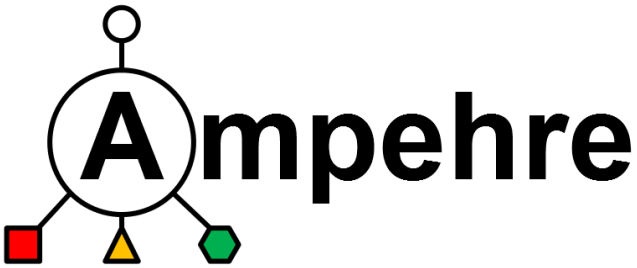

***A**ccurately **M**easuring **P**ower and **E**nergy for **H**eterogeneous **R**esource **E**nvironments*

University of Paderborn, Germany

Ampehre v1.0.0

Authors:
Achim Lösch
Christoph Knorr
Ahmad El-Ali
Alex Wiens

April 4, 2018

Contents

1 Project Description

The Ampehre project is a BSD-licensed modular software framework used to sample various types of sensors embedded in integrated circuits or on circuit boards deployed to servers with a focus to heterogeneous computing. It enables accurate measurements of power, energy, temperature, and device utilization for computing resources such as CPUs (Central Processing Unit), GPUs (Graphics Processing Unit), FPGAs (Field Programmable Gate Array), and MICs (Many Integrated Core) as well as system-wide measuring via IPMI (Intelligent Platform Management Platform). For this, no dedicated measuring equipment such as DMMs (Digital Multimeter) is needed. We have implemented the software in a way that the influence of the measuring procedures running as a multi-threaded CPU task has a minimum impact to the overall CPU load. The modular design of the software facilitates the integration of new resources. Though it has been enabled to integrate new resources since version v0.5.1, the effort to do so is still quite high. Accordingly, our plans for the next releases are broader improvements on the resource integration as well as an extensive project review to stabilize the code base.

Until version v0.8.0 the library used for resource measurement was *libmeasurement*. From version v1.0.0 on alternatively the PAPI library can be used to execute the measurements. For this the wrapper library *libapapi* was created to make the PAPI capabilities available through the well-known interface of *libmeasurement*. By default the tools use *libmeasurement*.

2 Build instructions

Required dependencies:

- CMake
- gcc, g++

Optional dependencies:

- NVML for Nvidia GPU support
- maxeleros for Maxeler FPGA support
- mpss-micmgmt for Intel Xeon Phi support
- cpufrequtils/cpupowerutils
- QT and QWT for GUI tools

Included dependencies:

- PAPI
- cjson

2.1 Cloning

Clone the project to a local directory.

```
git clone --recurse-submodules https://github.com/akiml/ampehre
```

This also clones the PAPI repository as a submodule to the `papi/` folder.

2.2 Building

Ampehre uses CMake as primary build system. The Makefile is used for starting the build, installing and cleaning up. Run `make` to start the build process into the subdirectory `build/`.

The PAPI build process was not changed and is triggered from the CMake build script. The called script can be found in the file `papi/cmake_build.sh`.

Run `make clean` to clean the CMake and PAPI build folders.

Using `make install` you can install Ampehre to your system.

2.3 Adjust build parameters

It might be necessary to adjust certain build parameters. Foremost you can change paths to *gcc* and *g++* inside the file `Makefile`. Next you can deactivate options for certain supported measurement sources at the top of the root `CMakeLists.txt`.

You might have a newer version of cpufrequtils installed named cpupowerutils. In this case adjust the linked library in `libmeasure/cpu_intel_xeon_sandy/CMakeLists.txt` or create a symlink to

`libcpupower.so` called `libcpufreq.so` at the adequate position. However, there is no guarantee for compatibility to newer versions.

In case the paths to optional dependencies for the NVML or Maxeler libraries can't be found, adjust the search paths in the *find external libraries* section of the root CMake file `CMakeLists.txt`.

Note that it is most certainly necessary to call `make clean` after changing build parameters to remove the CMake cache and generated Makefiles.

2.4 Linux Kernel Modules

The *driver_measure* module offers access to mainboard power readings through DELL IDRAC7 IPMI messages. To build and install the *driver_measure* module run `make` and `make install` in `driver/`. There are folders with two versions: the module in `driver` offers IPMI and MSR measurements, while the module in `driver_ipmi` only supports IPMI. The old *libmeasurement* requires the module in `driver`.

The *amsr* module offers access to MSR values used to read energy, temperature and frequency values for Intel CPUs. It is a modified version of the original *msr* kernel module, extended with whitelisting of available registers. The *amsr* device nodes can be made available to normal users since only whitelisted registers are accessible. To add registers to the whitelist modify the arrays at the top of the `amsr.c` file and rebuild. To build the *amsr* module run `make`. Note: *msr_safe* is a similar kernel module supported by the PAPI rapl component.

For both modules, make sure that the respective device files are accessible to users. Adjust accessibility using *chmod*.

3 Usage

Ampehre provides several possibilities to execute measurements.

- Measurement of a complete program execution
- Integration into your own program to measure a certain section
- Live monitoring of the system including a graphical output

3.1 Tools

3.1.1 hettime

hettime allows to measure the execution of a newly started application similar to *time*. For this hettime starts the measurement, executes the program and waits for its completion before stopping the measurement. hettime has several parameters to configure the measurement, but it expects at least the path to the executable. Using `-a` you can pass arguments to the measured program.

```
hettime [OPTIONS] -e EXECUTABLE [-a ARGS]
```

By default measurements for the modules are performed at a sampling rate of 100 ms. To adjust the sampling rate use the following parameters in ms:

```
-c SAMPLE_CPU
-g SAMPLE_GPU
-f SAMPLE_FPGA
-m SAMPLE_MIC
-s SAMPLE_SYS
```

By default hettime uses the libmeasurement library for measurment. To start hettime with the libapapi library use the `LD_PRELOAD` environment variable.

```
LD_PRELOAD=/path/to/libms_common_apapi.so hettime
```

This allows to use additional configuration of the APAPI library.

3.1.2 msmonitor

To do a live monitoring of your system you can use the msmonitor gui tool.

3.1.3 msmonitor_cs

To use msmonitor remotely start the msmonitor server program on the host you want to monitor, the default port is 2900:

```
msmonitor_server [-p PORT]
```

Now you can start the client on a different host to receive the monitored data:

```
msmonitor_client
```

In the settings menu you can define the target address and port to connect to the monitored server.

3.2 Program integration

To integrate the measurement tool in your own program you need to include the `ms_measurement.h` header file and link against the `libms_common.so` library. Now you can make use of the measurement library to measure a certain section of you program. The following listing shows the necessary steps to execute a measurement.

```
// initialize the measurement library using the default parameters
MS_VERSION version = { .major = MS_MAJOR_VERSION,
                       .minor = MS_MINOR_VERSION,
                       .revision = MS_REVISION_VERSION };
MS_SYSTEM *ms = ms_init(&version, CPU_GOVERNOR_ONDEMAND,
                        2000000, 2500000, GPU_FREQUENCY_CUR,
                        IPMI_SET_TIMEOUT, SKIP_PERIODIC,
                        VARIANT_FULL);

// allocate the data structure for one measurement
MS_LIST *m1 = ms_alloc_measurement(ms);

// Set timer for m1
// Measurements perform every (10ms/30ms)
ms_set_timer(m1, CPU,     0, 10000000, 10);
ms_set_timer(m1, GPU,     0, 10000000, 10);
ms_set_timer(m1, FPGA,    0, 30000000, 10);
ms_set_timer(m1, SYSTEM,  0, 30000000, 10);
ms_set_timer(m1, MIC,     0, 30000000, 10);

// setup the m1 data structure for the next measurement
ms_init_measurement(ms, m1, CPU | GPU | FPGA | SYSTEM | MIC);

// measure the section you want to analyze
ms_start_measurement(ms);

do_something();

ms_stop_measurement(ms);
ms_join_measurement(ms);
ms_fini_measurement(ms);

// now you can look at the results

// destroy the measurement data structure
ms_free_measurement(m1);

// finally shutdown the measurement library
ms_fini(ms);
```

To analyze the measurement results use the methods in the respective module headers.

```
// method for dram energy consumption
// defined in the ms_cpu_intel_xeon_sandy.h header
printf("consumed energy of cpu 1 dram bank : %.2lf mWs\n",
           cpu_energy_total_dram(m, 1));
```

3.3 APAPI configuration

On invocation libapapi reads certain environment variables for configuration. This might be especially handy when using hettime with libapapi. The detailed file formats are described in appendix A.

- `APAPI_CMPLIST` allows to define which components are measured. The default is `rapl:nvml:maxeler:micknc:ipmi`
- `APAPI_EVENTLIST` allows to define the list of events per component that are measured. The default list `default_eventlist.txt` is included.
- `APAPI_EVENTOPS` allows to define how the events are processed. This is only needed when adding new events. The default list `default_eventops.txt` is included.

4 Extend measurement capabilities

To add additional indicators you need to modify several files:

- PAPI component
- libapapi event definitions
- libapapi value copying
- libmeasurement data structures and methods

4.1 PAPI component

The PAPI library supports extension by defining a new component. The components can be found in the `papi/src/components` directory and every component has its own folder. If you want to create a completely new component you can find an example component in the `example` folder. Every component creates an internal list of available events on initialization in `_example_init_component`. `_example_update_control_state` is called to maintain a struct that contains the chosen events. If the read method `_example_read` is called the values of chosen events need to be read and copied to the output array. In all of this steps the different events need to be handled and modified in case of extension, however available components contain additional support methods.

If a new component is created it needs to be added to the list of compiled components `--with-components` in the build script `papi/cmake_build.sh`.

4.2 libapapi event definitions

Once the component is available in PAPI the event definitions for libapapi must be adjusted. The detailed file formats are described in appendix A.

The file defined in the environment variable `APAPI_EVENTLIST` contains the list of events to be considered if a component is activated. Simply add the new events to the list similar to the default file. Since the specified file will override the default file you also need to copy the desired events from the default file into the specified eventlist file.

The file defined in the environment variable `APAPI_EVENTOPS` contains a list of parameters that specifiy how to handle event values. Accumulating values are treated differently. Add a new entry per event according to the specifications in the appendix. Your specified file will be merged with the default definitions so no copying of default definitions is necessary, except to override a default definition.

Finally, to activate the component you need to declare the list of used components in the `APAPI_CMPLIST` environment variable.

4.3 libmeasurement data structures and methods

For every libmeasurement component exists a struct containing the desired variables and the accessing methods.

4.4 libapapi value copying

To copy the measured event values to the libmeasurement structs the `libms_common_apapi` wrapper copies the values from the internal PAPI arrays into the target data structures. To minimize the overhead of event traversing a list of mappings between the PAPI arrays and the libmeasurement structs are created in the method `__create_mapper`. Follow the pattern of the existing entries of checking the current component and existence of the event and finally defining the four entries for the mapping:

- `source` – pointer to the PAPI array entry
- `destination` – pointer to the struct variable
- `type` – type conversion specifier
- `factor` – factor to convert value to the destination prefix

For struct variables that only need to be changed at the end of the measurement, check the `last_measurement` variable.

Appendices

A libapapi definition file specification

A.1 eventlist file

Every line is interpreted as one event name. Lines starting with '#' are ignored. Empty lines are ignored.

Example:

```
# rapl
rapl:::PACKAGE_ENERGY:PACKAGE0
rapl:::PACKAGE_ENERGY:PACKAGE1
rapl:::DRAM_ENERGY:PACKAGE0
rapl:::DRAM_ENERGY:PACKAGE1
```

If the `APAPI_EVENTLIST` environment variable is defined and the rapl component is active, only the four specified events are measured.

A.2 eventops file

Every line is interpreted as one event definition. Lines starting with '#' are ignored. Empty lines are ignored. One definition consists of several fields delimited by a ','. Fields should not contain unnecessary whitespace. One definition consists of the following fields:

- event name
- operation to use to compute value1 from value0 (last two values for value0 are called sample1 (last) and sample0 (previous))
  - `APAPI_OP1_SAMPLE_DIFF`
    - value1 = sample1 - sample0
  - `APAPI_OP1_SAMPLE1_MUL_DIFF_TIME`
    - value1 = sample1 * (time1 - time0)
  - `APAPI_OP1_AVG_SAMPLE_MUL_DIFF_TIME`
    - value1 = (sample0 + sample1) / 2.0 * (time1 - time0)
  - `APAPI_OP1_DIV_DIFF_TIME`
    - value1 = sample1 / (time1 - time0)
- statistics to compute for value0, single term or '—' delimited list of terms:
  - `APAPI_STAT_NO` - no statistics
  - `APAPI_STAT_MIN`
  - `APAPI_STAT_MAX`
  - `APAPI_STAT_AVG`
  - `APAPI_STAT_ACC`

  - APAPI_STAT_ALL

- statistics to compute for value1
- maximal raw counter value or 0 if counter is not accumulating
- name for type of value0
- unit for value0
- factor to use for value0
- name for type of value1
- unit for value1
- factor to use for value1

Example:

```
# this event is an accumulating event
# the maximal raw counter value is not zero
rapl:::PP0_ENERGY:PACKAGE1,APAPI_OP1_DIV_DIFF_TIME,APAPI_STAT_ACC,
APAPI_STAT_MIN|APAPI_STAT_MAX|APAPI_STAT_AVG,0x7fffffff8000,
energy,Ws,2147483648.0,power,W,2.147483648

# this event is not-accumulating
# the maximal raw counter value is zero
rapl:::THERM_STATUS:PACKAGE0:CPU0,APAPI_OP1_NOP,
APAPI_STAT_MIN|APAPI_STAT_MAX|APAPI_STAT_AVG,APAPI_STAT_NO,
0,temperature,C,1,-,-,1
```

Note: The line breaks in the example are due to the document layout. The original entries must not contain line breaks.

Entries from the default file and the file declared in the APAPI_EVENTOPS environment variable are merged. Entries in the user specified file override entries in the default file.