スーパーコンピュータ富岳において

高性能で省電力なジョブを実行するための参考資料

May 2025

この文書は The 1st R-CCS/RIST Joint Seminar on Advanced use of Supercomputer Fugaku and Arm computer systems，2025/4/23 の発表資料を元に作成しています。
この文書ではジョブ当たりの電力消費量[Wh] を主なエネルギー指標として用います。

© Copyright 2025, R-CCS

内容

- 富岳のパワーノブ（電力制御機能）の紹介
- 富岳ハードウエアがサポートするデータ精度
- 演算のベクトル処理とその性能効果
- 性能・電力・データ精度に関する比較検討
- 高性能かつ省エネなジョブの実行を目指しての指針
- 省電力ジョブを後押しする「富岳ポイント」サービスの紹介

富岳の主なパワーノブ（電力制御機能）

＊コンビ名でブースト・エコモードと称する

パワーノブ名称	パワーノブ機能	電力制御点	説明・指定可能な値
freq	周波数変更	CPU Socket	CPU内部の周波数を制御。値 (単位:Hz) ・2000000000 (NORMALモード) ・2200000000 (BOOSTモードと呼ぶ)
throttling_state	メモリアクセス制御	メモリ	メモリアクセスコントローラーとメモリ間のバス使用率を制御 0, 1, 2, .. , 9: 0:バス使用率100%（最高性能）, 1: 90%, 2: 80%, .. , 9: 10%
issue_state	命令発行制限	CPUコア	CPUコアが同時に処理する命令数を制御 0: 4命令 1: 2命令
ex_pipe_state	EXA only	CPUコア	整数演算パイプライン数を制御 0: パイプA、パイプBを利用 1: パイプAのみ利用
eco_state	エコモードとFLA only組み合わせ	CPUコア	FLA onlyは浮動小数点演算パイプライン数を半減し、エコモードはさらに電力削減効果を増やす機能。 0: エコモード無効、FLA only無効 1: エコモード無効、FLA only有効 2: エコモード有効、FLA only有効（単にエコモードと称する事が多い）
retention_state	Retention制御	CPUコア	アイドル時の低電力状態（Retention状態）への遷移を制御。 0: Retention状態に遷移しない 1: Retention状態に遷移する

富岳のパワーノブ（電力制御機能）

- パワーノブの適用方法
  - A) ジョブ投入時の指示行で指定(*1)：ジョブ毎（ノード内一律）に適用
  - B) アプリの中でPMlibを利用(*2)：電力制御点毎に適用可能、Fortran/C/C++/Pythonに対応
  - C) アプリの中でPowerAPIを利用(*3)：電力制御点毎に適用可能、C言語に対応

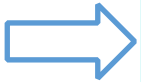
- 消費電力の確認方法
  - A) ジョブ管理ソフトの出力(statsファイル）：ジョブ毎の消費電力
  - B) アプリの中でPMlibを利用：指定区間毎の消費電力。Fortran/C/C++/Pythonに対応
  - C) アプリの中でPowerAPIを利用：指定区間毎の消費電力。C言語に対応。
  - D) fappでアプリを11回以上実行、fapppxとCPU性能解析レポートで表示(*4)：指定区間毎の消費電力。Fortran/C/C++に対応。

(*1) 利用手引書 利用およびジョブ実行編 7.2. パワーモード指定
(*2) オープンソース PMlib tutorial
(*3) 利用手引書 利用およびジョブ実行編 7.3 Power API
(*4) マニュアル プロファイラ使用手引書 3. 詳細プロファイラ, 4. CPU 性能解析レポート

パワーノブの適用と結果の確認方法

- 例A)：ジョブ投入時の指示行で指定
  - 最も簡便に指定可能。利用手引書 利用およびジョブ実行編 7.2. パワーモード指定
  - ジョブ全体を通しての結果であることに注意する

```
# ジョブスクリプト指示行での適用例
#PJM --rsc-list "freq=2000"
#PJM --rsc-list "eco_state=2"
#PJM --rsc-list "retention_state=0”
#PJM -s
```



```
*.statsファイル中に表示：132,133行目
AVG POWER CONSUMPTION OF NODE (IDEAL) : 142.9     (消費電力： W)
ENERGY CONSUMPTION OF NODE (IDEAL) : 4.2796       (消費電力量： Wh)
```

- 2025年４月から、省略時には以下の設定（通称ブースト・エコモード）が採用されている

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

- 従来の標準設定（通称ノーマルモード）でジョブを実行するには以下を指定する

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

パワーノブの適用と結果の確認方法

- 例B) ：アプリの中でPMlibを利用
  - Fortran/C/C++/Pythonに対応するオープンソースライブラリ。PMlibリポジトリと、チュートリアル
  - より詳細なアプリ内指定区間毎の消費電力と計算性能情報を同時にレポート
  - 富岳の上ではコンパイル済みのspackモジュールとして利用可能

```
!cx Fortranプログラムでの呼び出し例
call f_pm_initialize (1)
call f_pm_start (”label")
 .. 測定したい計算部分 ..
call f_pm_stop (”label")
call f_pm_report ("")
```

```
spack load pmlib@10.0-clang
export PMLIB_REPORT=BASIC
export POWER_CHOOSER=NODE
./a.out
```

標準出力ファイル中にレポート表示：

```
    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
-----------+--------+-------------------------+-----------
```

パワーノブの適用と結果の確認方法

- 例C)：アプリの中でPower APIを利用（抜粋Cソース・結果）
  - 利用手引書 利用およびジョブ実行編 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]
```

パワーノブ結果の確認方法

- 例4：FAPPを17回実施し、CPU性能解析レポートに出力した例
  - プロファイラ使用手引書 3. 詳細プロファイラ，4. CPU 性能解析レ
  - 4-1 計算ノードでfappを反復実行

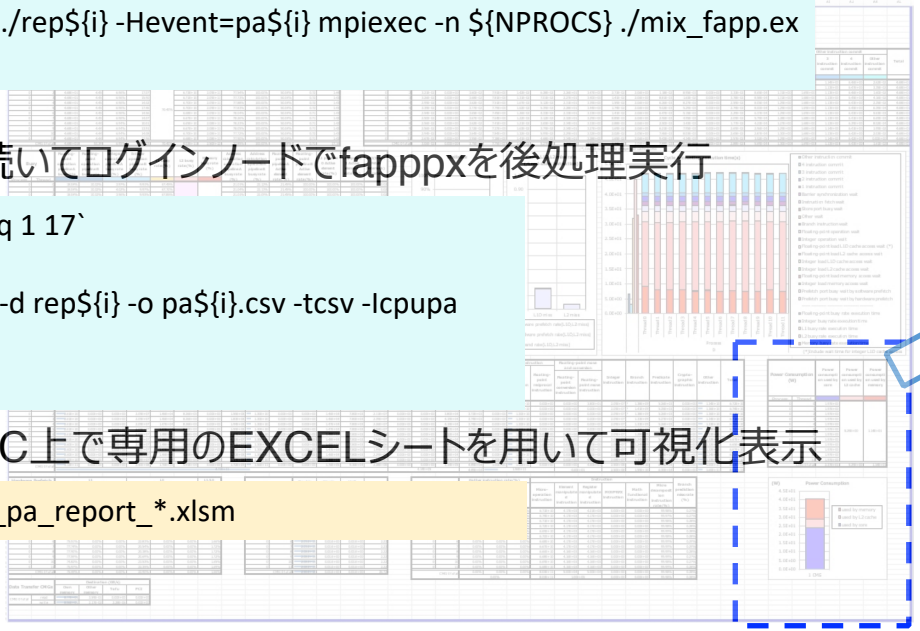
```
for i in `seq 1 17`
do
fapp -C -d ./rep${i} -Hevent=pa${i} mpiexec -n ${NPROCS} ./mix_fapp.ex
done
```

  - 4-2 続いてログインノードでfapppxを後処理実行

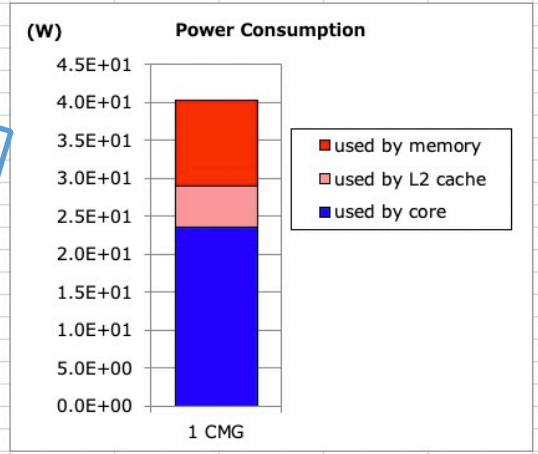
```
for i in `seq 1 17`
do
fapppx -A -d rep${i} -o pa${i}.csv -tcsv -Icpupa
done
pwd
```

  - 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
```

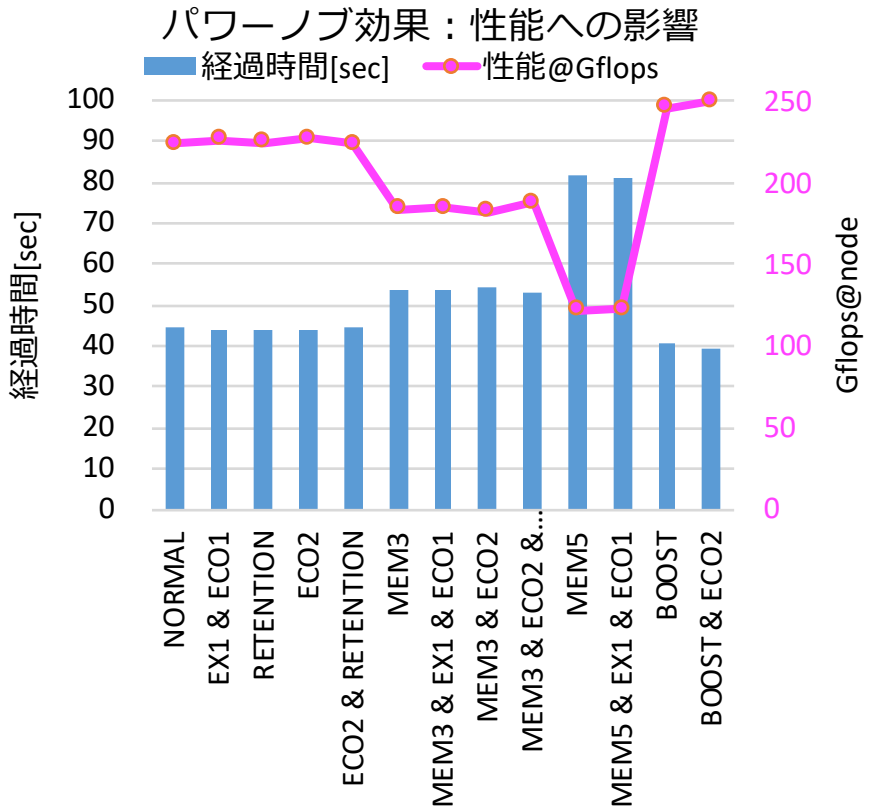


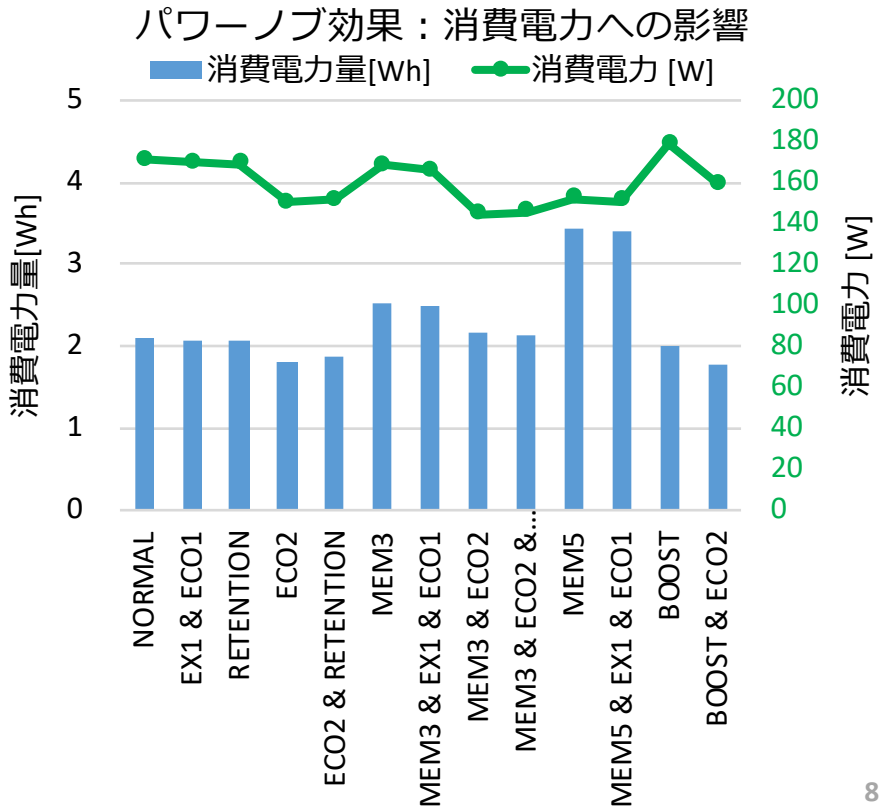
Power Consumption (W)		Power consumption used by core	Power consumption used by L2 cache	Power consumption used by memory
Process	Thread			
0	0	1.97E+00		
0	1	1.97E+00		
0	2	1.97E+00		
0	3	1.97E+00		
0	4	1.97E+00		
0	5	1.97E+00	5.29E+00	1.14E+01
0	6	1.97E+00		
0	7	1.97E+00		
0	8	1.97E+00		
0	9	1.97E+00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



性能・電力：パワーノブ各種の効果測定例

- ベクトル化率の高い単純カーネルでの測定例



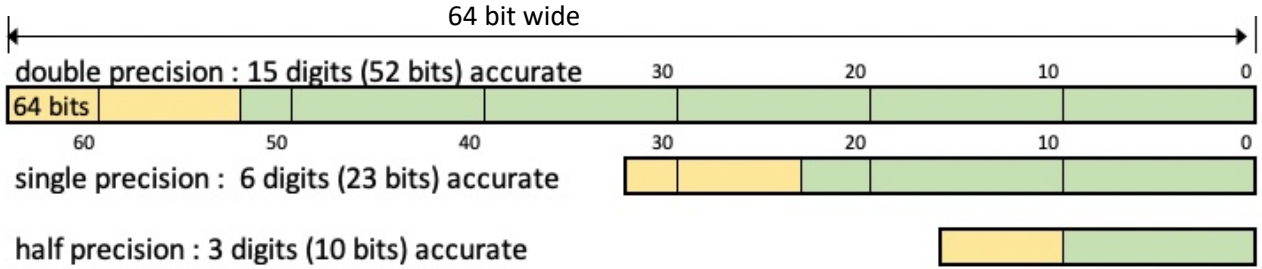


富岳がハードウエアでサポートするデータ精度

- データ型の選択は演算精度・演算性能・必要メモリサイズ・消費電力に大きく影響する
  - ハードウエア命令がサポートする浮動小数点データ型

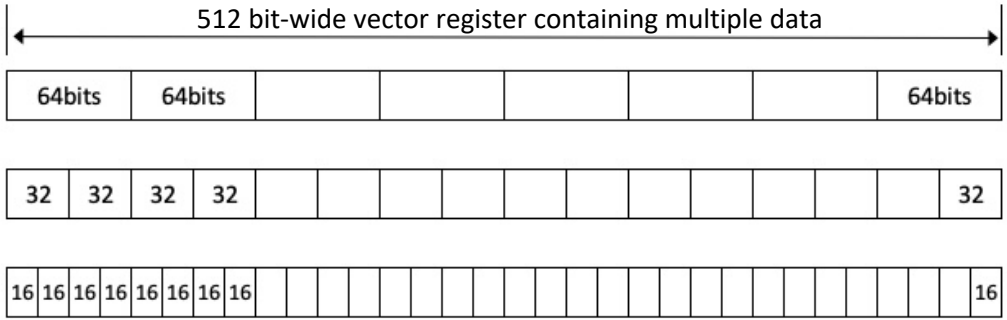
data type	double precision	single precision	half precision
bits wide	64	32	16
accuracy	15 digits	6 digits	3 digits
fortran	real(kind=8)	real(kind=4)	real(kind=2)
C/C++	double	float	_Float16
Python	float64	float32	float16

- 整数データ型も同様に
  64 / 32 / 16 / 8 ビット整数型
- ソフトウエア言語がサポートする実数・整数演算はこれらのデータ型にマップされる



演算のベクトル処理とその性能効果

- 富岳CPUがサポートする 512 ビット幅ベクトル演算 – SVE



倍精度浮動小数点演算　：8要素同時実行(*1)

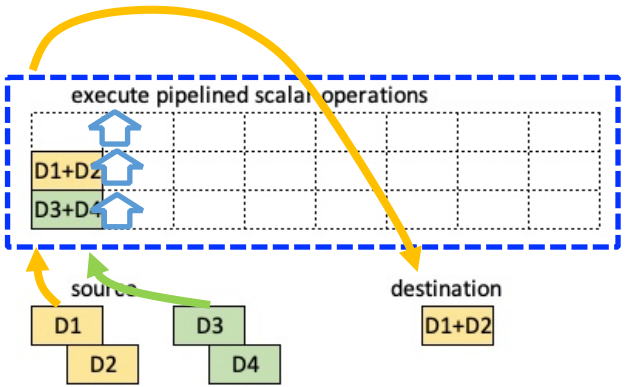
(*1) 積和演算の場合は16要素同時実行

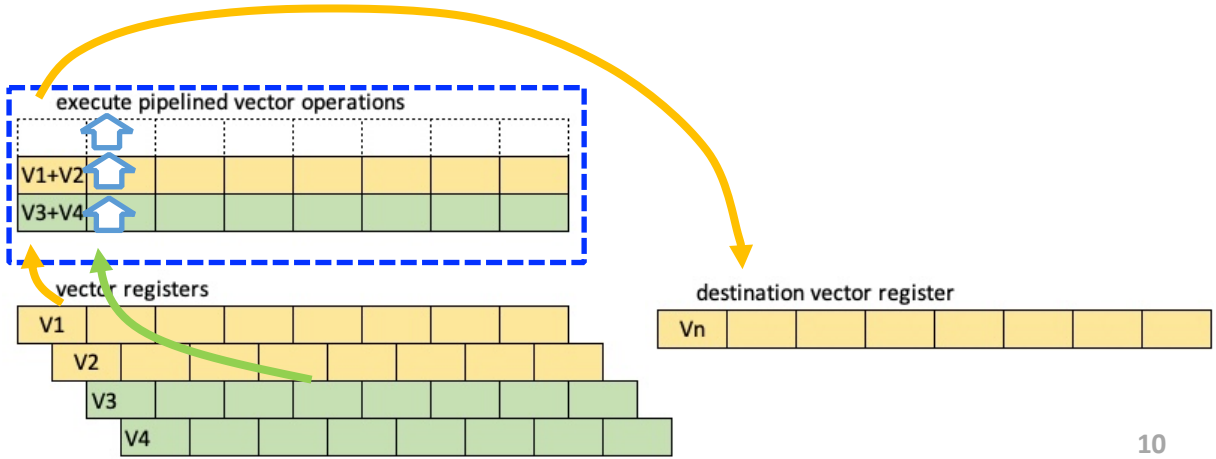
単精度浮動小数点演算　：16要素同時実行(*2)

(*2)積和演算の場合は32要素同時実行

半精度浮動小数点演算：32要素同時実行(*3)

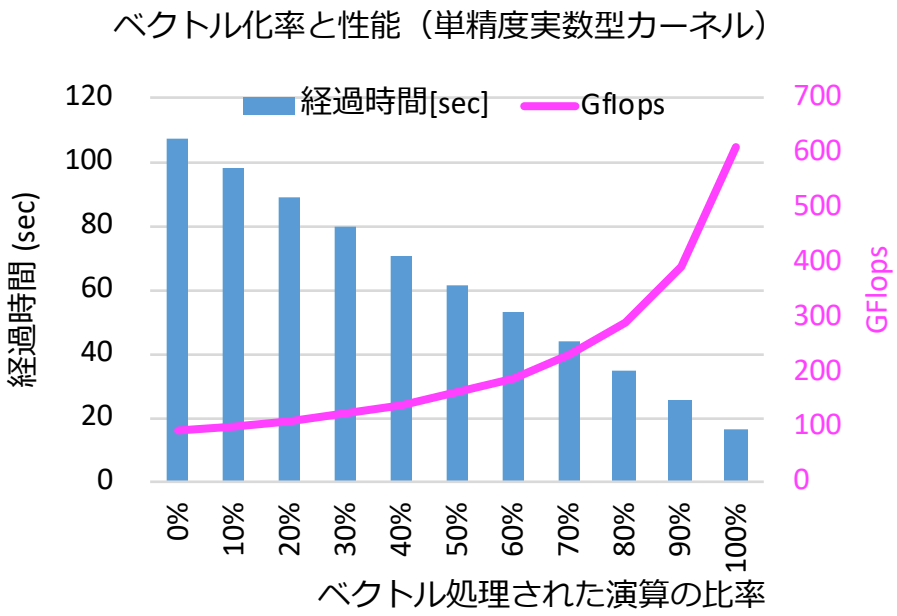
(*3)積和演算の場合は64要素同時実行

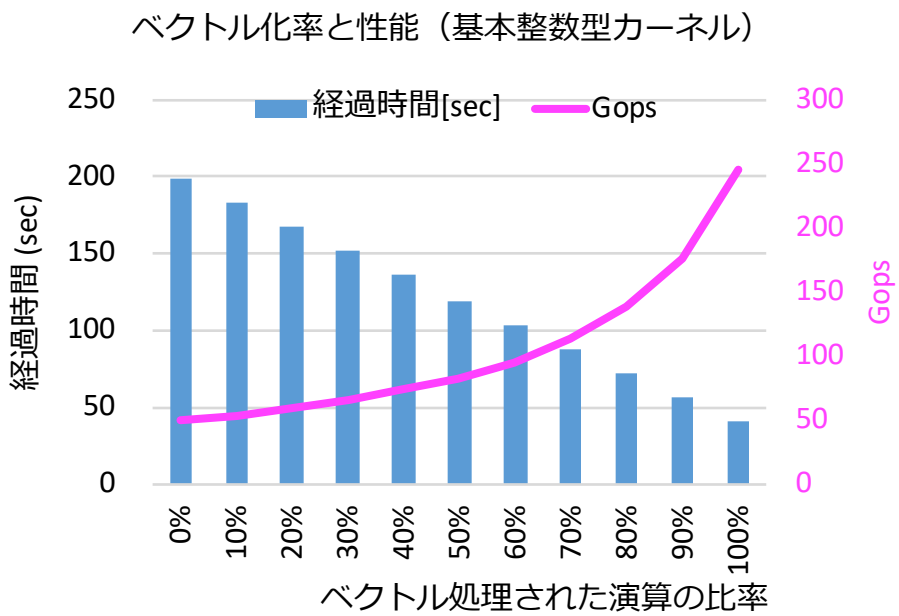




演算のベクトル処理とその性能効果

- ベクトル処理による性能向上の効果例（実数演算・整数演算）
  - アプリケーションを高性能で実行するためにはSVEによるベクトル化処理を用いることが重要
  - 高性能計算を実現することは結果的にジョブ当たりの消費電力低減にも寄与する（後出）





性能・電力・データ精度に関する比較検討

- 以下の検討が容易にできるアプリカーネルを一点ピックアップして、各側面から比較
  - パラメタ
    - 浮動小数点データの精度（倍精度・単精度・半精度）
    - SVEの効果(ベクトル化率100%のカーネル・0%のカーネル）
    - パワーノブの効果（主にノーマルモード対ブースト・エコモード比較）
  - 評価尺度
    - 計算時間、計算性能
    - ノード消費電力量(Wh)、ノード消費電力(W)
  - 設定する並列モデル
    - ノードあたり 4 process x 12 threads
  - 各種比較は1ノードあたりの値を基本とする

性能・電力・データ精度に関する比較検討

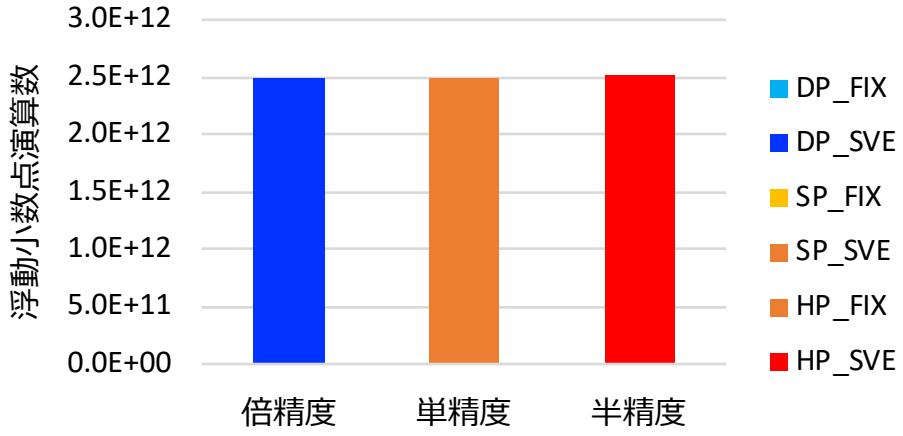
- アプリカーネル(1)：ソースプログラム抜粋

```
!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
```

データ精度とカーネル性能の関係（ベクトル化カーネルの場合）

- アプリカーネル(1)@コンパイラfastオプション
  - コンパイラが生成したSVE融合積和命令が適切に実行された
  - データ型を倍精度・単精度・半精度実数とした場合の実効性能を比較

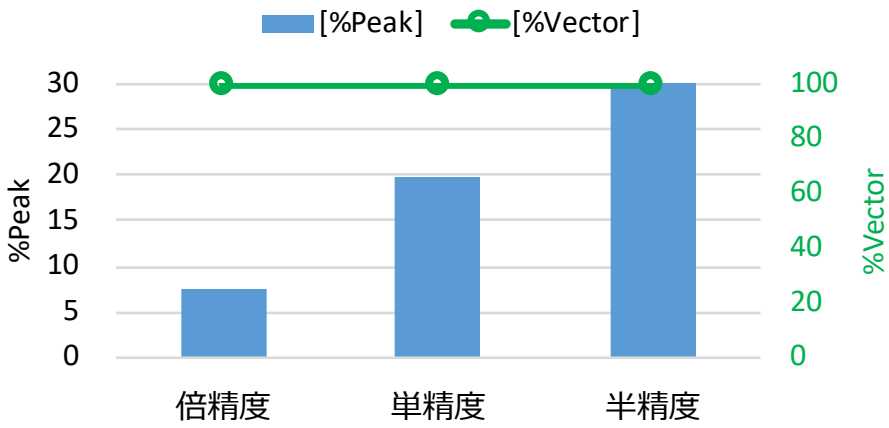
演算命令のタイプ（SVE/FIX）



DP_FIX：倍精度スカラー演算,
SP_FIX：単精度スカラー演算,
HP_FIX：半精度スカラー演算,

DP_SVE：倍精度SVE演算
SP_SVE：単精度SVE演算
HP_SVE：半精度SVE演算

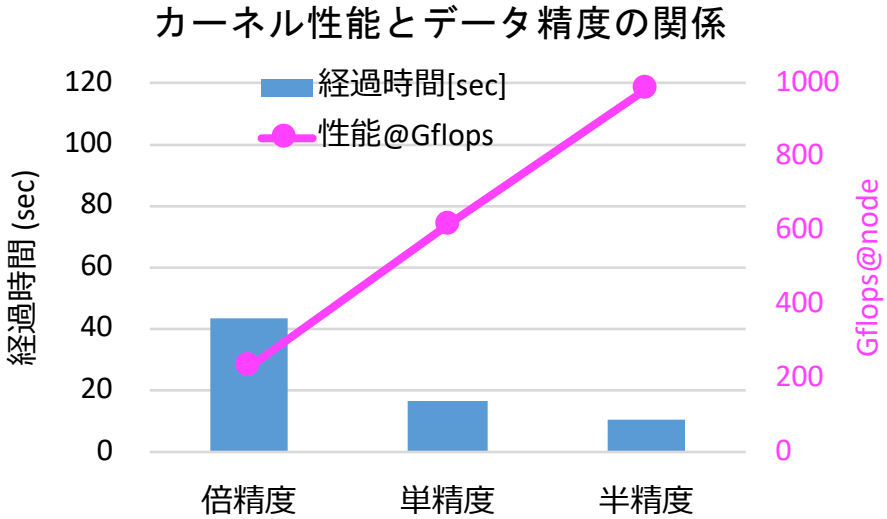
ピーク性能比とベクトル化率

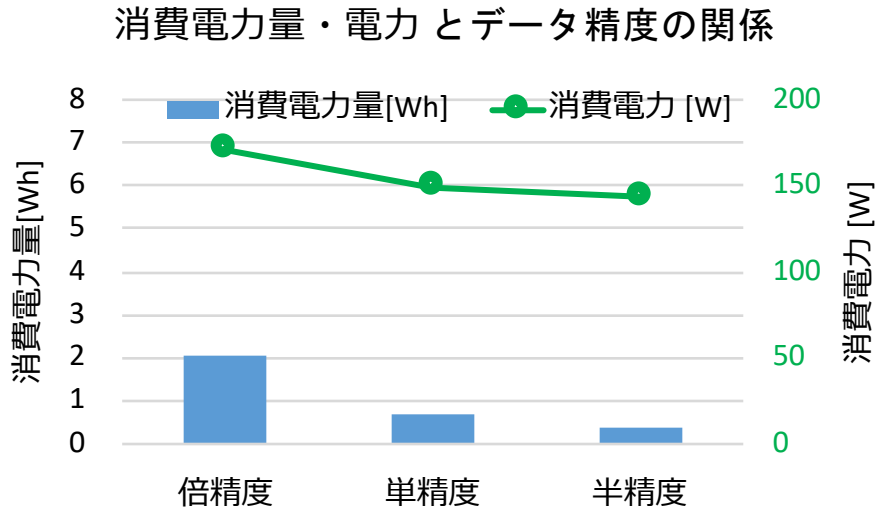


ピーク性能比：ノーマルモードでの倍精度・理論ピーク性能値に対する実効性能比

データ精度とカーネル性能の関係（ベクトル化カーネルの場合）

- アプリカーネル(1) @コンパイラfastオプション
  - 演算精度（データ型・演算とも）を下げた場合の性能向上(flops)・省電力(Wh)の効果が大きい



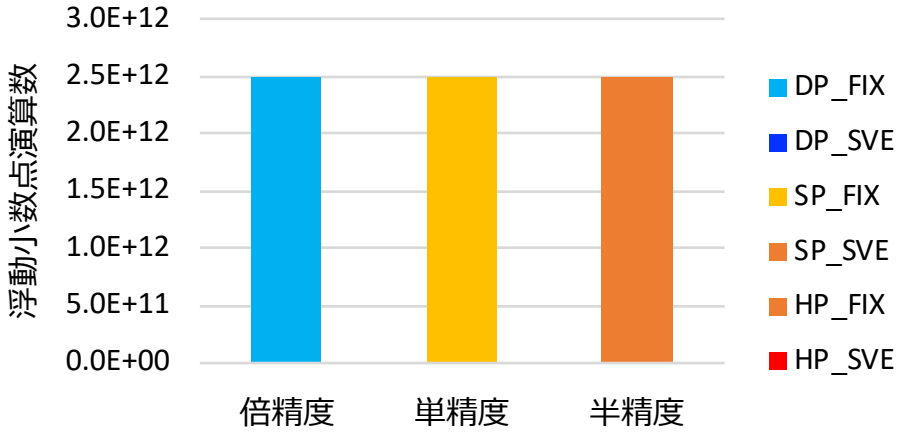


単精度演算・半精度演算の適用が可能ならば、ベクトル化されたカーネルでは、
より高性能でかつ消費電力量[Wh]を低減したジョブの実行が可能となる

データ精度とカーネル性能の関係（スカラーカーネルの場合）

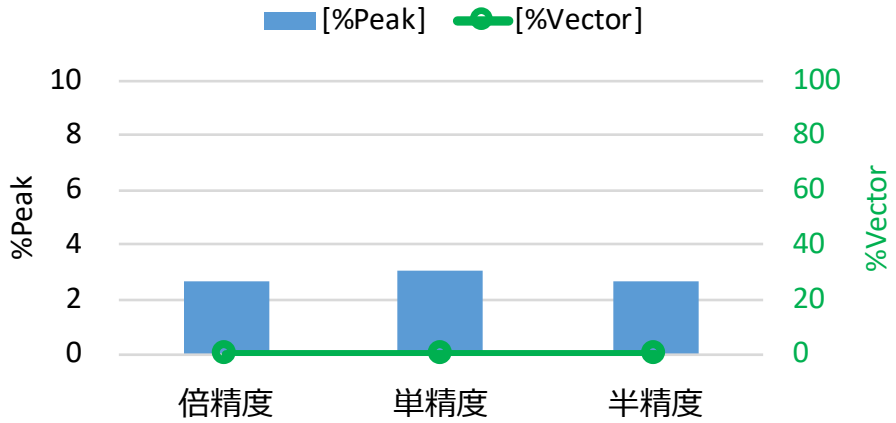
- アプリカーネル(1)@コンパイラnosimdオプション
  - SVE命令生成を抑止した場合
  - 浮動小数点データ型を倍精度・単精度・半精度とした場合の結果を比較

演算命令のタイプ（SVE/FIX）



DP_FIX：倍精度スカラー演算,
SP_FIX：単精度スカラー演算,
HP_FIX：半精度スカラー演算,
DP_SVE：倍精度SVE演算
SP_SVE：単精度SVE演算
HP_SVE：半精度SVE演算

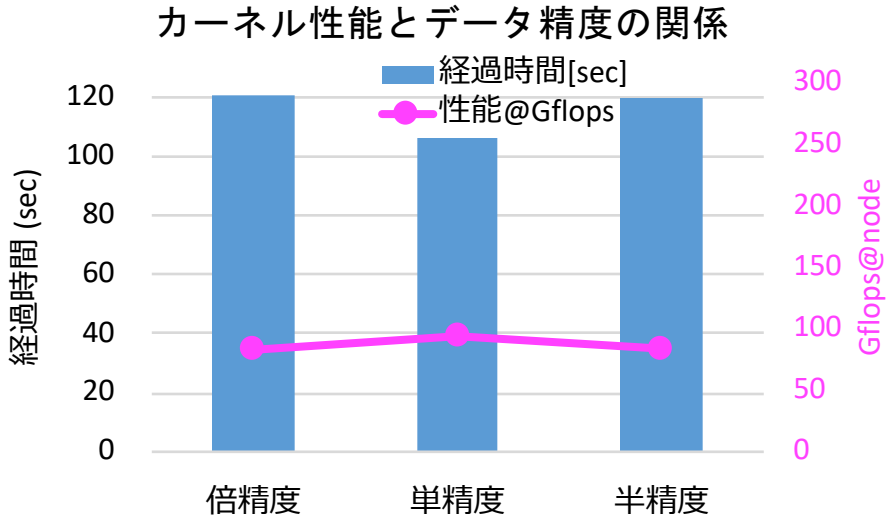
ピーク性能比とベクトル化率

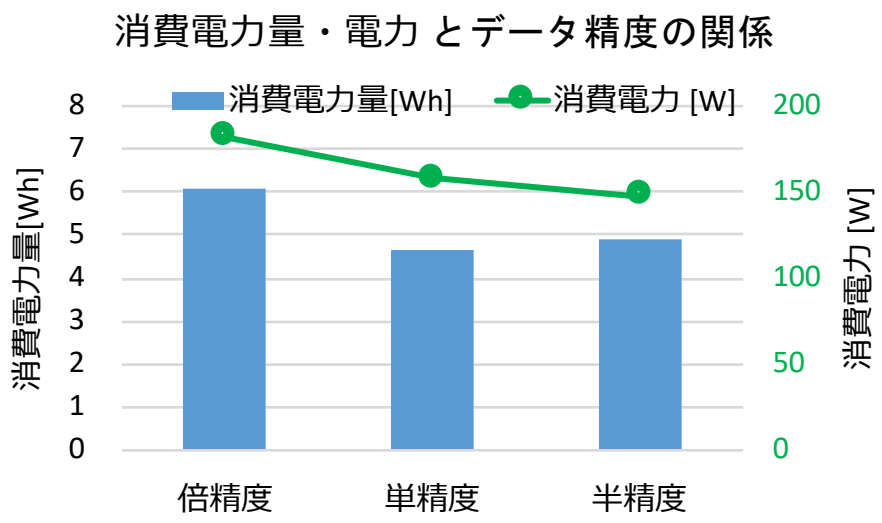


ピーク性能比：ノーマルモードでの理論ピーク性能値に対する実効性能比

データ精度とカーネル性能の関係（スカラーカーネルの場合）

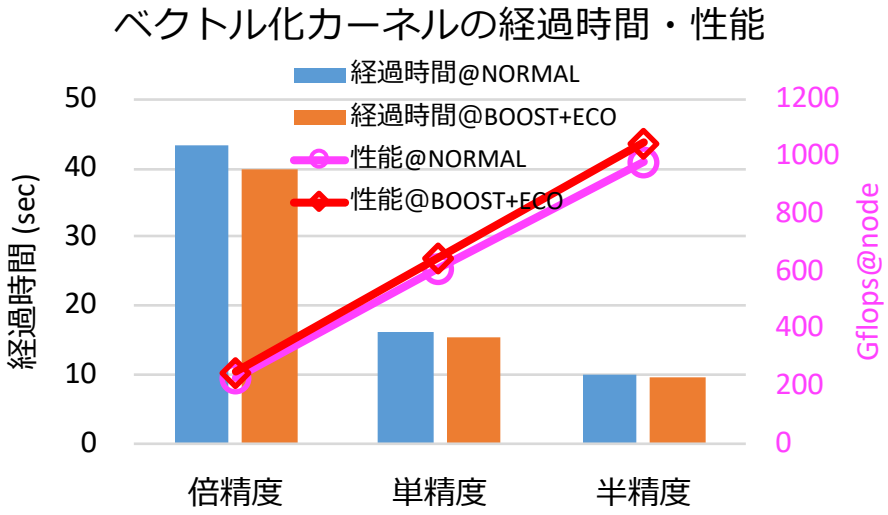
- アプリカーネル(1) @コンパイラnosimdオプション
  - 演算精度を下げる効果はベクトルカーネルほど大きくない
  - 省電力効果は倍精度から単精度への変更で20%程度
  - 半精度演算の性能が単精度演算よりも悪化したのは想定外（要調査）

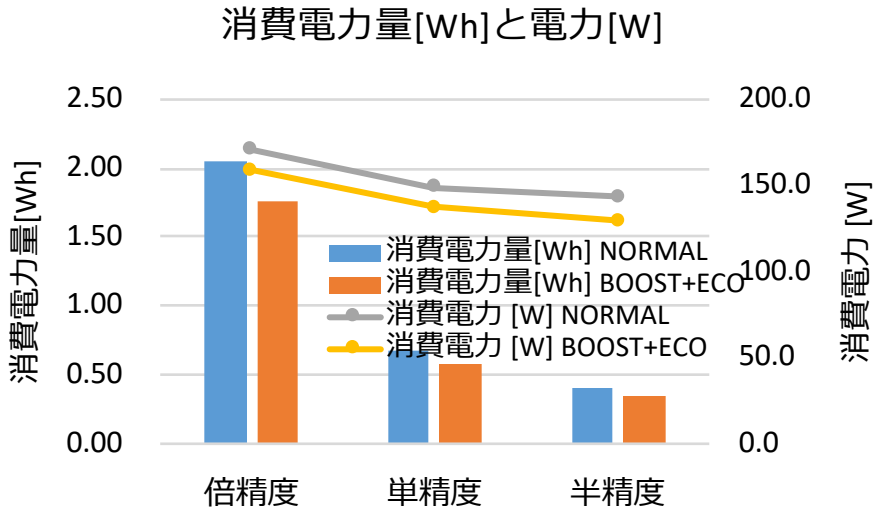




性能・電力と動作モードの関係：ノーマルモード対ブーストエコモード

- ベクトル化カーネルの場合
  - アプリカーネル(1)@コンパイラfastオプション

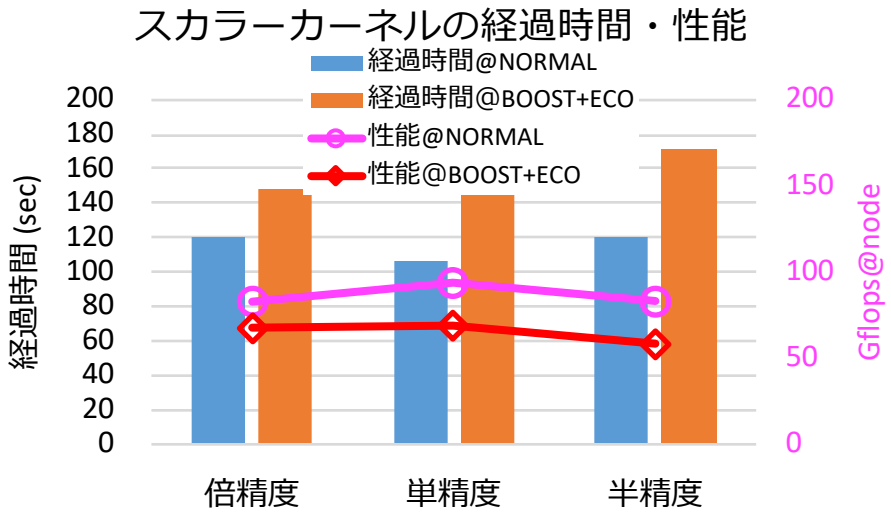


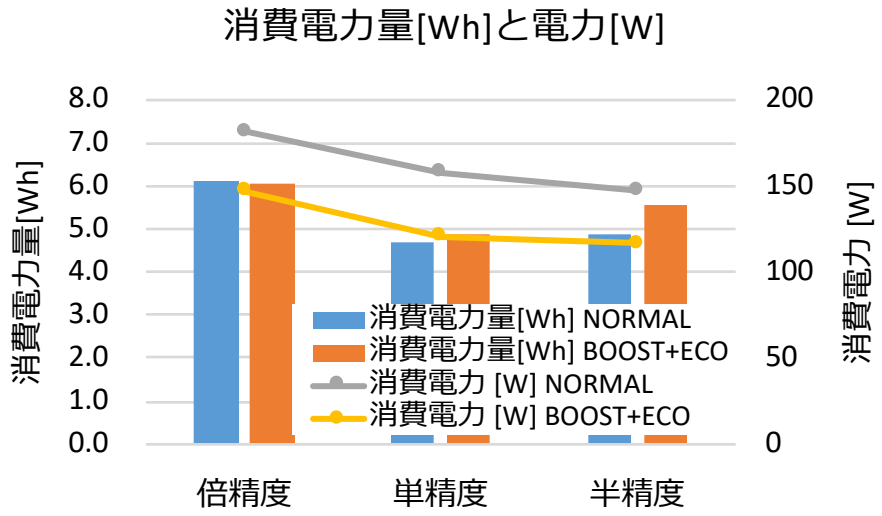


- ベクトク化カーネルではブーストエコモードがノーマルモードよりも高い性能を示し、かつ消費電力量[Wh]の低減をきたす
- この傾向は他のベクトル化されたカーネル一般にもあてはまる。例外は演算パイプラインへの負荷が100%を超える場合

性能・電力と動作モードの関係：ノーマルモード対ブーストエコモード

- スカラーカーネルの場合
  - アプリカーネル(1)@コンパイラnosimdオプション



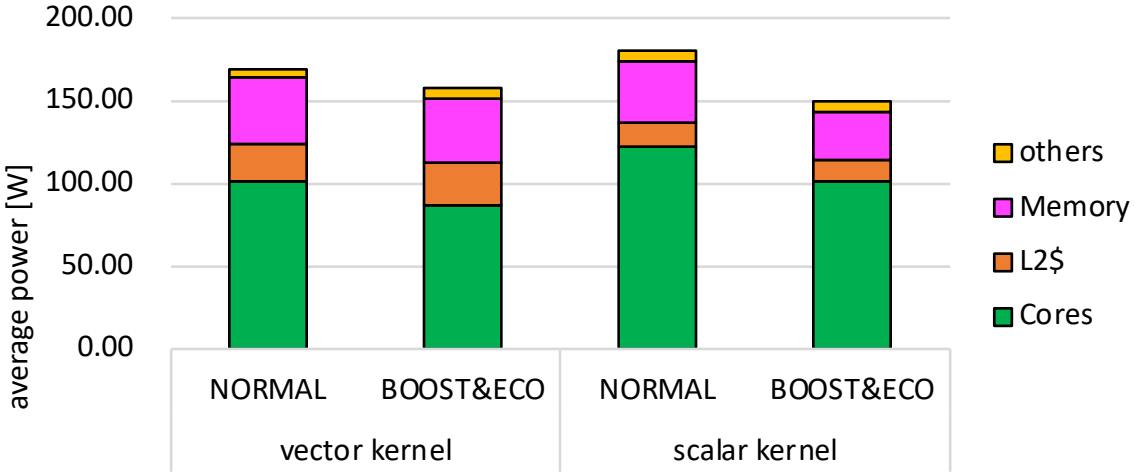


- スカラーカーネルではブーストエコで性能が低下した。消費電力量(Wh)は両モード間であまり変わらない
- スカラーカーネルの演算命令スケジューラの効果により、パイプラインビジー率が130%であるため

ベクトル化・スカラーカーネルの電力内訳

- ベクトル化・スカラーカーネルの平均電力（W）
  - 最も電力を消費するパーツは計算コア (*)
  - ブーストエコモードの効果がよく現れるのも計算コアの部分 (*)
  - スカラーカーネルがベクトルカーネルよりも多く電力を消費するパーツも計算コア

power[W] breakdown for double precision 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

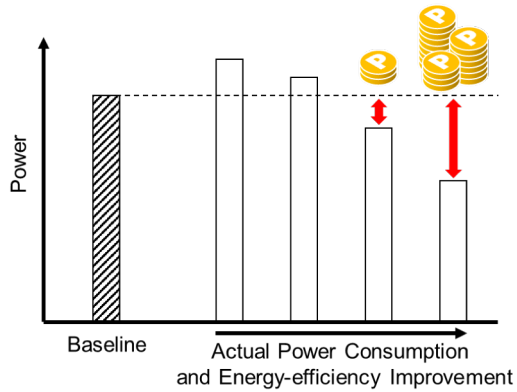
高性能かつ省エネなジョブ実行を目指しての指針

- ガイドライン
  - 自分のアプリの性能特性・電力特性を把握して最適化を進める。
    - まずノーマルモードとブーストエコモードでの比較を行うことを推奨。パワーノブ最適化はオプション。
    - 自分のアプリのベクトル化の状況を確認する

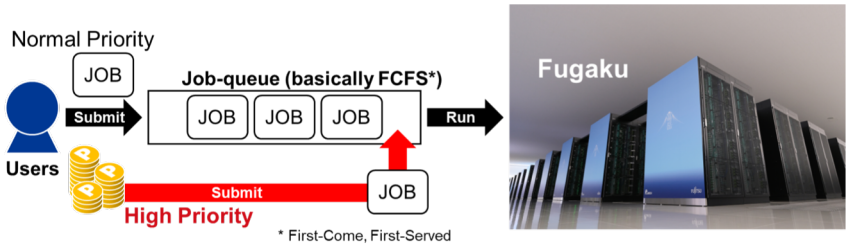
  - 計算時間を短縮を目指すことが結果的に省エネなジョブの実行へつながる
    - ソースコードレベルで富岳に適した最適化の実施が望ましい
      - SVE処理機能を最大限に活用する
      - 適切なデータ型の採用を再検討する
    - 最適化に向けたサポート資源
      - 自力で頑張る場合、参考情報が富岳ウエブサイトに豊富に掲載されている
        - プログラミングガイド　：言語・目的別に整理されたガイド６冊
        - 「富岳」利用セミナー資料：中級編　Part1、Part2、Part3、ハンズオン、他
        - A64FXチューニング事例：個別アプリケーションのチューニング事例
      - RIST高度化支援サービスを利用する

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.

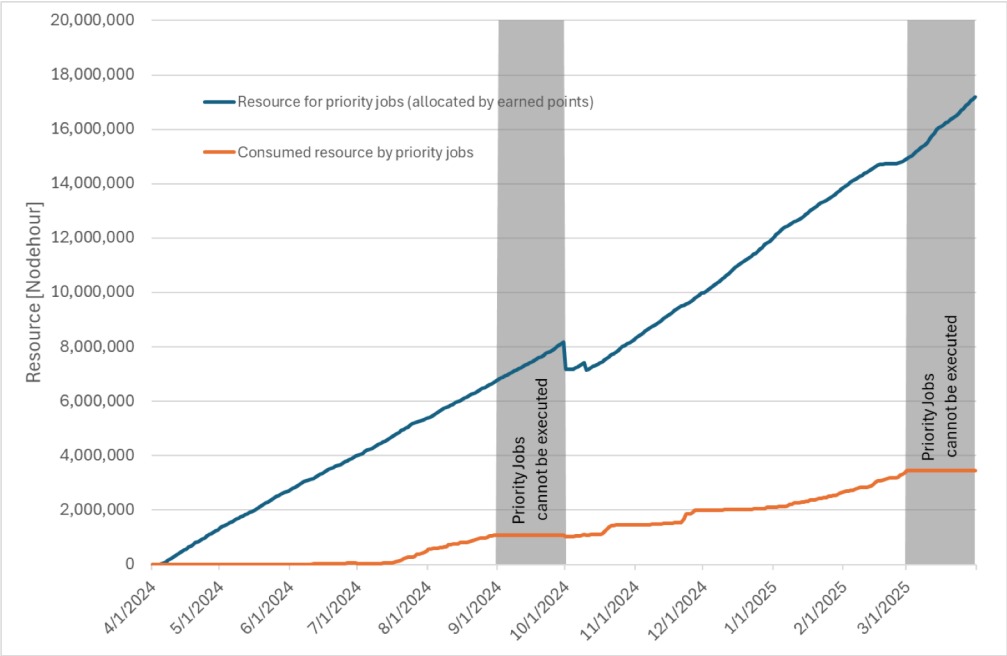


- **Points Redeeming:**
  Jobs can be executed with high priority using the "computational resources for priority execution" redeemed with points.



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)

以上です