Introducing PMlib

Nov. 2024 Kazunori Mikami

What is PMlib?

- Class Library for monitoring the performance of user application
  - Measure and report the computing performance of code sections
  - Internal interface with HWPC(hardware performance counters)
  - Internal interface with power API
  - Statistics per job, per process, per thread
  - Available for C++ / C / Fortran / Python programs
  - Tested platforms – Linux variants
    - Supercomputer Fugaku, FX servers with a64fx chip
    - Intel Xeon servers with Skylake and other x86_64 chips
  - Open source with GPL license
    - PMlib development https://github.com/mikami3heart/PMlib
    - PMlib Python https://github.com/mikami3heart/PMlib-pybind
    - Official download site http://avr-aics-riken.github.io/PMlib/

What is good about PMlib?

- HPC oriented measurements and reports
  - Detect HWPCs and obtain statistics for the target sections
    - appropriate set of events are automatically measured by choosing : FLOPS, VECTOR, BANDWIDTH, etc.
  - Flops stats per data type precision
    - half precision(16) / single precision(32) / double precision(64)
  - Evaluation metric includes
    - sustained performance per peak performance ratio
    - vectorization ratio of f.p. instructions
  - integrated power consumption measurement through power API
    - appropriate set of power components are automatically measured by choosing : NODE, NUMA, PARTS
  - Reporting level of details is controlled by run time environment variables
- Internal precise timer is used
- Python API is available at last
- Plain text output

How to use PMlib

- Install PMlib at your preference
  - On Fugaku, PMlib is already available as spack module.
    - spack load pmlib@9.2-clang # fortran and clang mode C++/C
    - spack load pmlib@9.2-trad # trad mode C++/C
    - spack load pmlib@10.0-pybind # python binding
  - On other systems, install PMlib at your preferred location, which is usually quite easy.

- Insert PMlib API calls in your source code
  - Typically, users would want to call only 4 basic APIs, i.e. initialize/start/stop/report
- Compile and build the application
- Run the application with your favorite reporting format. The format is controlled through environment variables. The default values are usually good enough.

PMlib basic API

- PMlib basic APIs – most users may need basic APIs only

function	C++ API	C API	APIFortran	Python API	calling location	arguments
Setup PMlib	initialize	C_pm_initialize	f_pm_initialize	initialize	after MPI_Init(), if any	none [*1]
mark start of the section	start	C_pm_start	f_pm_start	start	anywhere. start and stop must be in pair	string(section label)
mark end of the section	stop	C_pm_stop	f_pm_stop	stop	anywhere. start and stop must be in pair	string(section label) [*1]
produce statistics report	report	C_pm_report	f_pm_report	report	before MPI_Finalize(), if any	file name or pointer [*2]

[*1] Optional arguments can be given
[*2] argument in report()
in C++ API : "" (NULL) for stdout. otherwise FILE *
in C API: "" (NULL) for stdout. otherwise char *filename
in fortran API: "" (NULL) for stdout. otherwise character*(*) filename
in Python API: "" (NULL) for stdout. otherwise string filename

How to code PMlib enabled application

- Fortran code before

```
program main


call mykernel()


end
```

- Fortran code after

```
program main
call f_pm_initialize (1)
call f_pm_start (”label")
call mykernel ()
call f_pm_stop (”label")
call f_pm_report ("")
end
```

initialize

measuring section

produce report

- C++ code before

```
int main(int argc, char *argv[])
{



mykernel();


return 0;
}
```

- C++ code after

```
#include <PerfMonitor.h>
using namespace pm_lib;
PerfMonitor PM;
int main(int argc, char *argv[])
{
PM.initialize();
PM.start(”label");
mykernel();
PM.stop (”label”);
PM.report(stdout);
return 0;
}
```

header file, etc

initialize

measuring section

produce report

How to code PMlib enabled Python application

- Python serial code before

```
mykernel()
```

- Python serial code after

```
import pyPerfMonitor
pmlib = pyPerfMonitor.PerfMonitor()
pmlib.initialize(1)
pmlib.start("label")
mykernel()
pmlib.stop("label")
pmlib.report("")
```

loading

initialize

measuring section

produce report

- Python mpi4py code before

```
from mpi4py import MPI

mykernel()
```

- Python mpi4py code after

```
from mpi4py import MPI
import pyMpiPerfMonitor
pmlib = pyMpiPerfMonitor.PerfMonitor()
pmlib.initialize(1)
pmlib.start("label")
mykernel()
pmlib.stop("label")
pmlib.report("")
```

loading

initialize

measuring section

produce report

- Note that imported module names are different

How to build and run PMlib enabled apps on Fugaku

```
#PJM -N PMLIB-FORT-THREADS-4x12
#PJM --rsc-list "rscunit=rscunit_ft01,rscgrp=small,elapse=00:10:00,node=1"
#PJM --mpi "max-proc-per-node=4"
#PJM -j
source /vol0004/apps/oss/spack/share/spack/setup-env.sh
spack load pmlib@9.2-clang
export LD_LIBRARY_PATH=/lib64:${LD_LIBRARY_PATH}  # to avoid "xos LPG 2002" warning
FFLAGS="-Kopenmp -w -Cpp -Nlst=t "
LDFLAGS="-lPMmpi -lpapi_ext -lpower_ext -lpapi -lpfm -lpwr "     # for PMlib MPI version
#    LDFLAGS="-lPM -lpapi_ext -lpower_ext -lpapi -lpfm -lpwr "    # for PMlib serial version
LDFLAGS+="-lstdc++ "              # for fortran linker

WKDIR=${HOME}/tmp/check_thread_pattern
mkdir -p $WKDIR; cd $WKDIR; if [ $? != 0 ] ; then echo 'Directory error'; exit; fi
cp ${HOME}/pmlib/src_tests/src_parallel/parallel_mix.f90 main.f90
mpifrt ${FFLAGS} main.f90 ${LDFLAGS}


xospastop
export OMP_NUM_THREADS=12
export PMLIB_REPORT=FULL
export HWPC_CHOOSER=FLOPS
export POWER_CHOOSER=PARTS
mpiexec -np 4  ./a.out
more  output.${PJM_JOBID}/0/*/std*
```

- A job to build and run a fortran app using PMlib
- More examples are available in ${PMLIB_DIR}/doc/scripts

PMlib reports

- The report format is controlled by the environment variables
- PMLIB_REPORT: reporting level of details
  - choose from BASIC, DETAIL, FULL, OFF
- HWPC_CHOOSER: type of performance metric
  - choose from BANDWIDTH, FLOPS, VECTOR, CACHE, CYCLE, LOADSTORE, or USER
- POWER_CHOOSER: summary of power consumption
  - choose from NODE, NUMA, PARTS, OFF

PMlib report example

- tested code – stream benchmark written in C++

```
PM.start("stream_check");

    for (k=0; k<NTIMES; k++)
    {
    PM.start("sub1_copy");
#pragma omp parallel for
    for (j=0; j<N; j++)
        c[j] = a[j];
    PM.stop ("sub1_copy");

    PM.start("sub2_scale");
#pragma omp parallel for
    for (j=0; j<N; j++)
        b[j] = scalar*c[j];
    PM.stop ("sub2_scale");
    PM.start("sub3_add");
#pragma omp parallel for
    for (j=0; j<N; j++)
        c[j] = a[j]+b[j];
    PM.stop ("sub3_add");

    PM.start("sub4_triad");
#pragma omp parallel for
    for (j=0; j<N; j++)
        a[j] = b[j]+scalar*c[j];
    PM.stop ("sub4_triad");
    }

PM.stop ("stream_check");
```

PMlib report examples

Report for PMLIB_REPORT=BASIC option page1

```
# PMlib Basic Report ------------------------------------------------------------ #

    Performance Statistics Report from PMlib version 9.2.0
    Linked PMlib supports: MPI, OpenMP, HWPC, PowerAPI, no-OTF on this system
    Host name : a01-4008c
    Date      : 2024/11/07 : 13:00:27

    Parallel Mode:   Hybrid (4 processes x 12 threads)
    The following cotroll variables are provided to PMlib as environment variable.
        HWPC_CHOOSER=BANDWIDTH
        POWER_CHOOSER=NODE
        PMLIB_REPORT=BASIC
    Active PMlib elapsed time (from initialize to report/print) = 3.774e+02 [sec]
    Basic process stats as the average of all the processes are reported below.
    See Legend page if the section name is annotated with special symbols such as (*),(+).

Section          | number of| measured | weight| time per| std.dv of | hardware counted data access
Label            |   calls  | time[sec]   [%]   call[sec]    time    |   Bytes    std.dv  Mem+LLC bandwidth
-----------------+----------+-----------------------------------------+-------------------------------
stream_check (*) :       30   3.769e+02  99.85  1.256e+01  5.65e-01    6.728e+13  1.35e+10 178.52 GB/sec(*)
sub3_add         :    30000   9.746e+01  25.82  3.249e-03  2.38e-01    1.924e+13  3.19e+09 197.46 GB/sec
sub4_triad       :    30000   9.723e+01  25.76  3.241e-03  2.17e-01    1.921e+13  3.01e+09 197.60 GB/sec
sub2_scale       :    30000   7.519e+01  19.92  2.506e-03  1.41e-01    1.441e+13  2.97e+09 191.70 GB/sec
sub1_copy        :    30000   7.502e+01  19.88  2.501e-03  1.38e-01    1.437e+13  3.48e+09 191.52 GB/sec
-----------------+----------+-----------------------------------------+-------------------------------
Sum of exclusive sections     3.449e+02  91.38                          6.724e+13           194.95 GB/sec
[sum of all processes]                                                 2.690e+14           779.80 GB/sec
-----------------+----------+-----------------------------------------+-------------------------------
[active PMlib elapsed time]   3.774e+02 100.00
```

PMlib report examples

Report for PMLIB_REPORT=BASIC option page2

```
# PMlib hardware performance counter (HWPC) report of the averaged process ------- #

    Report for option HWPC_CHOOSER=BANDWIDTH is generated.

Section          | CMG_bus_RD CMG_bus_WR RD [Bytes] WR [Bytes]  Mem [B/s]    [Bytes]
-----------------+-------------------------------------------------------------------
stream_check (*) :  1.877e+11  7.514e+10  4.805e+13  1.924e+13  1.785e+11  6.728e+13 (*)
sub3_add         :  5.627e+10  1.890e+10  1.441e+13  4.838e+12  1.975e+11  1.924e+13
sub4_triad       :  5.627e+10  1.877e+10  1.441e+13  4.806e+12  1.976e+11  1.921e+13
sub2_scale       :  3.754e+10  1.877e+10  9.610e+12  4.804e+12  1.917e+11  1.441e+13
sub1_copy        :  3.750e+10  1.863e+10  9.601e+12  4.768e+12  1.915e+11  1.437e+13
-----------------+-------------------------------------------------------------------
```

PMlib report examples

Report for PMLIB_REPORT=DETAIL

additional process level statistics

```
## PMlib Process Report --- Elapsed time for individual MPI ranks ------
Section : stream_check (*)
MPI rankID :     call    time[s] time[%]  t_wait[s]  t[s]/call
Rank     0 :       30  3.779e+02   99.9  2.394e-01  1.260e+01
Rank     1 :       30  3.781e+02  100.0  0.000e+00  1.260e+01
Rank     2 :       30  3.752e+02   99.2  2.899e+00  1.251e+01
Rank     3 :       30  3.767e+02   99.6  1.419e+00  1.256e+01
```

```
## PMlib hardware performance counter (HWPC) report for individual MPI ranks ---------
Section : stream_check (*)
MPI rankID : CMG_bus_RD CMG_bus_WR RD [Bytes] WR [Bytes]  Mem [B/s]    [Bytes]
Rank     0 :  1.877e+11  7.514e+10  4.806e+13  1.924e+13  1.781e+11  6.730e+13
Rank     1 :  1.877e+11  7.515e+10  4.805e+13  1.924e+13  1.780e+11  6.729e+13
Rank     2 :  1.877e+11  7.514e+10  4.804e+13  1.924e+13  1.793e+11  6.727e+13
Rank     3 :  1.876e+11  7.514e+10  4.804e+13  1.924e+13  1.786e+11  6.727e+13
```


Report for PMLIB_REPORT=FULL

additional thread level statistics

```
## PMlib Thread Report for MPI rank 0  ----------------------

Section : stream_check (*)
Thread  call  time[s]  t/tav[%] CMG_bus_RD CMG_bus_WR RD [Bytes] WR [Bytes]  Mem [B/s]    [Bytes]
   0      30  3.763e+02  100.0   1.877e+11  7.514e+10  4.806e+13  1.924e+13  1.788e+11  6.730e+13
   1      30  3.763e+02  100.0   1.877e+11  7.517e+10  4.806e+13  1.924e+13  1.789e+11  6.730e+13
   2      30  3.763e+02  100.0   1.878e+11  7.520e+10  4.807e+13  1.925e+13  1.789e+11  6.732e+13
        ...
  11      30  3.763e+02  100.0   1.878e+11  7.520e+10  4.807e+13  1.925e+13  1.789e+11  6.732e+13
```

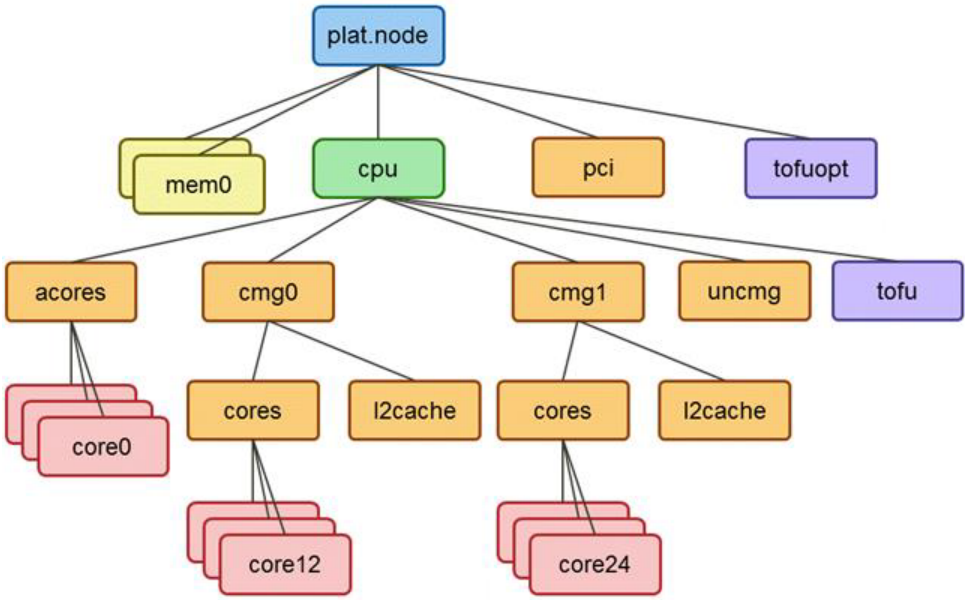
PMlib report examples - HWPC groups

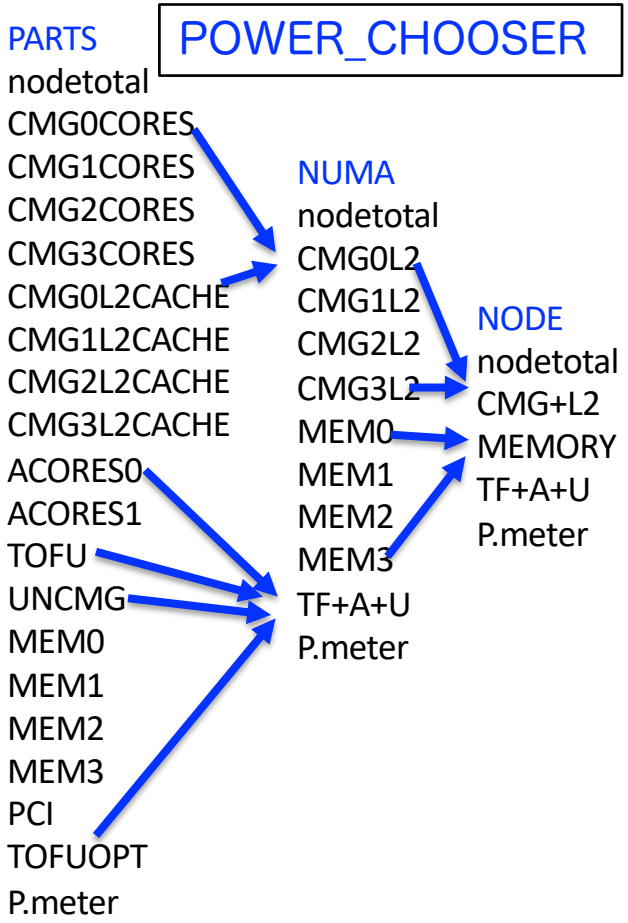
the choice of HWPC_CHOOSER and the corresponding output values

```
 HWPC_CHOOSER     output values
----------------+-------------------------------------------------------------------------------------------
BANDWIDTH       | CMG_bus_RD CMG_bus_WR RD [Bytes] WR [Bytes]  Mem [B/s]    [Bytes]
----------------+-----------------------------------------------------------------
stream_check (*) :  1.877e+11  7.514e+10  4.804e+13  1.924e+13  1.790e+11  6.728e+13 (*)

FLOPS           |     HP_OPS     SP_OPS     DP_OPS   Total_FP   [Flops]    [%Peak]
----------------+-----------------------------------------------------------------
stream_check (*) :  0.000e+00  0.000e+00  2.400e+12  2.400e+12  6.346e+09  8.263e-01 (*)

VECTOR          |  HP_SVE_op  HP_FIX_op  SP_SVE_op  SP_FIX_op  DP_SVE_op  DP_FIX_op   Total_FP  Vector_FP
----------------+-------------------------------------------------------------------------------------------
stream_check (*) :  0.000e+00  0.000e+00  0.000e+00  0.000e+00  2.400e+12  2.532e+07  2.400e+12  2.400e+12
                   [Vector %]
                   1.000e+02 (*)
CACHE           |   LOAD_INS  STORE_INS     L1_HIT     L1_TCM     L2_TCM [L1$ hit%] [L2$ hit%] [L*$ hit%]
----------------+-------------------------------------------------------------------------------------------
stream_check (*) :  9.873e+11  3.328e+11  4.112e+12  1.889e+11  1.878e+11  8.569e+01  7.933e-02  8.577e+01 (*)

CYCLE           |    TOT_CYC    TOT_INS    FP_inst   FMA_inst [FMA_ins%]  [Ins/cyc]
----------------+-----------------------------------------------------------------
stream_check (*) :  7.020e+11  4.647e+12  3.000e+11  8.400e+05  2.800e-04  5.516e-01 (*)

LOADSTORE       |   LOAD_INS  STORE_INS   SVE_LOAD  SVE_STORE SVE_SMV_LD SVE_SMV_ST  GATHER_LD SCATTER_ST
----------------+-------------------------------------------------------------------------------------------
stream_check (*) :  9.827e+11  3.325e+11  6.008e+11  3.010e+11  0.000e+00  0.000e+00  0.000e+00  0.000e+00
                  [Vector %]
                   6.857e+01 (*)
```

Components for power consumption report

Power API measuring objects for Fugaku





- Note
  - The job statistics log file produced by the job manager do not contain ACORES[01]、TofuOPT、PCI values.
  - PMlib produces such measuring stats.

Power consumption report

Report for POWER_CHOOSER=NODE option

```
# PMlib Power Consumption report per node basis --------------------------------- #

    Report is generated for POWER_CHOOSER=NODE option.

     The aggregate power consumption of 4 processes on 1 nodes =  5.86e+04 [J] ==   1.63e+01 [Wh]
     The power consumption of the master node per each section is shown below.
     Remark that only the sections executed by rank 0 thread 0 are shown.

                    Estimated power inside node [W]
Section           |  total | CMG+L2  MEMORY  TF+A+U | Energy[Wh]
------------------+--------+------------------------+----------
stream_check (*) :  155.3   105.6    44.6     8.0   1.63e+01
sub3_add         :  156.5   106.0    45.8     8.1   4.24e+00
sub4_triad       :  158.0   107.4    45.9     8.1   4.27e+00
sub2_scale       :  155.7   106.3    45.0     8.1   3.25e+00
sub1_copy        :  156.2   106.6    45.0     8.1   3.25e+00
------------------+--------+------------------------+----------
```

- Note
  - The power consumption measuring objects do not necessarily correspond to the software objects such as process or thread.

Power consumption report

Report for POWER_CHOOSER=NUMA option

```
 Estimated power inside node [W]
Section           |  total |CMG0+L2 CMG1+L2 CMG2+L2 CMG3+L2   MEM0    MEM1    MEM2    MEM3   TF+A+U | Energy[Wh]
------------------+--------+-------------------------------------------------------------------------+----------
stream_check (*)  :  155.5    26.4    26.4    26.5    26.5    11.2    11.2    11.1    11.1     8.0   1.63e+01
sub3_add          :  156.4    26.3    26.4    26.6    26.5    11.7    11.6    11.3    11.5     8.1   4.23e+00
sub4_triad        :  158.2    27.1    26.9    26.7    26.8    11.7    11.5    11.3    11.5     8.1   4.27e+00
sub2_scale        :  155.9    26.6    26.6    26.7    26.7    11.1    11.3    11.4    11.2     8.1   3.25e+00
sub1_copy         :  156.1    26.6    26.6    26.7    26.7    11.1    11.2    11.3    11.3     8.1   3.25e+00
```

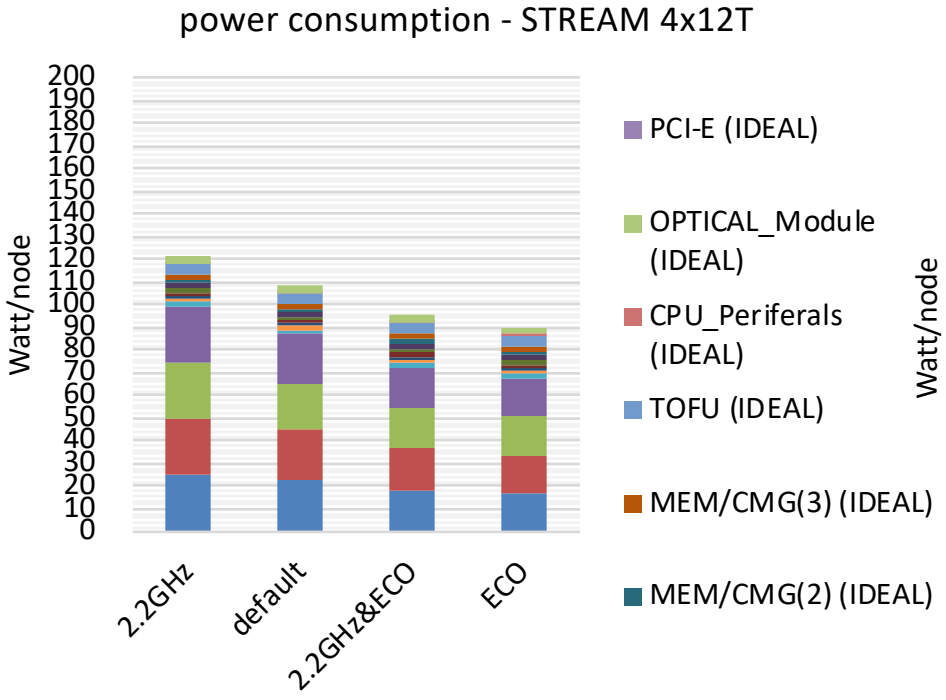
Report for POWER_CHOOSER=PARTS option

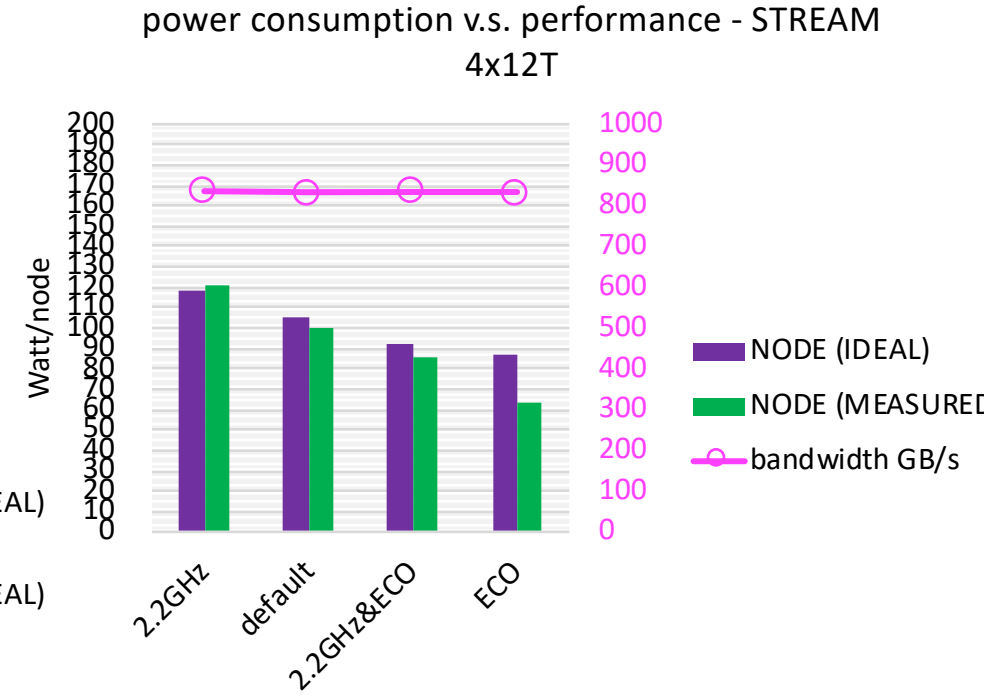
```
 Estimated power inside node [W]
Section           |  total |   CMG0    CMG1    CMG2    CMG3  L2CMG0  L2CMG1  L2CMG2  L2CMG3  Acore0  Acore1  TofuD   UnCMG   MEM0    MEM1    MEM2    MEM3    PCI   TofuOpt | Energy[Wh]
------------------+--------+-------------------------------------------------------------------------------------------------------------------------------------------------------------+----------
stream_check (*)  :  155.5    22.8    22.8    22.8    22.8     3.6     3.6     3.6     3.6     0.5     0.5     4.9     0.3    11.2    11.2    11.1    11.1     0.0     1.7   1.63e+01
sub3_add          :  156.8    22.6    22.9    23.0    22.8     3.7     3.6     3.7     3.7     0.5     0.5     5.0     0.3    11.7    11.5    11.4    11.5     0.0     1.7   4.24e+00
sub4_triad        :  158.1    23.4    23.2    23.1    23.2     3.7     3.6     3.6     3.7     0.5     0.5     5.0     0.3    11.7    11.4    11.3    11.5     0.0     1.7   4.27e+00
sub2_scale        :  156.0    23.0    23.0    23.0    23.0     3.6     3.6     3.7     3.6     0.5     0.5     5.0     0.3    11.1    11.3    11.4    11.2     0.0     1.7   3.26e+00
sub1_copy         :  156.2    23.0    23.0    23.0    23.0     3.6     3.6     3.7     3.6     0.5     0.5     5.0     0.3    11.1    11.3    11.3    11.3     0.0     1.7   3.25e+00
------------------+--------+-------------------------------------------------------------------------------------------------------------------------------------------------------------+----------
```

total				
CMG0	L2CMG0	Acore0	MEM0	PCI
CMG1	L2CMG1	Acore1	MEM1	TofuOpt
CMG2	L2CMG2	TofuD	MEM2	| P.meter
CMG3	L2CMG3	UnCMG	MEM3	| Energy[Wh]

Some benchmark experiments using PMlib

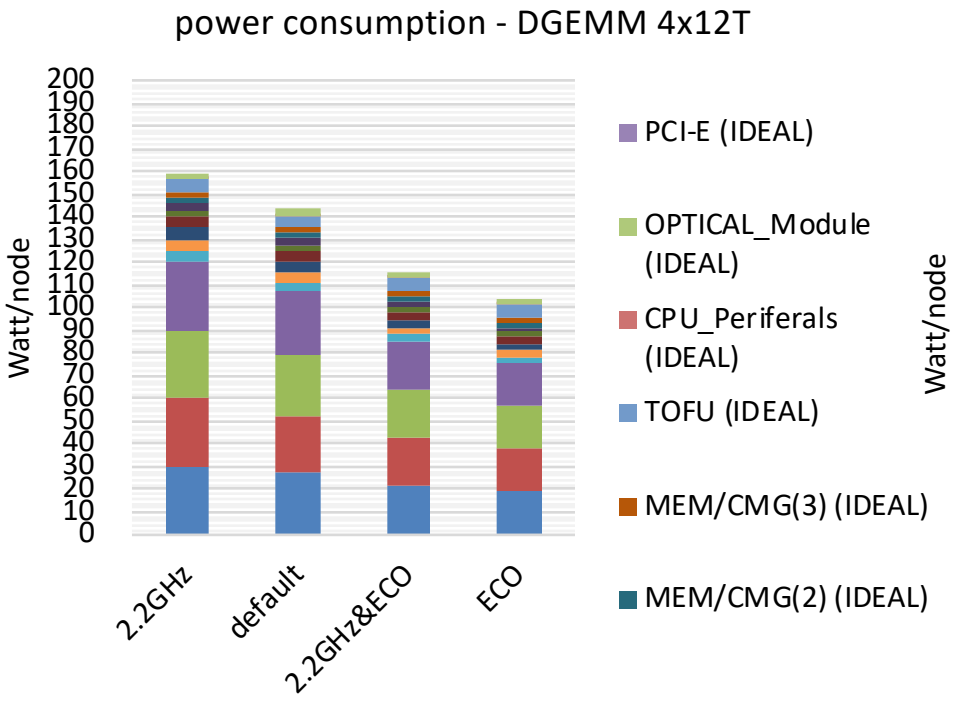
some experiments with PMlib – GHz x Eco mode mix

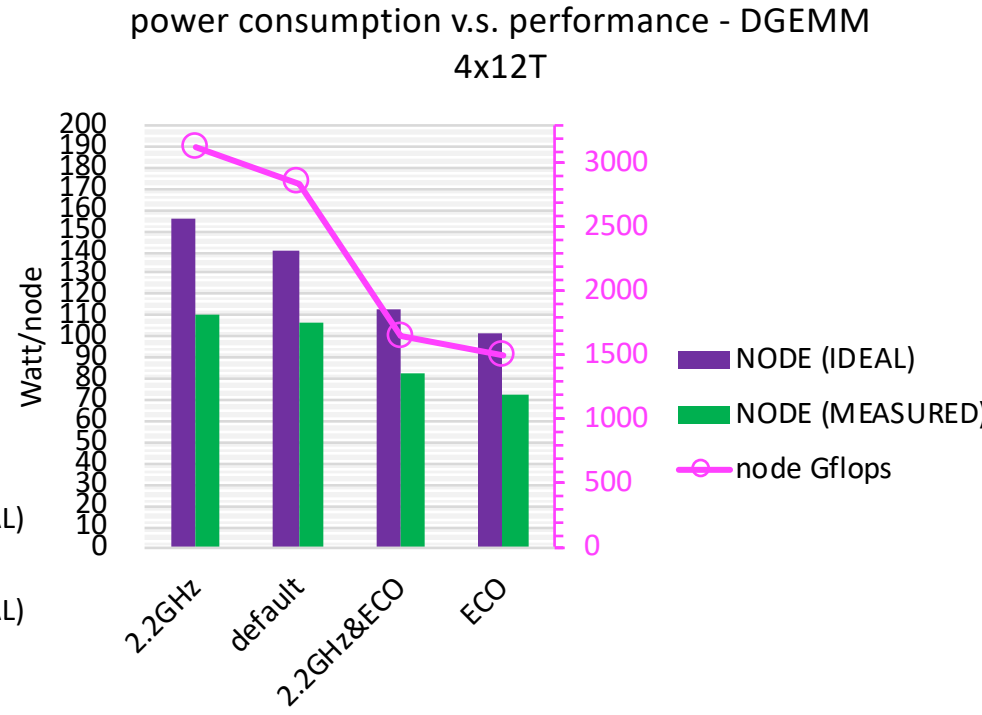




- Note that the tested array size is set much smaller than standard stream

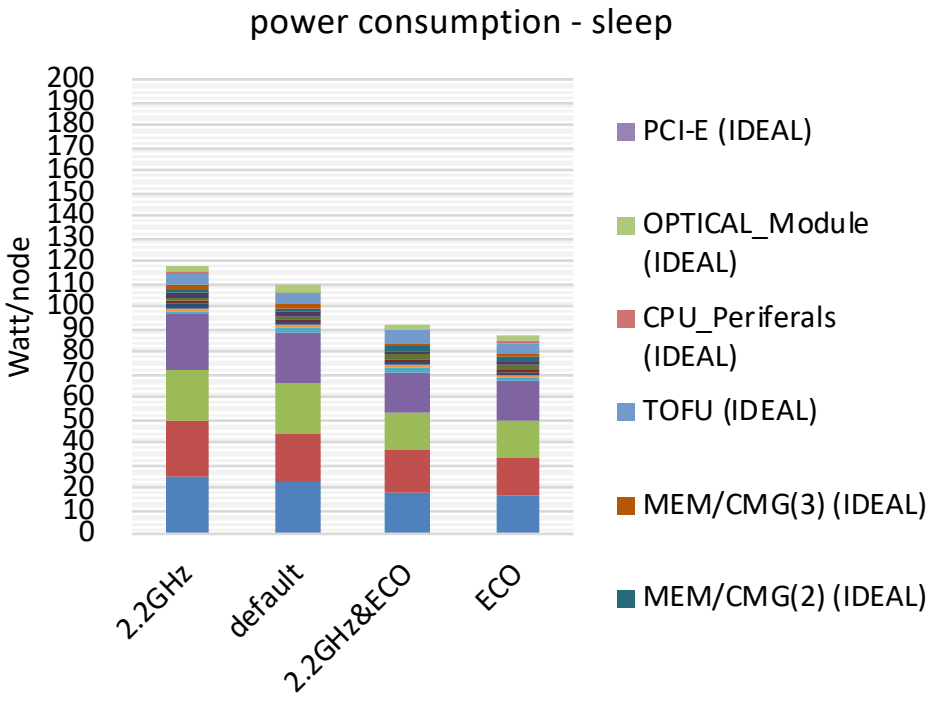
some experiments with PMlib – GHz x Eco mode mix

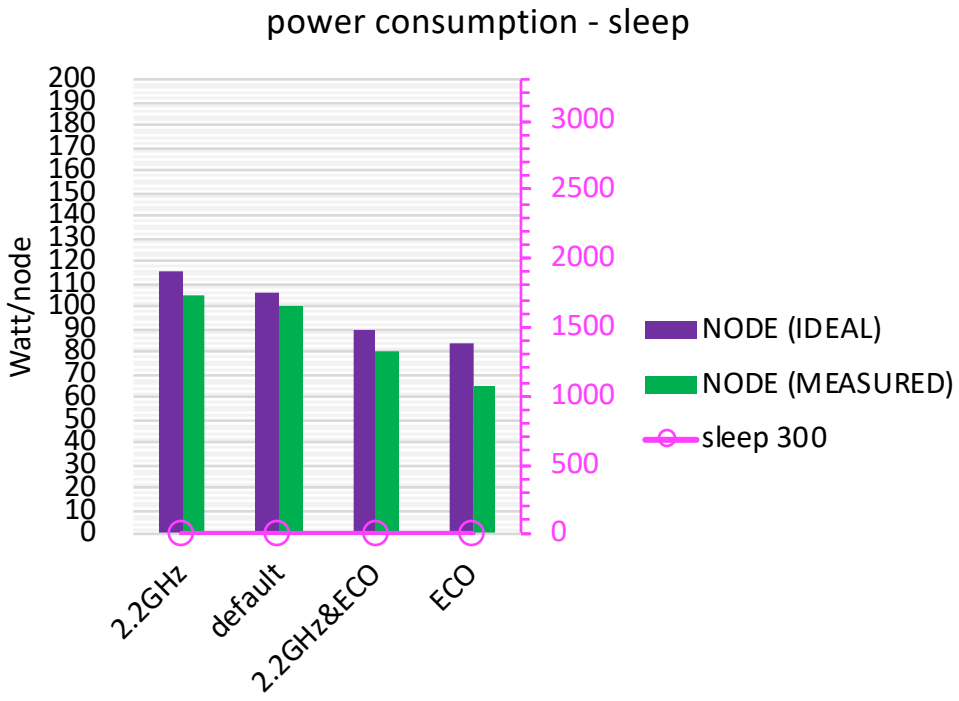




- Note that the tested array size is set much smaller than standard stream

some experiments with PMlib – GHz x Eco mode mix



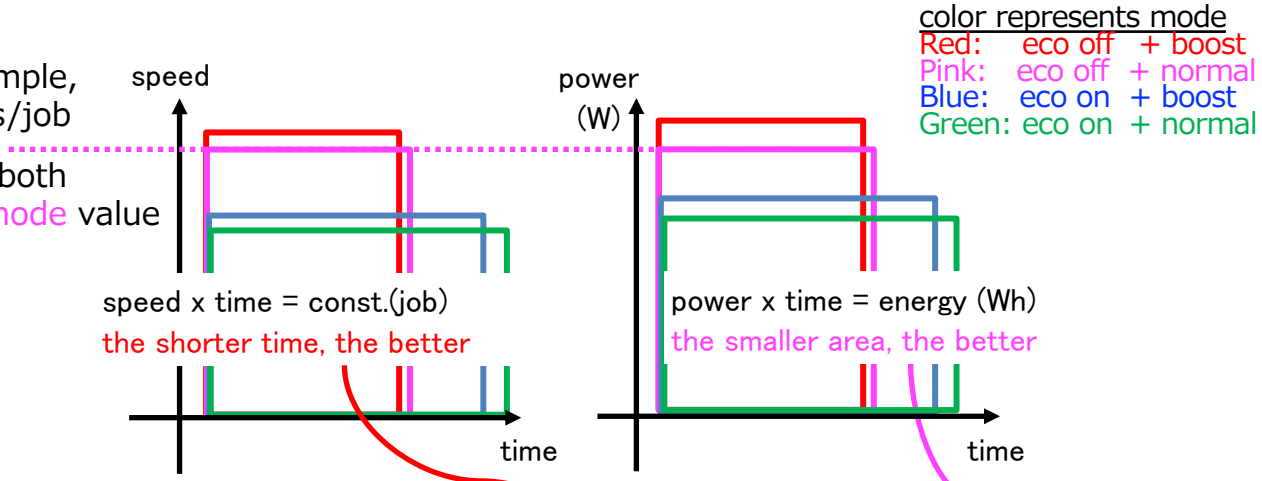


- Note that the tested array size is set much smaller than standard stream

some experiments with PMlib – real application

- **A quick comparison of NTChem target job performance-power balance**
  - Using 4 modes of CPU frequency (2GHz/2.2GHz) and economy (on/off)

- NTChem power profile example, same job, same # of nodes/job
- speed and power axes are both relative to eco off normal mode value



eco mode	freq. mode	speedup /job	time (sec.)	power (W)	energy (Wh)
off	boost	7.8	990.2	161.3	44.4
	normal	7.4	1040.2	143.8	41.5
on	boost	5.5	1408.4	117.4	45.9
	normal	5.1	1526.4	105.6	44.8

Backup slides

PMlib extended API

- extended PMlib APIs - for advanced users

function	C++ API	C API	APIFortran	Python API	arguments
set properties for a section	setProperties	C_pm_setproperties	f_pm_setproperties	N/A	see doxygen in doc
read the power control knob	getPowerKnob	C_pm_getpowerknob	f_pm_getpowerknob	N/A	ditto
Set value to power control knob	setPowerKnob	C_pm_setpowerknob	f_pm_setpowerknob	N/A	ditto
produce report per MPI process group	printGroup	C_pm_printgroup	f_pm_printgroup	N/A	ditto
produce report per MPI communicator	printComm	C_pm_printcomm	f_pm_printcomm	N/A	ditto
start the user measurement section	start	C_pm_start	f_pm_start	start	HWPC_CHOOSER=USER Triggers arithmetic performance measurement
end the user measurement section	stop	C_pm_stop	f_pm_stop	stop	HWPC_CHOOSER=USER 2nd argument specifies the arithmetic formula
Produce OTF trace file	postTrace	C_pm_posttrace	f_pm_posttrace	N/A	OTF v1.1 library must be available on the system

- There are other APIs. The details for the APIs and their arguments are explained in the doc/html pages
  - doc/Readme.md how to generate Doxygen files
  - doc/html/index.html HTML files for Web browser produced by Doxygen

Legend(1)

```
# PMlib Legend - the symbols used in the reports  ----------------------

    PMlib report controll environment variable:

 PMLIB_REPORT=BASIC(default):
       Produce the basic report which contains the averaged timer information, and if available
       the measured HWPC event counts and the performance for each of the sections,
       and if available the estimated power consumption required to execute the section.
       The power consumption is estimated only for the sections run by rank 0 thread0 per node basis.
 PMLIB_REPORT=DETAIL:
       In addition to basic report, produce the process report and the HWPC report.
       The process report contains the section timer information for all the processes.
       the HWPC report contains the HWPC event counts and the performance for for all the processes.
 PMLIB_REPORT=FULL:
       In addition to detail report, produce the thread report which contains the HWPC event counts
       and the performance for each of the OpenMP threads for all the processes.
       The maximum number of threads is limited to the physical number of compute cores per CPU.

 The Section table shows the averaged value from all the processes.
 The total time in the aggregate active sections row is taken from active PMlib elapse time.
 Following annotation symbols are attached to the section lable if judged as so by PMlib.
 (*) : The section is inclusive, i.e. the stats includes other sections inside of it.
       The section without (*) simbol is exclusive, which does not include other section.
 (+) : The section is executed inside of OpenMP parallel region, which can overlap with
       other sections including itself.
       For this type of parallel construct, the execution time must be interpreted carefully
       based on the inclusive section stats for that parallel region, and on the thread report.
       The section without (+) is defined in serial region. It can start parallel region inside.
 The sections without any annotation symbols, i.e. exclusive and in serial region,
 are suited to simply nested loop kernels often seen in HPC applications.
```

Legend(2)

```
Symbols in PMlib hardware performance counter (HWPC) report:
Detected CPU architecture: Fujitsu A64FX
The available HWPC_CHOOSER values and their HWPC events for this CPU are shown below.

HWPC_CHOOSER=FLOPS:
      HP_OPS:    half precision floating point operations
      SP_OPS:    single precision floating point operations
      DP_OPS:    double precision floating point operations
      Total_FP:  total floating point operations
      [Flops]:   floating point operations per second
      [%Peak]:   sustained performance over peak performance
HWPC_CHOOSER=BANDWIDTH:
      CMG_bus_RD:  CMG local memory read counts
      CMG_bus_WR:  CMG local memory write counts
      RD [Bytes]:  CMG local memory read bytes
      WR [Bytes]:  CMG local memory write bytes
      Mem [B/s]:  CMG local memory read&write bandwidth
      [Bytes]  :  CMG local memory read&write bytes
HWPC_CHOOSER=VECTOR:
      HP_SVE_op:  half precision f.p. ops by SVE instructions
      HP_FIX_op:  half precision f.p. ops by scalar/armv8 instructions
      SP_SVE_op:  single precision f.p. ops by SVE instructions
      SP_FIX_op:  single precision f.p. ops by scalar/armv8 instructions
      DP_SVE_op:  double precision f.p. ops by SVE instructions
      DP_FIX_op:  double precision f.p. ops by scalar/armv8 instructions
      Total_FP:   total floating point operations
      Vector_FP:  floating point operations by vector instructions
      [Vector %]: percentage of vectorized f.p. operations
```

Legend(3)

```
HWPC_CHOOSER=CACHE:
      LOAD_INS:   memory load instructions
      STORE_INS:  memory store instructions
      L1_HIT:     L1 data cache hits
      L1_TCM:     L1 data cache misses
      L2_TCM:     L2 cache misses
      [L1$ hit%]: data access hit(%) in L1 cache
      [L2$ hit%]: data access hit(%) in L2 cache
      [L*$ hit%]: sum of hit(%) in L1 and L2 cache
HWPC_CHOOSER=LOADSTORE:
      LOAD_INS:   memory load instructions
      STORE_INS:  memory store instructions
      SVE_LOAD:   memory read by SVE and Advanced SIMD load instructions.
      SVE_STORE:  memory write by SVE and Advanced SIMD store instructions.
      SVE_SMV_LD: memory read by SVE and Advanced SIMD multiple vector contiguous structure load instructions.
      SVE_SMV_ST: memory write by SVE and Advanced SIMD multiple vector contiguous structure store
instructions.
      GATHER_LD:  memory read by SVE non-contiguous gather-load instructions.
      SCATTER_ST: memory write by SVE non-contiguous scatter-store instructions.
      [Vector %]: percentage of SVE load/store instructions over all load/store instructions.
HWPC_CHOOSER=CYCLE:
      TOT_CYC:   total cycles
      TOT_INS:   total instructions
      FP_inst:   floating point instructions
      FMA_inst:  fused multiply+add instructions
      [FMA_ins%]: percentage of FMA instructions over all f.p. instructions
      [Ins/cyc]: performed instructions per machine clock cycle
HWPC_CHOOSER=USER:
      User provided argument values (Arithmetic Workload) are accumulated and reported.
```

Legend(4)

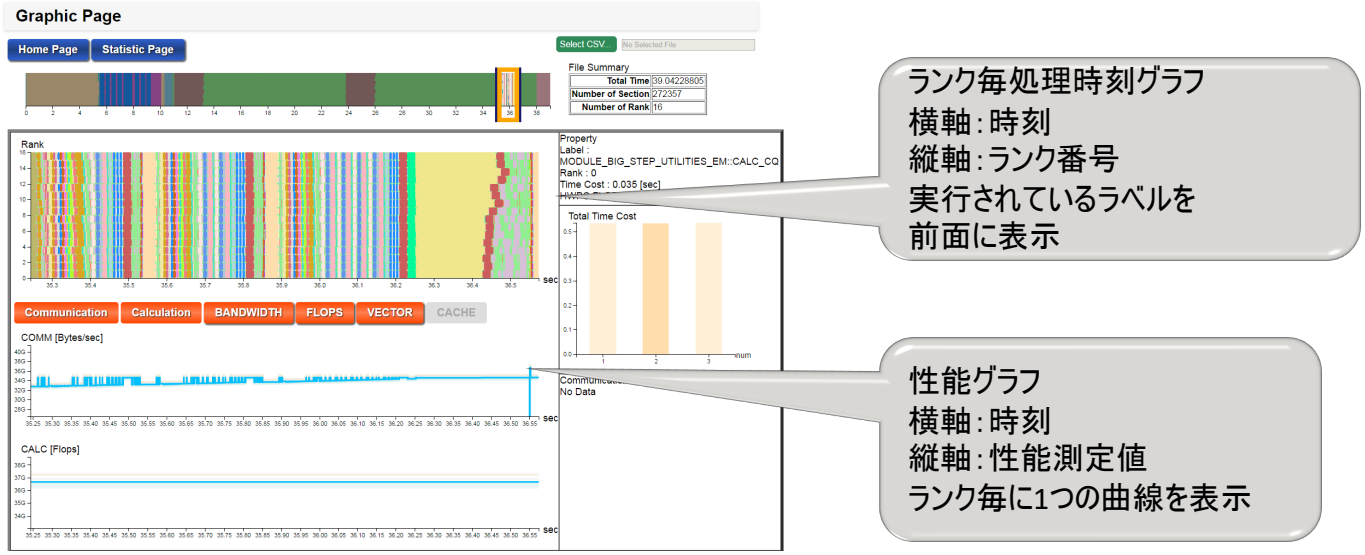
```
Remarks.
      Symbols represent HWPC (hardware performance counter) native and derived events
      Symbols in [] are frequently used performance metrics which are calculated from these events.
      The values in the Basic Report section shows the arithmetic mean value of the processes.
      The values in the Process Report section shows the sum of threads generated by the process.
      The values in the Thread Report section shows the precise thread level statistics.
Special remarks for A64FX BANDWIDTH report.
      CMG_bus_RD and CMG_bus_WR both count the CMG aggregated values, not core.
      Basic Report and Process Report both show the appropriate statistics.
      But Thread Report shows the values in redundant manner.
```

Legend(5)

```
Symbols in PMlib power consumption report:
The available POWER_CHOOSER values and their output data are shown below.

POWER_CHOOSER=OFF(default):
      power consumption report is not produced:
POWER_CHOOSER=NODE:
      total     : Total of all parts. (CMG + MEMORY + TF+A+U)
      CMG+L2    : All compute cores and L2 cache memory in all 4 CMGs
      MEMORY    : Main memory (HBM)
      TF+A+U    : TofuD network router and interface + Assistant cores + other UnCMG parts
      Energy[Wh]: power comsumption in watt-hour unit
POWER_CHOOSER=NUMA:
      total     : Total of all parts. (CMG[0-3] + MEM[0-3] + TF+A+U)
      CMG0+L2   : compute cores and L2 cache memory in CMG0. ditto for CMG[1-3]+L2.
      MEM[0-3]  : Main memory (HBM) attached to CMG0[1,2,3]
      TF+A+U    : TofuD network router and interface + Assistant cores + other UnCMG parts
      Energy[Wh]: power comsumption in watt-hour unit
POWER_CHOOSER=PARTS:
      total     : Total of all parts.
      CMG[0-3]  : compute cores in CMG0, CMG1, CMG2, CMG3
      L2CMG[0-3]: L2 cache memory in CMG0, CMG1, CMG2, CMG3
      Acore[0-1]: Assistant core 0, 1.
      TofuD     : TofuD network router and interface
      UnCMG     : Other UnCMG parts (CPU parts excluding compute cores, assistant cores or TofuD)
      PCI       : PCI express interface
      TofuOpt   : Tofu optical modules
      Energy[Wh]: power comsumption in watt-hour unit
```

Optional interface with legacy OTF V1

- OTF - Open Trace Format- version 1.1 file output provisioning
  - Visualize the computing performance along with the time history



Needs pre-installed OTF V1.1 library on the system.
Needs OTF V1.1 compliant visualization software.
Experimental Web based visualization software – TRAiL – is available per request basis. Looks pretty, but it was never tested with large size jobs