pyJoules

Release 0.2.0

INRIA, University of Lille

Oct 05, 2021

CONTENTS

CHAPTER

ONE

ABOUT

pyJoules is a software toolkit to measure the energy footprint of a host machine along the execution of a piece of Python code. It monitors the energy consumed by specific device of the host machine such as :

- intel CPU socket package
- RAM (for intel server architectures)
- intel integrated GPU (for client architectures)
- nvidia GPU

1.1 Limitation

1.1.1 CPU, RAM and integrated GPU

pyJoules uses the Intel "*Running Average Power Limit*" (RAPL) technology that estimates energy consumption of the CPU, ram and integrated GPU. This technology is available on Intel CPU since the Sandy Bridge generation (2010).

For the moment, **pyJoules** use the linux kernel API to get energy values reported by RAPL technology. That's why CPU, RAM and integrated GPU energy monitoring is not available on windows or MacOS.

As RAPL is not provided by a virtual machine, **pyJoules** can't use it anymore to monitor energy consumption inside a virtual machine.

1.1.2 Nvidia GPU

pyJoules uses the nvidia "*Nvidia Management Library*" technology to measure energy consumption of nvidia devices. The energy measurement API is only available on nvidia GPU with Volta architecture (2018)

1.1.3 Monitor only function energy consumption

pyjoules monitor device energy consumption. The reported energy consumption is not only the energy consumption of the code you are running. This includes the *global energy consumption* of all the process running on the machine during this period, thus including the operating system and other applications.

That is why we recommend to eliminate any extra programs that may alter the energy consumption of the machine hosting experiments and to keep only the code under measurement (*i.e.*, no extra applications, such as graphical interface, background running task…). This will give the closest measure to the real energy consumption of the measured code.

CHAPTER

TWO

QUICKSTART

2.1 Installation

You can install **pyJoules** with pip : `pip install pyJoules`

2.2 Decorate a function to measure its energy consumption

To measure the energy consumed by the machine during the execution of the function `foo()` run the following code:

To measure the energy consumed by the machine during the execution of the function `foo()` run the following code with the `measure_energy` $$ $decorator$:

```
from pyJoules.energy_meter import measure_energy

@measure_energy
def foo():
    # Instructions to be evaluated.

foo()
```

This will print in the console the recorded energy consumption of all the monitorable devices during the execution of function `foo`.

CHAPTER

THREE

MISCELLANEOUS

PyJoules is an open-source project developed by the Spirals research group (University of Lille and Inria) that take part of the Powerapi initiative.

3.1 Mailing list and contact

You can contact the developer team with this address : powerapi-staff@inria.fr

You can follow the latest news and asks questions by subscribing to our mailing list

3.2 Contributing

If you would like to contribute code you can do so via GitHub by forking the repository and sending a pull request.

When submitting code, please make every effort to follow existing coding conventions and style in order to keep the code as readable as possible.

CHAPTER

FOUR

TABLE OF CONTENTS

4.1 Usage

4.1.1 Decorator

Decorate a function to measure its energy consumption

To measure the energy consumed by the machine during the execution of the function `foo()` run the following code:

To measure the energy consumed by the machine during the execution of the function `foo()` run the following code with the <a href="../API/main$_api.html#pyJoules.energy_meter.measure_energy" > `measure_energy` < /a > $decorator$:

```
from pyJoules.energy_meter import measure_energy

@measure_energy
def foo():
    # Instructions to be evaluated.

foo()
```

This will print in the console the recorded energy consumption of all the monitorable devices during the execution of function `foo`.

Configure the decorator specifying the device to monitor

You can easily configure which device to monitor using the parameters of the <a href="/API/main$_api.html#pyJoules.energy_meter.measure_energy" > `measure_energy` < /a > $decorator.For example, the following example only monitors the CPU energy consumption on the CPU socket 1 and the NvidiaGP$

```
from pyJoules.energy_meter import measure_energy
from pyJoules.device.rapl_device import RaplPackageDomain
from pyJoules.device.nvidia_device import NvidiaGPUDomain

@measure_energy(domains=[RaplPackageDomain(1), NvidiaGPUDomain(0)])
def foo():
    # Instructions to be evaluated.

foo()
```

for more information about device you can monitor, see here :

Configure the output of the decorator

If you want to handle data with different output than the standard one, you can configure the decorator with an EnergyHandler instance from the pyJoules.handler module.

As an example, if you want to write the recorded energy consumption in a .csv file:

```
from pyJoules.energy_meter import measure_energy
from pyJoules.handler.csv_handler import CSVHandler

csv_handler = CSVHandler('result.csv')

@measure_energy(handler=csv_handler)
def foo():
# Instructions to be evaluated.

for _ in range(100):
    foo()

csv_handler.save_data()
```

This will produce a csv file of 100 lines. Each line containing the energy consumption recorded during one execution of the function `foo`. Other predefined `Handler` classes exist to export data to *MongoDB* and *Panda* dataframe.

4.1.2 Context manager

Use a context manager to add tagged "breakpoint" in your measurement

If you want to know where is the *hot spots* where your python code consume the most energy you can add *breakpoints* during the measurement process and tag them to know amount of energy consumed between this breakpoints.

For this, you have to use a context manager to measure the energy consumption. It is configurable as the decorator . For example, here we use an EnergyContext to measure the energy consumption of CPU 1 and nvidia gpu 0 and report it in a csv file

```
from pyJoules.energy_meter import EnergyContext
from pyJoules.device.rapl_device import RaplPackageDomain
from pyJoules.device.nvidia_device import NvidiaGPUDomain
from pyJoules.handler.csv_handler import CSVHandler

csv_handler = CSVHandler('result.csv')

with EnergyContext(handler=csv_handler, domains=[RaplPackageDomain(1),␣
↪NvidiaGPUDomain(0)], start_tag='foo') as ctx:
    foo()
    ctx.record(tag='bar')
    bar()

csv_handler.save_data()
```

This will record the energy consumed :

- between the beginning of the `EnergyContext` $$ and the call of the `ctx.record` method

- between the call of the `ctx.record` method and the end of the <a href="../API/main$_api.html#pyJoules.energy_meter.EnergyContext$" `EnergyContext` $$

Each measured part will be written in the csv file. One line per part.

4.1.3 Manual usage

Use a EnergyMeter to measure energy consumption without decorator or context manager

If you want need more flexibility and measure energy consumption of piece of code that can't be bound inside a decorated function of a context manager, you can use an instance of `EnergyMeter` $$.

Instance of `EnergyMeter` $$ $is the under layer tool used by context manager and decorator to measure energy consumption.$

Create the EnergyMeter

Before using an energy meter, you have to create it with devices that it have to monitor. For this, use an `DeviceFactory` $$ $to create and configure the monitored devices$

The following piece of code show how to create an `EnergyMeter` $$ $that monitor CPU, DRAM and GPU energy consumption.$

```
domains = [RaplPackageDomain(0), RaplDramDomain(0), NvidiaGPUDomain(0)]
devices = DeviceFactory.create_devices(domains)
meter = EnergyMeter(devices)
```

Tips : call the `DeviceFactory.create_devices` $$ $without parameter to get the list of all monitorable devices.$

Use the EnergyMeter

When you have your `EnergyMeter` $$ $you can use it to measure energy consumption of piece of code.$

An `EnergyMeter` $$ $have three main method$:

> `start` $$: $to start the energy consumption monitoring$

 `record` $$: $to tag a hotspot in monitored piece of code$

 `stop` $$: $to stop the energy consumption monitoring$

The following piece of code show how to use an `EnergyMeter` $$ $to monitor piece of code$:

```
meter.start(tag='foo')
foo()
meter.record(tag='bar')
bar()
meter.stop()
```

Get the EnergyTrace

When you finished to measure the energy consumed during execution of your piece of code, you can retrieve its energy trace using the <a href="../API/main_api.html#pyJoules.energy_meter.EnergyMeter.get_trace" > EnergyMeter.get_trace < /a > method

This will return an iterator on some <a href="../API/main_api.html#pyJoules.energy_trace.EnergySample" > EnergySample < /a > .Eachenergysamplecontainsenergyconsumptioninformationmeasuredbetweeneachcallto < ahref = "../API/main_api.html#pyJoules.energy_meter.EnergyMeter.start" > start < /a >, < ahref = "../API/main_api.html#pyJoules.energy_meter.EnergyMeter.record" > record < /a > and < ahref = "../API/main_api.html#pyJoules.energy_meter.EnergyMeter.stop" > stop < /a > method.

For example, the trace of the previous example contains two <a href="../API/main_api.html#pyJoules.energy_trace.EnergySample" > EnergySample < /a > .Onethatcontainstheenergymeasuredbetween < ahref = "../API/main_api.html#pyJoules.energy_meter.EnergyMeter.start" > start < /a > and < ahref = "../API/main_api.html#pyJoules.energy_meter.EnergyMeter.record" > record < /a > methods(duringfoomethodexecution)andthesecondthatcontainsenergymeasuredbetween < ahref = "../API/main_api.html#pyJoules.energy_meter.EnergyMeter.record" > record < /a > and < ahref = "../API/main_api.html#pyJoules.energy_meter.EnergyMeter.stop" > stop < /a > method(duringbarmethodexecution).

Energy sample contains :

- a tag
- a timestamp (the beginning of the measure)
- the duration of the measure
- the energy consumed during the measure

Full Example

```
from pyJoules.device import DeviceFactory
from pyJoules.device.rapl_device import RaplPackageDomain, RaplDramDomain
from pyJoules.device.nvidia_device import NvidiaGPUDomain
from pyJoules.energy_meter import EnergyMeter

domains = [RaplPackageDomain(0), RaplDramDomain(0), NvidiaGPUDomain(0)]
devices = DeviceFactory.create_devices(domains)
meter = EnergyMeter(devices)

meter.start(tag='foo')
foo()
meter.record(tag='bar')
bar()
meter.stop()
```

(continues on next page)

(continued from previous page)

```
trace = meter.get_trace()
```

4.2 Handlers

4.2.1 Print Handler

This handler print the measured energy sample on the standard output

How to Use it

This handler is the default handler. When you use a context manager and decorated function without specifying any handler, the `PrintHandler` will be used

Example :

```
with EnergyContext(domains=[RaplPackageDomain(1), NvidiaGPUDomain(0)], start_tag=
↪'foo') as ctx:
    foo()
    ctx.record(tag='bar')
    bar()
```

Output

The previous example will produce the following result on the standard output

```
begin timestamp : AAAA; tag : foo; duration : BBBB; package_0 : CCCC; nvidia_gpu_0␣
↪: DDDD
begin timestamp : AAAA2; tag : bar; duration : BBBB2; package_0 : CCCC2; nvidia_gpu_
↪0 : DDDD2
```

with :

- AAAA* : timestamp of the measured interval beginning
- BBBB* duration of the measured interval (in seconds)
- CCCC* energy consumed by CPU 0 during the measured interval
- DDDD* energy consumed by GPU 0 during the measured interval

4.2.2 CSV Handler

This handler save the measured energy sample on a csv file

How to Use it

Create a `CSVHandler` instance and pass it as a parameter of the context manager or the function decorator. You have to specify the filename of the file that will store the energy sample.

When the measure is done, you have to use the `save_data` method to write the data on disk.

Example :

```
from pyJoules.handler.csv_handler import CSVHandler
csv_handler = CSVHandler('result.csv')

with EnergyContext(handler=csv_handler, domains=[RaplPackageDomain(1),␣
↪NvidiaGPUDomain(0)], start_tag='foo') as ctx:
    foo()
    ctx.record(tag='bar')
    bar()

csv_handler.save_data()
```

Output

The previous example will produce the following csv file `result.csv`

```
timestamp;tag;duration;package_0;nvidia_gpu_0
AAAA;foo;BBBB;CCCC;DDDD
AAAA2;bar;BBBB2;CCCC2;DDDD2
```

with :

- AAAA* : timestamp of the measured interval beginning
- BBBB* duration of the measured interval (in seconds)
- CCCC* energy consumed by CPU 0 during the measured interval
- DDDD* energy consumed by GPU 0 during the measured interval

4.2.3 Pandas Handler

This Handler save the measured energy sample on a panda Dataframe

How to Use it

Create a `PandasHandler` instance and pass it as a parameter of the context manager or the function decorator.

When the measure is done, you can retrieve the dataframe using the `get_dataframe` method

Example :

```
from pyJoules.handler.pandas_handler import PandasHandler
pandas_handler = PandasHandler()

with EnergyContext(handler=pandas_handler, domains=[RaplPackageDomain(1),␣
↪NvidiaGPUDomain(0)], start_tag='foo') as ctx:
    foo()
    ctx.record(tag='bar')
    bar()

df = pandas_handler.get_dataframe()
```

Output

This will produce the following dataframe :

```
  timestamp  tag  duration  package_0  nvidia_gpu_0
0  AAAA  foo     BBBB        CCCC        DDDD
1  AAAA2  bar     BBBB2        CCCC2        DDDD2
```

with :

- AAAA* : timestamp of the measured interval beginning
- BBBB* duration of the measured interval (in seconds)
- CCCC* energy consumed by CPU 0 during the measured interval
- DDDD* energy consumed by GPU 0 during the measured interval

4.2.4 MongoDB Handler

This handler save the measured energy sample on a mongoDB database

How to Use it

Create a `MongoHandler` instance and pass it as a parameter of the context manager or the function decorator. You have to specify the uri of the database, the database and the collection name.

When the measure is done, you have to use the `save_data` method to store the data on base.

Example :

```
from pyJoules.handler.mongo_handler import MongoHandler
mongo_handler = MongoHandler(uri='mongodb://localhost', database_name='db',␣
↪collection_name='collection')

with EnergyContext(handler=mongo_handler, domains=[RaplPackageDomain(1),␣
↪NvidiaGPUDomain(0)], start_tag='foo') as ctx:
```

(continues on next page)

(continued from previous page)

```
    foo()
    ctx.record(tag='bar')
    bar()

mongo_handler.save_data()
```

Output

The previous example will store the following record on mongo db database

```
{
   "name":"trace_0",
   "trace":[
      {
         "timestamp":"AAAA",
         "tag":"foo",
         "duration":"BBBB",
         "energy":{
            "package_0":"CCCC",
            "nvidia_gpu_0":"DDDD"
         }
      },
      {
         "timestamp":"AAAA2",
         "tag":"bar",
         "duration":"BBBB2",
         "energy":{
            "package_0":"CCCC2",
            "nvidia_gpu_0":"DDDD2"
         }
      }
   ]
}
```

with :

- AAAA* : timestamp of the measured interval beginning
- BBBB* duration of the measured interval (in seconds)
- CCCC* energy consumed by CPU 0 during the measured interval
- DDDD* energy consumed by GPU 0 during the measured interval

Trace name

Each trace stored in the database is named. Trace name is computed by adding an integer (which is incremented each time a new trace is stored) to a string prefix. By default, this prefix is `trace` so the first trace you store will be named `trace_0`, the second `trace_1`.

You can change this default prefix by specifying the `trace_name_prefix` with the prefix you want to use.

4.3 Devices

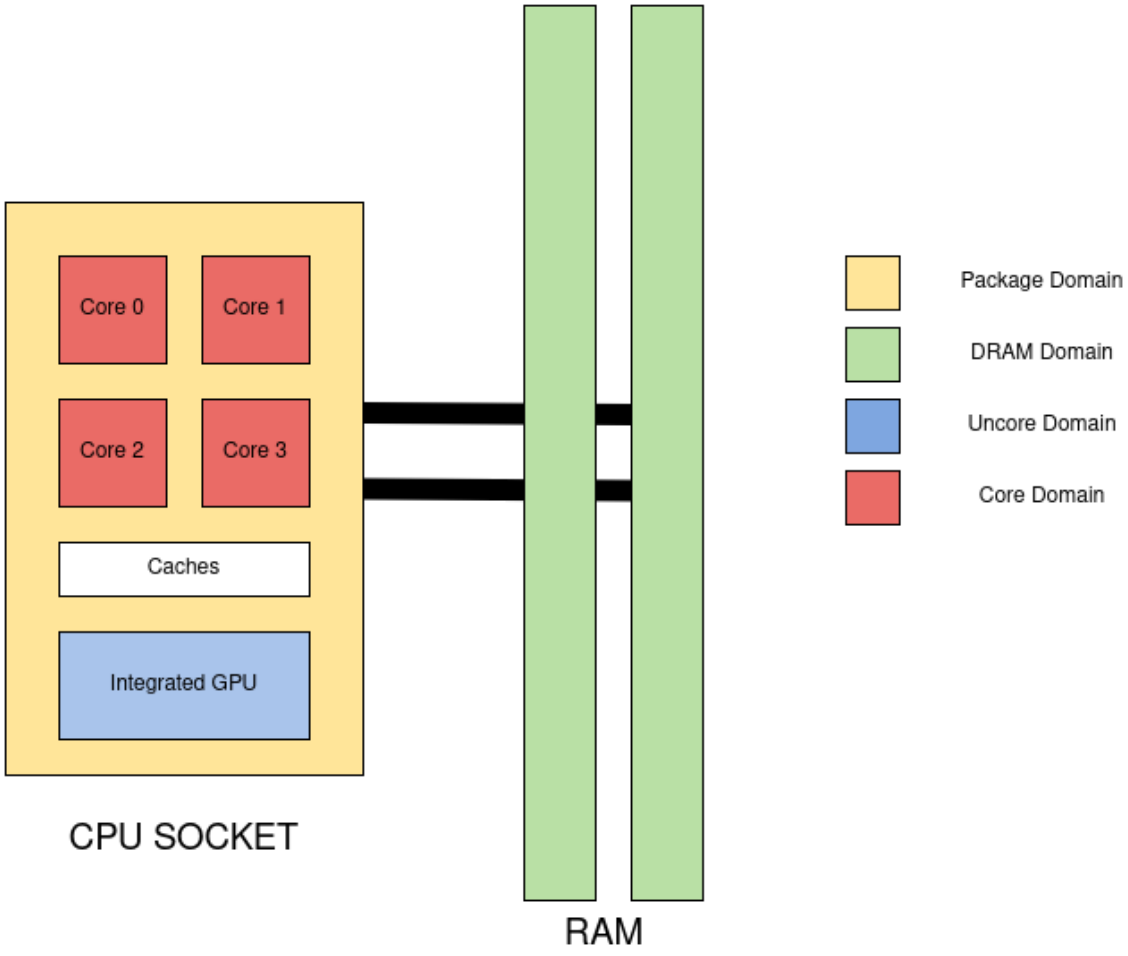
4.3.1 Intel CPU

PyJoules support energy consumption monitoring of intel cpu using the "Running Average Power Limit" (RAPL) technology. RAPL is available on CPU since the Sandy Bridge generation (2010)

Domains

You can monitor energy consumption from several part of a CPU, called domain.

Each monitorable domain is described on this image :



With :

- Package : correspond to the wall cpu energy consumption
- core : correpond to the sum of all cpu core energy consumption
- uncore : correspond to the integrated GPU

Usage

To configure your function decorator, context manager or energy meter to measure specific part of a CPU, pass as `domain` attribute a list of instance of a subClass of `pyJoules.device.rapl_device.RaplDomain` corresponding to the domain you want to monitor.

For example, if you want to configure a context manager to measure the energy consumed by the Core domain follow this example :

```
from pyJoules.device.rapl_device import RaplCoreDomain
with EnergyContext(domains=[RaplCoreDomain(0)):
    foo()
```

You can use the following class to select the list of domain you want to monitor :

- `RaplPackageDomain` : whole CPU socket (specify the socket id in parameter)
- `RaplDramDomain` : RAM (specify the socket id in parameter)
- `RaplUncoreDomain` : integrated GPU (specify the socket id in parameter)
- `RaplCoreDomain` : RAPL Core domain (specify the socket id in parameter)

4.3.2 Nvidia GPU

pyJoules uses the nvidia "*Nvidia Management Library*" technology to measure energy consumption of nvidia devices. The energy measurement API is only available on nvidia GPU with Volta architecture (2018)

Usage

To configure your function decorator, context manager or energy meter to measure the energy consumption of a GPU, pass as `domain` attribute a list of instance of `pyJoules.device.rapl_device.NvidiaGPUDomain`.

For example, if you want to configure a context manager to measure the energy consumed by the gpu of id `0` follow this example :

```
from pyJoules.device.nvidia_device import NvidiaGPUDomain
with EnergyContext(domains=[NvidiaGPUDomain(0)):
    foo()
```

4.4 API

4.4.1 Core modules

Decorator

@pyJoules.energy_meter.**measure_energy**(*func=None*, *handler=<pyJoules.handler.print_handler.PrintHandler object>*, *domains=None*)

Measure the energy consumption of monitored devices during the execution of the decorated function

Parameters

- **handler** (*EnergyHandler*) – handler instance that will receive the power consummation data
- **domains** (Optional[List[*Domain*]]) – list of the monitored energy domains

Context

class pyJoules.energy_meter.**EnergyContext**(*handler=<pyJoules.handler.print_handler.PrintHandler object>*, *domains=None*, *start_tag='start'*)

__init__(*handler=<pyJoules.handler.print_handler.PrintHandler object>*, *domains=None*, *start_tag='start'*)

Measure the energy consumption of monitored devices during the execution of the contextualized code

Parameters

- **handler** (*EnergyHandler*) – handler instance that will receive the power consummation data
- **domains** (Optional[List[*Domain*]]) – list of the monitored energy domains
- **start_tag** (str) – first tag of the trace

Class

class pyJoules.energy_meter.**EnergyMeter**(*devices*, *default_tag=''*)

Tool used to record the energy consumption of given devices

__init__(*devices*, *default_tag=''*)

Parameters

- **devices** (List[*Device*]) – list of the monitored devices
- **default_tag** (str) – tag given if no tag were given to a measure

gen_idle(*trace*)

generate idle values of an energy trace for each sample, wait for the duraction of a sample and measure the energy consumed during this period

Return type List[Dict[str, float]]

Returns the list of idle energy consumption for each sample in the trace

get_trace()
return the last trace measured

Raises *EnergyMeterNotStoppedError* – if the energy meter isn't stopped

Return type *EnergyTrace*

record(*tag=None*)
Add a new state to the Trace

Parameters **tag** (Optional[str]) – sample name

Raises *EnergyMeterNotStartedError* – if the energy meter isn't started

resume(*tag=None*)
resume the energy Trace (if no energy trace was launched, start a new one

Parameters **tag** (Optional[str]) – sample name

Raises *EnergyMeterNotStoppedError* – if the energy meter isn't stopped

start(*tag=None*)
Begin a new energy trace

Parameters **tag** (Optional[str]) – sample name

stop()
Set the end of the energy trace

Raises *EnergyMeterNotStartedError* – if the energy meter isn't started

class pyJoules.energy_trace.**EnergyTrace**(*samples*)
Trace of all EnergySample collected by a meter

__init__(*samples*)

Parameters **samples** (List[*EnergySample*]) – samples containing in the trace

append(*sample*)
append a new sample to the trace

clean_data(*guards=[]*)
Remove sample with negative energy values from the trace Guards can be added to specify rules to remove sample

Parameters **guards** (List[Callable[[*EnergySample*], bool]]) – list of function that is used as rules to remove samples. A guard is a function that take a sample as parameter and return True if it must be keept in the trace, False otherwise

remove_idle(*idle*)
substract idle energy values from the current trace

Parameters **idle** (List[Dict[str, float]]) – list of idle consumption values to substract to current trace idle consumption values must be grouped in a dictionary with their domain as key

Raises **ValueError** – if the number of values in the list doesn't match the number of sample in the trace or if a domain of the trace is not in the idle values

class pyJoules.energy_trace.**EnergySample**(*timestamp*, *tag*, *duration*, *energy*)

Variables

- **timestamp** (*float*) – begining timestamp

- **tag** (*str*) – sample tag
- **duration** (*float*) – duration of the sample in seconds
- **energy** (*Dict[str, float]*) – dictionary that contains the energy consumed during this sample

class pyJoules.device.device_factory.**DeviceFactory**

static **create_devices**(*domains=None*)
Create and configure the Device instance with the given Domains

Parameters **domains** (Optional[*Domain*]) – a list of Domain instance that as to be monitored (if None, return a list of all monitorable devices)

Return type List[*Device*]

Returns a list of device configured with the given Domains

Raises

- ***NoSuchDeviceError*** – if a domain depend on a device that doesn't exist on the current machine
- ***NoSuchDomainError*** – if the given domain is not available on the device

Exception

exception pyJoules.exception.**PyJoulesException**
PyJoules exceptions parent class

exception pyJoules.exception.**NoSuchDomainError**(*domain_name*)
Exception raised when a user ask to monitor a domain that is not available on the given device

domain_name
the domain name that is not available on this device

exception pyJoules.exception.**NoSuchDeviceError**
Exception raised when a Device that does not exist on the current machine is created

exception pyJoules.energy_meter.**NoNextStateException**
Exception raised when trying to compute duration or energy from a state which is the last state of an energy trace.

exception pyJoules.energy_meter.**StateIsNotFinalError**
Exception raised when trying to add a state to a non final state on an energy trace

exception pyJoules.energy_meter.**EnergyMeterNotStartedError**
Exception raised when trying to stop or record on a non started EnergyMeter instance

exception pyJoules.energy_meter.**EnergyMeterNotStoppedError**
Exception raised when trying to get energy samples from non stopped EnergyMeter instance

exception pyJoules.energy_meter.**SampleNotFoundError**
Exception raised when trying to retrieve a sample that does not exist on trace

4.4.2 Handler API

Abstract Class

class pyJoules.handler.**EnergyHandler**
An object that can handle the measured value of an energy trace

process(*trace*)

Class

class pyJoules.handler.print_handler.**PrintHandler**

process(*trace*)
Print the given sample on the standard output

class pyJoules.handler.csv_handler.**CSVHandler**(*filename*)

__init__(*filename*)

Parameters **filename** (str) – file name to store processed trace

save_data()
append processed trace to the file

class pyJoules.handler.mongo_handler.**MongoHandler**(*uri*, *database_name*, *collection_name*, *connected_timeout=30000*, *trace_name_prefix='trace_'*)

__init__(*uri*, *database_name*, *collection_name*, *connected_timeout=30000*, *trace_name_prefix='trace_'*)
Create a handler that will store data on mongo database

Parameters

- **uri** (str) – database uri using mongoDB uri format
- **connection_timeout** – Controls how long (in milliseconds) the driver will wait to find an available, appropriate server to carry out a database operation; while it is waiting, multiple server monitoring operations may be carried out, each controlled by connectTimeoutMS. Defaults to 30000 (30 seconds).
- **trace_name_prefix** (str) – prefix of the trace name used to identify a trace in mongo database. The trace name is computed as follow : trace_name_prefix + trace_position (trace position is the position of the current trace in the trace list processed by the handler)

save_data()
Save processed trace on the database

class pyJoules.handler.pandas_handler.**PandasHandler**
handle energy sample to convert them into pandas DataFrame

get_dataframe()
return the DataFrame containing the processed samples

Return type DataFrame

process(*trace*)

4.4.3 Device API

Abstract Class

class pyJoules.device.**Domain**
Identify a domain, a monitorable sub-part of a device

class pyJoules.device.**Device**
Interface to get energy consumption information about a specific device

static available_domains()
Returns names of the domain that could be monitored on the Device :rtype: List[*Domain*] :return: a list of domain names :raise NoSuchDeviceError: if the device is not available on this machine

configure(*domains=None*)
configure the device to return only the energy consumption of the given energy domain when calling the *pyJoules.device.Device.get_energy()* method

Parameters **domains** (Optional[List[*Domain*]]) – domains to be monitored by the device, if None, all the available domain will be monitored

Raises ***NoSuchDomainError*** – if one given domain could not be monitored on this machine

get_configured_domains()
Get the domains that was passed as argument to the configure function

Returns A list of Domain

Raises ***NotConfiguredDeviceException*** – if the device was not configured

get_energy()
Get the energy consumption of the device since the last device reset

Return type List[float]

Returns a list of each domain power consumption. Value order is the same than the domain order passed as argument to the *pyJoules.device.Device.configure()* method.

RAPL Device Classes

class pyJoules.device.rapl_device.**RaplDevice**
Interface to get energy consumption of CPU domains

static available_core_domains()
return a the list of the available energy Core domains

Return type List[*RaplCoreDomain*]

static available_domains()
return a the list of the available energy domains

Return type List[*RaplDomain*]

static available_dram_domains()
return a the list of the available energy Dram domains

Return type List[*RaplDramDomain*]

static available_package_domains()
return a the list of the available energy Package domains

Return type List[*RaplPackageDomain*]

static available_uncore_domains()
return a the list of the available energy Uncore domains

Return type List[*RaplUncoreDomain*]

configure(*domains=None*)
configure the device to return only the energy consumption of the given energy domain when calling the *pyJoules.device.Device.get_energy()* method

Parameters **domains** – domains to be monitored by the device, if None, all the available domain will be monitored

Raises ***NoSuchDomainError*** – if one given domain could not be monitored on this machine

get_energy()
Get the energy consumption of the device since the last device reset

Returns a list of each domain power consumption. Value order is the same than the domain order passed as argument to the *pyJoules.device.Device.configure()* method.

class pyJoules.device.rapl_device.**RaplDomain**(*socket*)

get_domain_name()

Return type str

Returns domain name without socket identifier

class pyJoules.device.rapl_device.**RaplCoreDomain**(*socket*)

get_domain_name()

Returns domain name without socket identifier

class pyJoules.device.rapl_device.**RaplUncoreDomain**(*socket*)

get_domain_name()

Returns domain name without socket identifier

class pyJoules.device.rapl_device.**RaplDramDomain**(*socket*)

get_domain_name()

Returns domain name without socket identifier

class pyJoules.device.rapl_device.**RaplPackageDomain**(*socket*)

get_domain_name()

Returns domain name without socket identifier

Nvidia GPU Device Classes

class pyJoules.device.nvidia_device.**NvidiaGPUDevice**

Interface to get energy consumption of GPUs

static available_domains()

Returns names of the domain that could be monitored on the Device :rtype: List[*NvidiaGPUDomain*] :return: a list of domain names :raise NoSuchDeviceError: if the device is not available on this machine

configure(*domains=None*)

configure the device to return only the energy consumption of the given energy domain when calling the *pyJoules.device.Device.get_energy()* method

Parameters **domains** (Optional[List[*NvidiaGPUDomain*]]) – domains to be monitored by the device, if None, all the available domain will be monitored

Raises ***NoSuchDomainError*** – if one given domain could not be monitored on this machine

get_energy()

Get the energy consumption of the device since the last device reset

Returns a list of each domain power consumption. Value order is the same than the domain order passed as argument to the *pyJoules.device.Device.configure()* method.

class pyJoules.device.nvidia_device.**NvidiaGPUDomain**(*device_id*)

Exception

exception pyJoules.device.**NotConfiguredDeviceException**

Exception raised when a user call a device method that need the device to be configured on a non configured device

PYTHON MODULE INDEX

p

INDEX

Symbols

A

C

D

E

G

M

N

P

R

S