parallel model
    - 4 process per node x 12 threads per process
  - Various comparisons are based on per node basis

case study of performance and power consumption

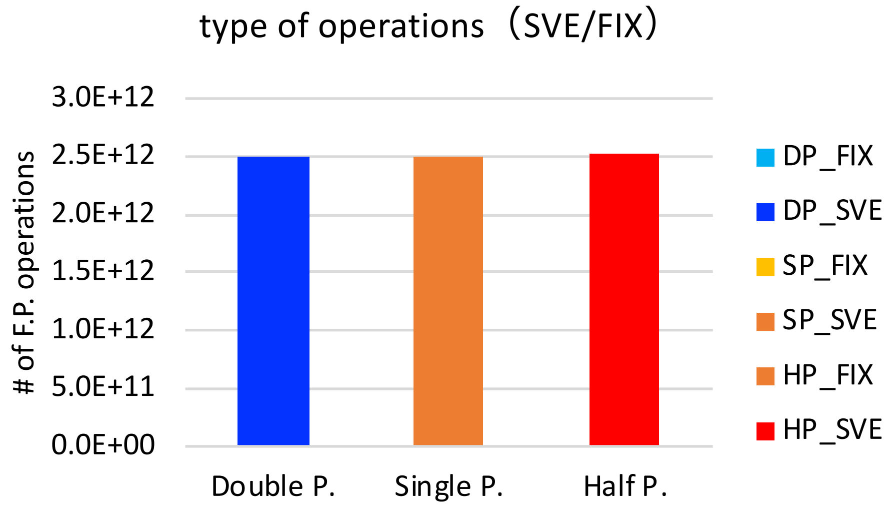
- apps kernel (1)

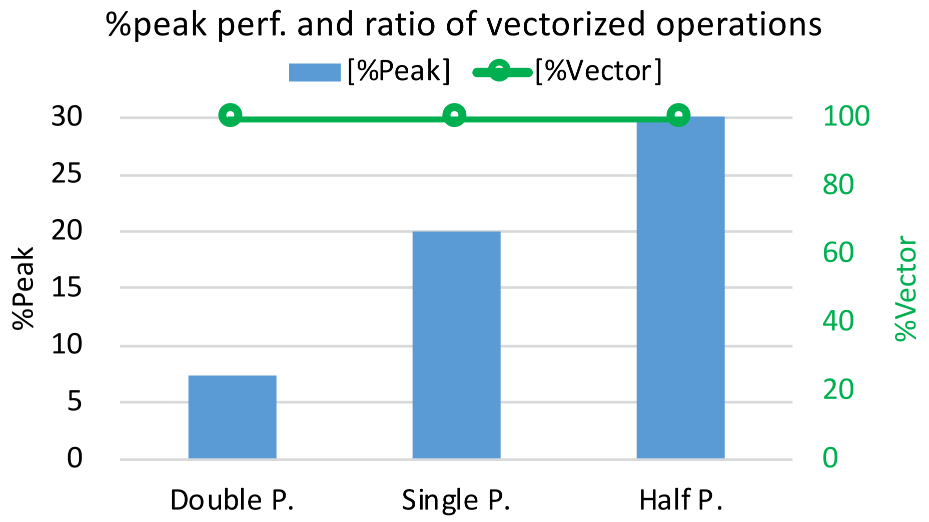
```
!cx 行列積と同内容
!cx 見かけ上のロードストア 3L+1S、実効的には 2L+1S

module data_type

real(kind=2),allocatable :: a2h(:,:), b2h(:,:), c2h(:,:)
real(kind=4),allocatable :: a2s(:,:), b2s(:,:), c2s(:,:)
real(kind=8),allocatable :: a2d(:,:), b2d(:,:), c2d(:,:)
real(kind=2) r_half
real(kind=4) r_single
real(kind=8) r_double
end module data_type

subroutine sub_dmix(n)
use data_type
integer :: n
!$omp parallel do private(i,j,k)
do k=1,n
do j=1,n
do i=1,n
c2d(i,j) = c2d(i,j) + a2d(i,k)*b2d(k,j)
end do
end do
end do
!$omp end parallel do
r_double = c2d(1,n) + c2d(n,1)
return
end subroutine
```

```
subroutine sub_smix(n)
use data_type
integer :: n
!$omp parallel do private(i,j,k)
do k=1,n
do j=1,n
do i=1,n
c2s(i,j) = c2s(i,j) + a2s(i,k)*b2s(k,j)
end do
end do
end do
!$omp end parallel do
r_single = c2s(1,n) + c2s(n,1)
return
end subroutine

subroutine sub_hmix(n)
use data_type
integer :: n
!$omp parallel do private(i,j,k)
do k=1,n
do j=1,n
do i=1,n
c2h(i,j) = c2h(i,j) + a2h(i,k)*b2h(k,j)
end do
end do
end do
!$omp end parallel do
r_half = c2h(1,n) + c2h(n,1)
return
end subroutine
```

Data precision and performance : vectorized kernel

- apps kernel (1)@ compiled with fast option
  - Compiler produced fused-multiply-add SVE instructions
  - Sustained performance of double, single and half precision arithmetic are compared



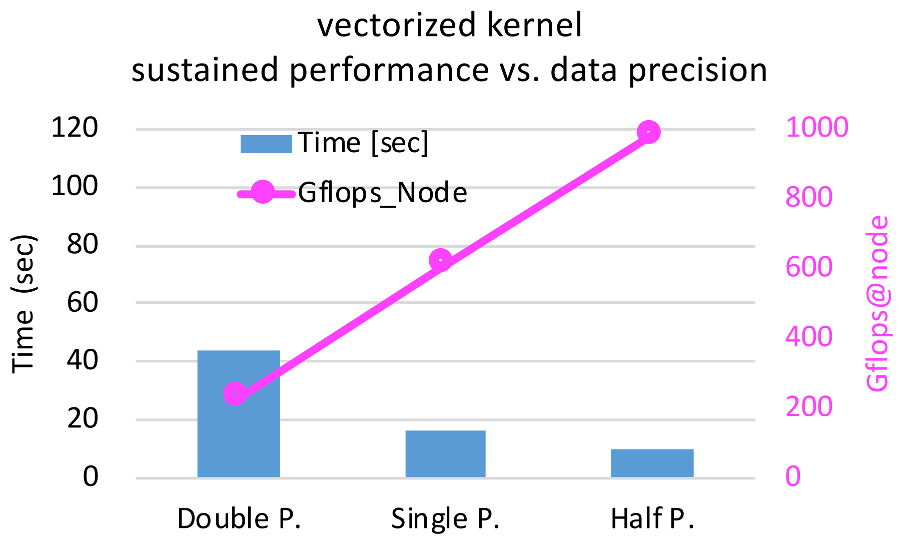


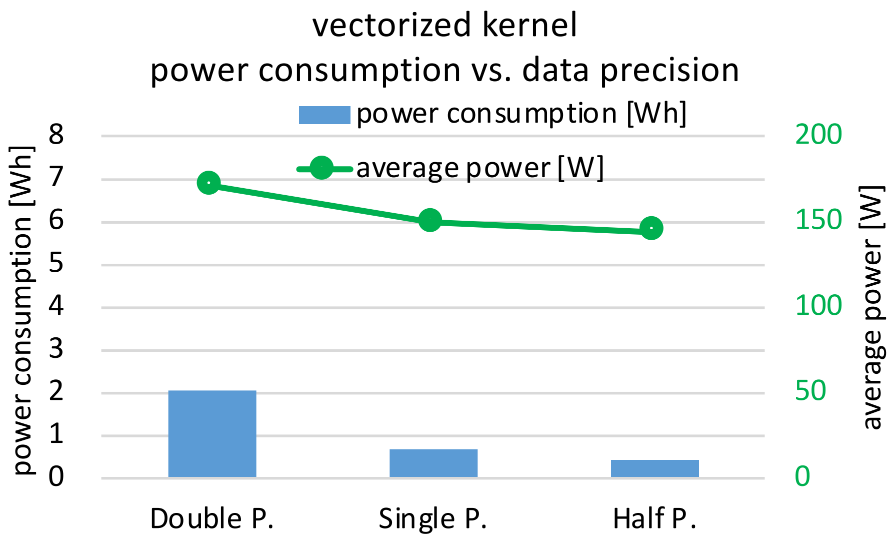
DP_FIX : Double P. scalar ops, DP_SVE : Double P. SVE ops
SP_FIX : Single P. scalar ops, SP_SVE : Single O. SVE ops
HP_FIX : Half P. scalar ops, HP_SVE : Half P. SVE ops

%peak perf. : rate of sustained performance against the theoretical peak performance in double precision at normal mode.

Data precision and performance : vectorized kernel

- apps kernel (1)@ compiled with fast option
  - Lowering data precision has a large effect on both the performance(flops) and the power consumption (Wh)



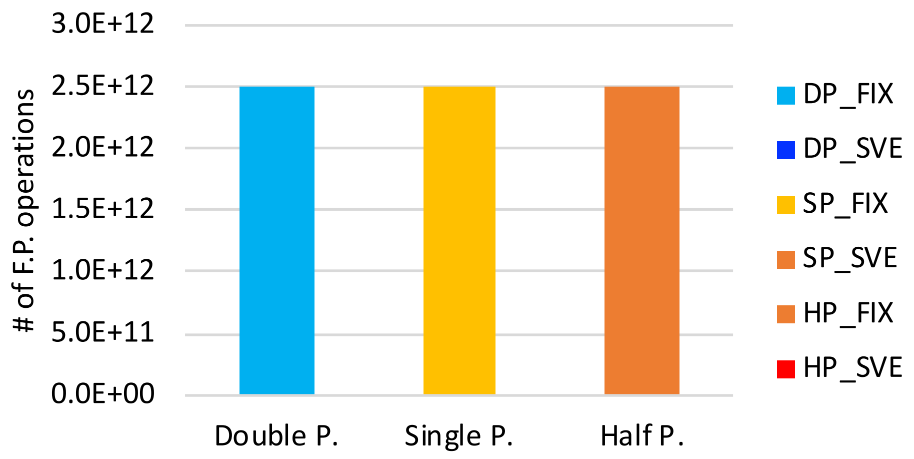


If lowering the data precision is acceptable, it will provide much faster computation and efficient power use for the vectorized applications.

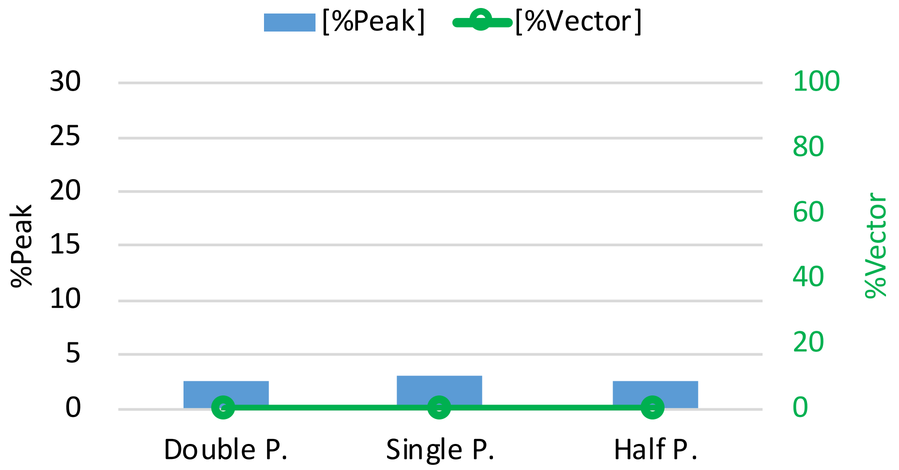
Data precision and performance : scalar kernel

- apps kernel (1)@ compiled with nosimd option
  - Scalar instructions
  - Sustained performance of double, single and half precision arithmetic are compared

type of operations （SVE/FIX）



%peak perf. and ratio of vectorized operations

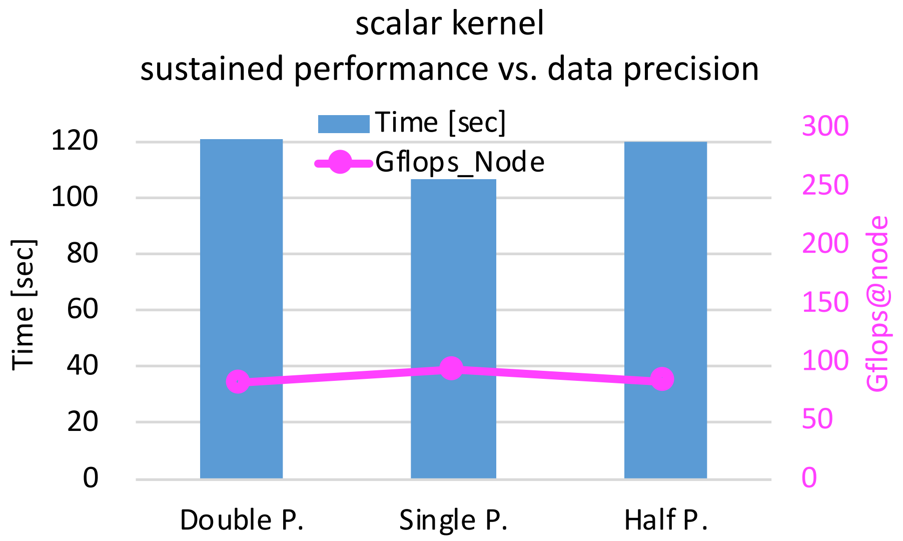


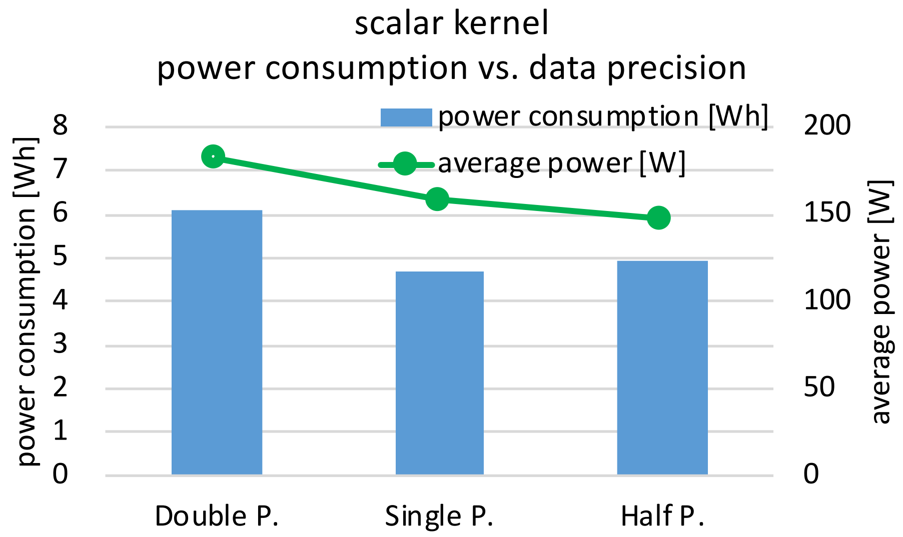
DP_FIX : Double P. scalar ops, DP_SVE : Double P. SVE ops
SP_FIX : Single P. scalar ops, SP_SVE : Single O. SVE ops
HP_FIX : Half P. scalar ops, HP_SVE : Half P. SVE ops

%peak perf. : rate of sustained performance against the theoretical peak performance in double precision at normal mode.

Data precision and performance : scalar kernel

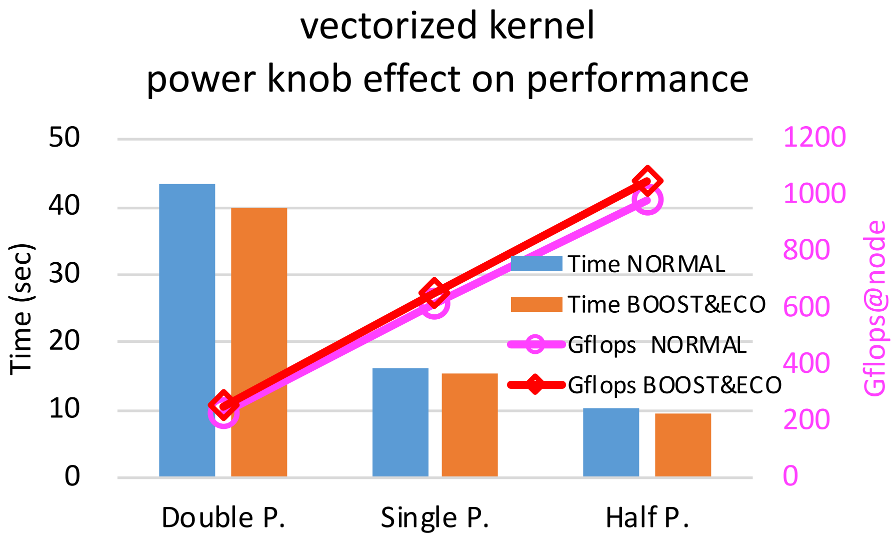
- apps kernel (1)@ compiled with nosimd option
  - Lowering the data precision did not affect performance so much
  - The power consumption was reduced by 20% for Double->Single.
  - The result of half precision is questionable. TBA.

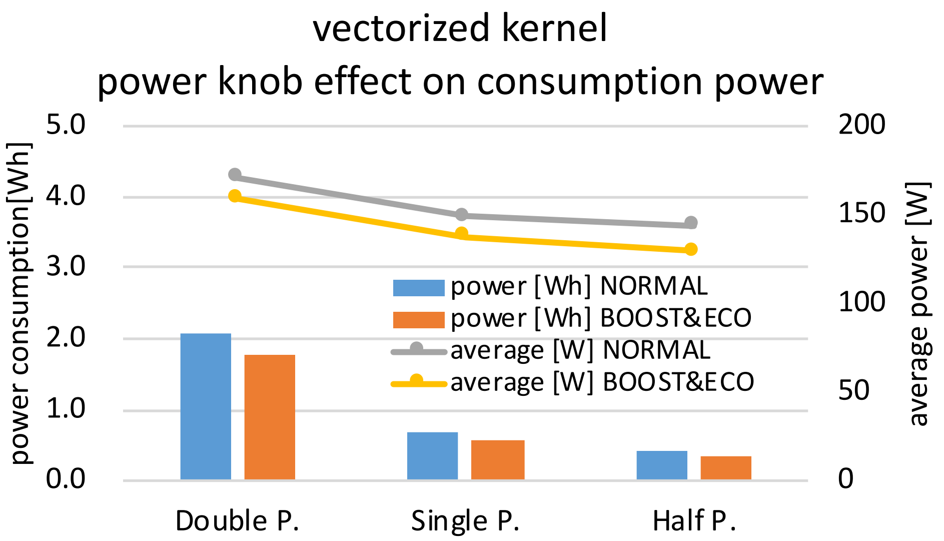




Power knob and performance : vectorized kernel

- apps kernel (1)@ compiled with fast option
  - NORMAL mode and BOOST ECO mode results are compared for each data precision

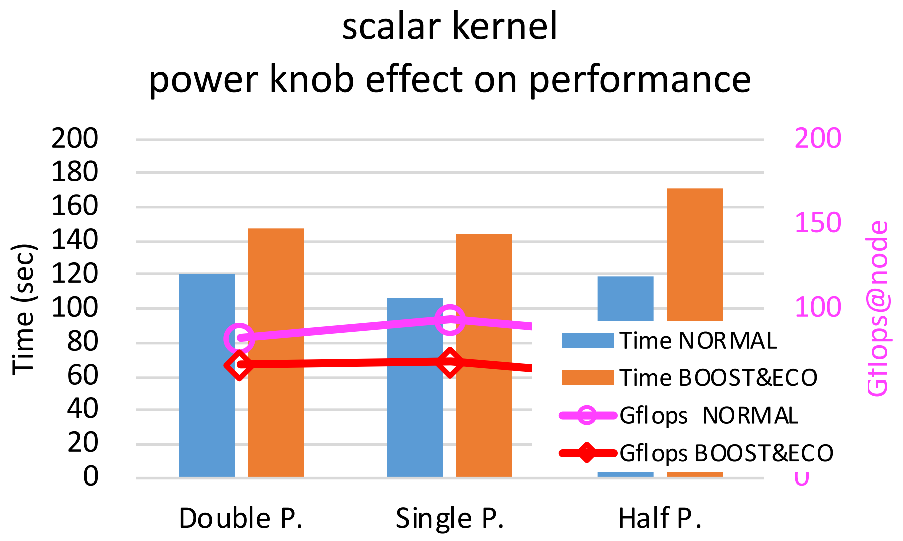


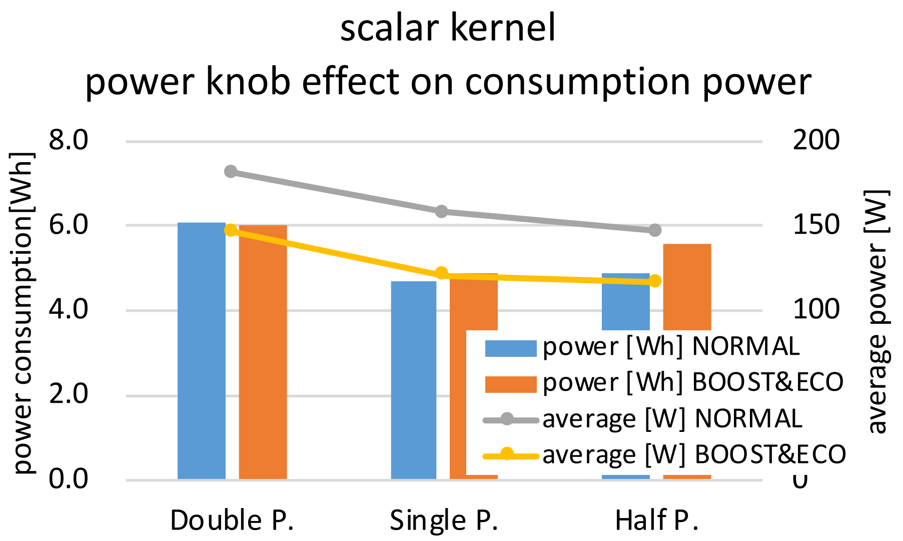


- For this vectorized kernel, BOOST&ECO mode delivers higher performance with lower power consumption than NORMAL mode
- This is true to vector kernels in general. Kernels causing pipeline busy rate of >100% do not apply.

Power knob and performance : scalar kernel

- apps kernel (1)@ compiled with nosimd option
  - NORMAL mode and BOOST ECO mode results are compared for each data precision

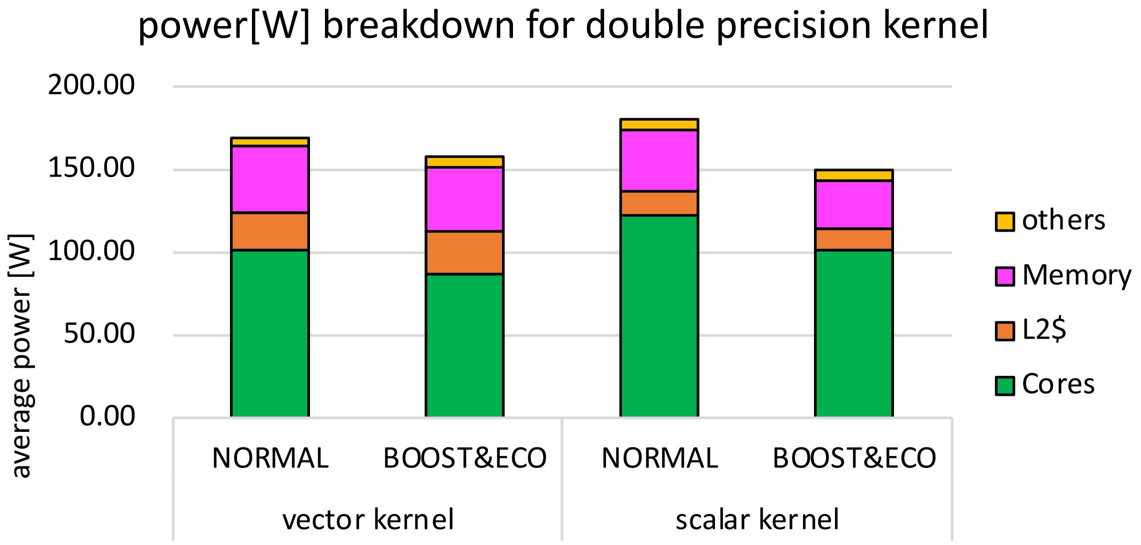




- For this scalar kernel, BOOST&ECO resulted in lower performance than NORMAL. Power consumption stayed on par. NORMAL mode is preferred for this kernel.
- Pipeline busy rate in NORMAL mode is 130% for this scalar kernel.

vectorized/scalar kernel power breakdown

- Average power, i.e. snapshot, of vectorized kernel and scalar kernel.
  - Most power spending component is compute cores (*)
  - The effect of BOOST&ECO mode is also apparent in compute core (*)
  - Scalar kernel spends more power for compute core than vector kernel



- Cores : 48 cores total
- L2$: L2D cache total
- Memory : HBM2 memory total
- others : TofuD and optical modules

(*) Power API does not provide further breakdown such as L1$ or functional pipelines

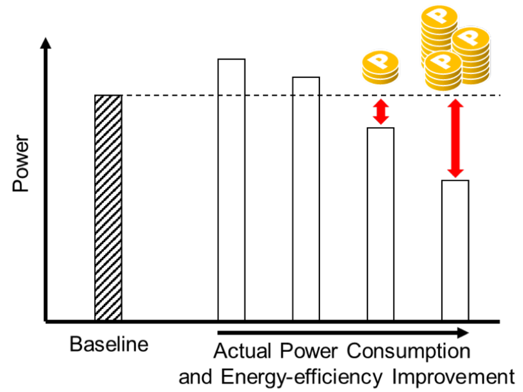
For achieving high performance, energy efficient jobs

- guidelines
  - Understand and optimize the performance and power characteristics of your app
    - NORMAL vs. BOOST&ECO comparison is a good start. Other knobs are optional.
    - Find out your vectorization ratio
  - Aim for wall time reduction through tuning, which will ultimately result in energy saving.
    - Perform source level optimization
      - Take advantage of SVE feature as much as possible
      - reconsider the choice of data precision
    - Resource for source level optimization
      - Try yourself. A wealth of reference material is available on Fugaku website(*)
        - Programming guides : categorized 6 books with many examples
        - Fugaku online seminar materials : tuning part1, part2, part3, and others
        - Case study on A64FX Tuning : example of specific applications
      - Ask for RIST advanced support service

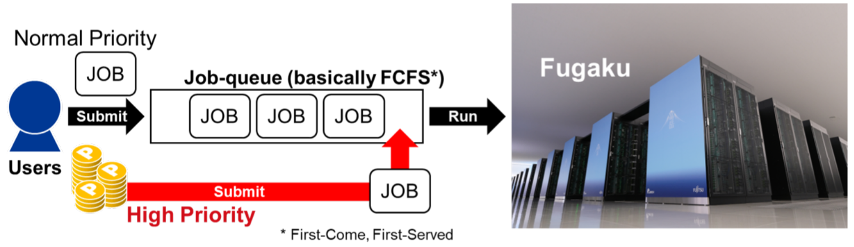
(*) users need Fugaku account to access the material

Fugaku Point Program

- **Fugaku point program has begun to incentivize user cooperation since Apr. 2023.**
- **Points Acquisition:**
  Projects (groups of users) can earn points based on the difference between the baseline power and the actual power consumption.

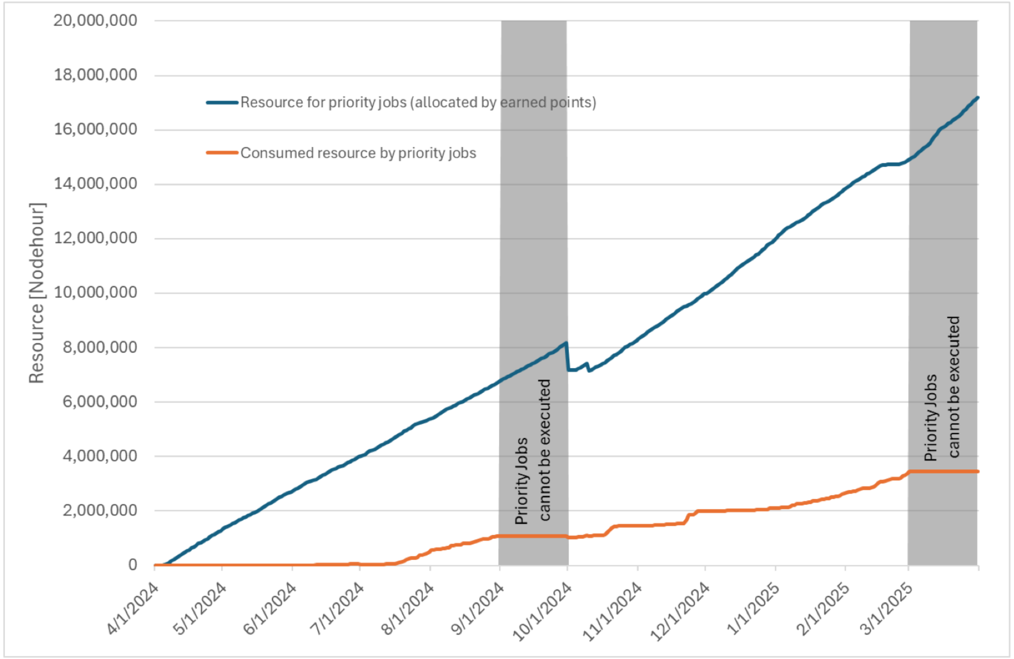


- **Points Redeeming:**
  Jobs can be executed with high priority using the "computational resources for priority execution" redeemed with points.



Fugaku Point Program

● Results of FY2024 (Points Acquisition and Redeeming (all groups))



- The redeeming rate was low.
- We are considering improving Fugaku point program and easier way to consume resource for priority jobs.

Fugaku Point Program

- You can check the status of Fugaku point on the login nodes in real time.
- Example

```
login $ accountj_pt
COLLECTDATE : 2024-02-08 17:18:26  unit[Node Second,Wh]
*--------------------------------[ SUBTHEME ]-----------------------------------------------------------*
SUBTHEME                    PARENT            LIMIT(N)         USAGE(N)         LIMIT(E)         USAGE(E)
group001                   Y23FM01       2,231,676,000    1,160,135,414       67,849,149    5,001,217,071
*--------------------------------[ SUBTHEME_PERIOD ]----------------------------------------------------*
SUBTHEME                         PERIOD       LIMIT(N)         USAGE(N)         LIMIT(E)         USAGE(E)
group001      20230401-20230930       1  1,115,838,000       85,942,585       33,924,574        2,388,876
group001      20231001-20240331       2  1,115,838,000       44,297,414       33,924,574        1,217,071
*--------------------------------[ f-pt RESOURCE GROUP ]----------------------------------------------*
GROUP                  POINT             LIMIT(N)                   USAGE(N)                  AVAILABLE(N)
group001               1,299                  200                        100                           100
*--------------------------------[ SUBTHEME ]-----------------------------------------------------------*
SUBTHEME                    PARENT            LIMIT(N)         USAGE(N)         LIMIT(E)         USAGE(E)
group002                   Y23FM02           3,600,000            9,704          109,450              250
*--------------------------------[ f-pt RESOURCE GROUP ]----------------------------------------------*
GROUP                  POINT             LIMIT(N)                   USAGE(N)                  AVAILABLE(N)
group002                 600                  100                         50                            50
```

For more details, access here.
(Fugaku User Only)

End of slides