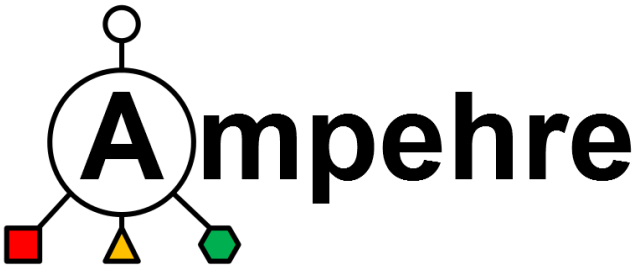

***A**ccurately **M**easuring **P**ower and **E**nergy for **H**eterogeneous **R**esource **E**nvironments*

University of Paderborn, Germany

Ampehre v0.8.0

Authors:
Achim Lösch
Ahmad El-Ali

August 3, 2017

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. Version 0.8.0 features a client/server implementation, so that the measurement accuracy increased.

2 User Manual

MSMonitor is a tool to measure and monitor the power consumption as well as the temperature of hetorgeneous nodes. This version features a client/server implementation, which enables a more precise and efficient measurement. In the following the client and server options are to be explained.

2.1 Server

The server is supposed to run on the heterogenous node itself, in order to measure the needed information. It awaits incoming requests and responds to them, so the server program is a pure command line application with the following two options:

Option	**Description**
-p	specifies the port which the server listens to for any requests. This port has to be equally chosen on the client
-d	enables a debug mode with verbose output

The server is capable of supplying up to 5 clients simultaneously. If the maximum amount of clients is already being served, new incoming connections will be queued until an active client logs off or times out.

2.2 Client

The Client is responsible for presenting the data in a graphical form. This takes away the load from the server, making the measured values more precise.
Just as the server, the client can be passed the same command line options.
After starting the client, a graphical user interface, made with Qt, is loaded. Initially it has got a big and empty mdi area in the center as well as a menu bar at the top (1).

The settings window and all the plots can be opened from the menu bar or via keyboard shortcuts. They will appear on the mdi area, where they can be moved around according to the user's preference.
The most important window is the settings window, which contains all controls to connect to the server, set up the gui refresh rates and data frequency. This can be seen in 2. Tooltips are available for each setting, in order to receive further information.
The settings can be exported to a configuration file, which can also be loaded on the next session, sothat the user does not have to redo the settings each time after starting the program. Directly after connecting to the server, the plots will be drawn.
A plot consists of two tabs. The first tab (plot) contains the coordinate system itself, a legend, a spin box to choose the thickness of the graphs, a screenshot button to export the current graph to png and a csv export button to export the values into a highly manageable comma separated format. This can be seen in 3.
The second tab (settings), as shown in 4, gives the options to remove currently unneeded graphs by simply clicking on the corresponding labels. Recorded

Figure 1: starting user interface of MSMonitor

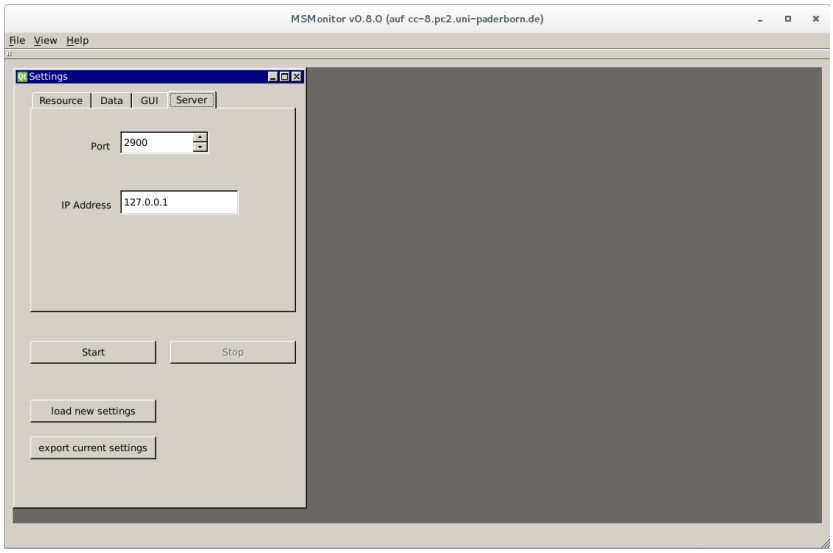

Figure 2: user interface after enabling the settings window

global minimum and maximum values are listed next to the labels.
If the graphs have high fluctuation, median and mean filter can be applied over an interval of the user's choice, in order to clarify the tendency of the graph. If the chosen interval is higher than the actual amount of values, nothing will be drawn until the amount of recorded data exceeds the chosen interval. Besides the option of drawing graphs, start and termination of specific programs can be symbolized on the graphs, by informing the server using the signals SIGUSR1

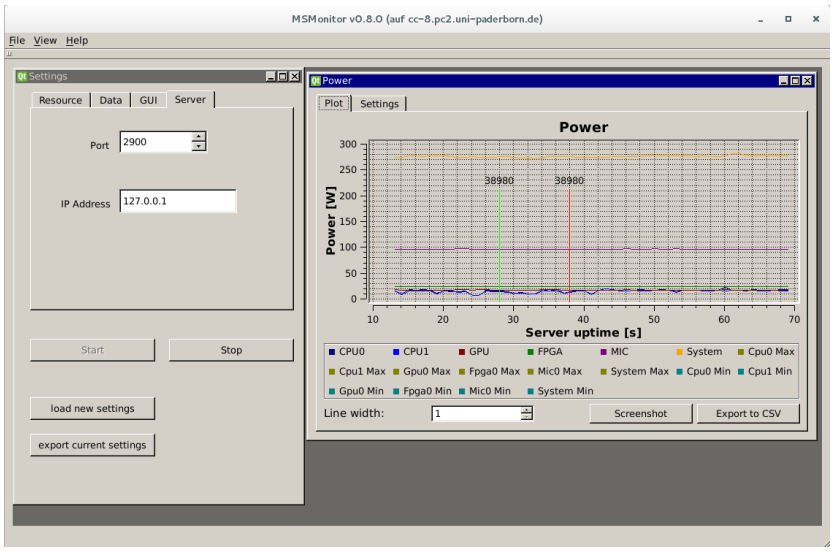

Figure 3: an active graph in msmonitor

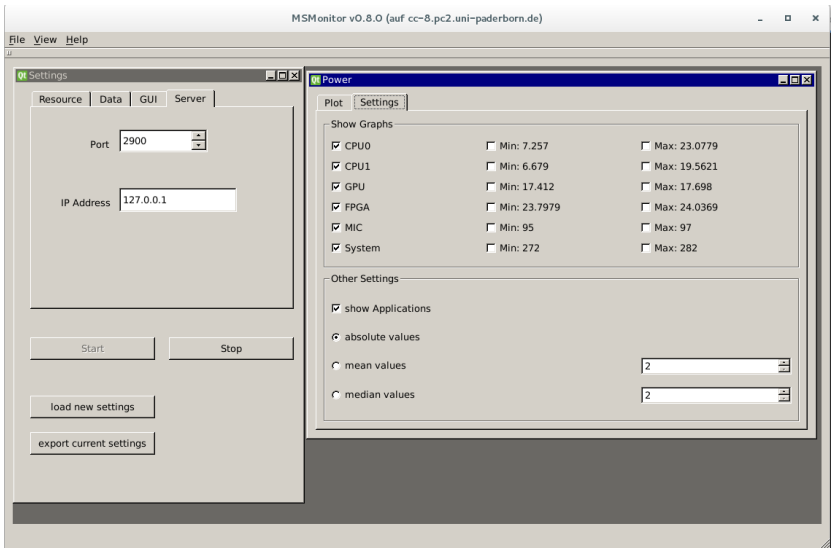

Figure 4: the second tab of a plot

and SIGUSR2. If an application sends a SIGUSR1 prior to starting, the server informs the clients about a newly started application. Sending a SIGUSR2 to the server, just before terminating the application, makes it inform the clients about a finished application. The unique pid ensures the exclusion of any mix-ups.

3 Protocol

In order to enable a stable communication between client and server, a protocol had to be defined. We decided to use tcp and ipv4. The structure of our protocol is based on http.

3.1 Message Types

An important part of a message is the type, so that server and client know how to deal with the incoming information. The following message types exist:

Option	Description
CLIENT_REG	The first contact is initiated by the client. This is an attempt to register to the server. Skipping this step leads to the server refusing to answer. If the maximum amount of clients is not reached yet, this should be successful.
SET_FREQ	The client receives the sampling rates from the heterogeneous node.
DATA_REQ	After the registration, the client can request measured data from the server.
DATA_RES	The server responds to a data request. The assignment of the data, which seem to be random numbers, to the corresponding value happens threw an equally defined enumeration on the server and the client.
TERM_COM	The communication is terminated from either side.

3.2 Message Structure

A message consists of a header, which contains the type and protocol version. The second part is the main body, which contains the needed information.

3.2.1 Header

This is an example for a header:

```
1 CLIENT_REG / MSMP/0.1
```

Listing 1: A MSMonitor Protocol Header

It consists of the message type, followed by the protocol name and version. The type gives the client and server information about how to interpret the following message body.
Please notice the resemblance to the highly known HTTP header:

```
1 GET /file.html HTTP/1.1
```

Listing 2: A HTTP Header

3.2.2 Body

The main body of the message highly depends on its type. So possible messages might be:

```
CLIENT_REG / MSMP/0.1
11010000000000000100000000000000000000000000000000000000000000000000
```

Listing 3: client registers to server

The value of datatype long, indicates which data is being requested. Each Bit set to '1' means, that the corresponding value is needed and '0', that it is not. The server will only send back those values, which are requested by the client. In this case it would be only four values.

```
CLIENT_REG / MSMP/0.1
REG: 0
```

Listing 4: server registers client

The server accepts the client and registers it with the unique id '0'. Now the client can request data by mentioning only the given id.

```
DATA_REQ / MSMP/0.1
REG: 0
```

Listing 5: client requests data

As the server knows about the client due to the registry in the beginning of the communication phase, the client can now request data by using its id only.

```
DATA_RES / MSMP/0.1
201.58
365
17
23
```

Listing 6: server responds with needed data

The server responds with the requested data in a defined order, which is equally known by server and client.

```
TERM_COM / MSMP/0.1
REG: 0
```

Listing 7: client terminates communication

After sending this message, the server knows exactly which client is logging of and erases it from the list in order to free space for other incoming clients.

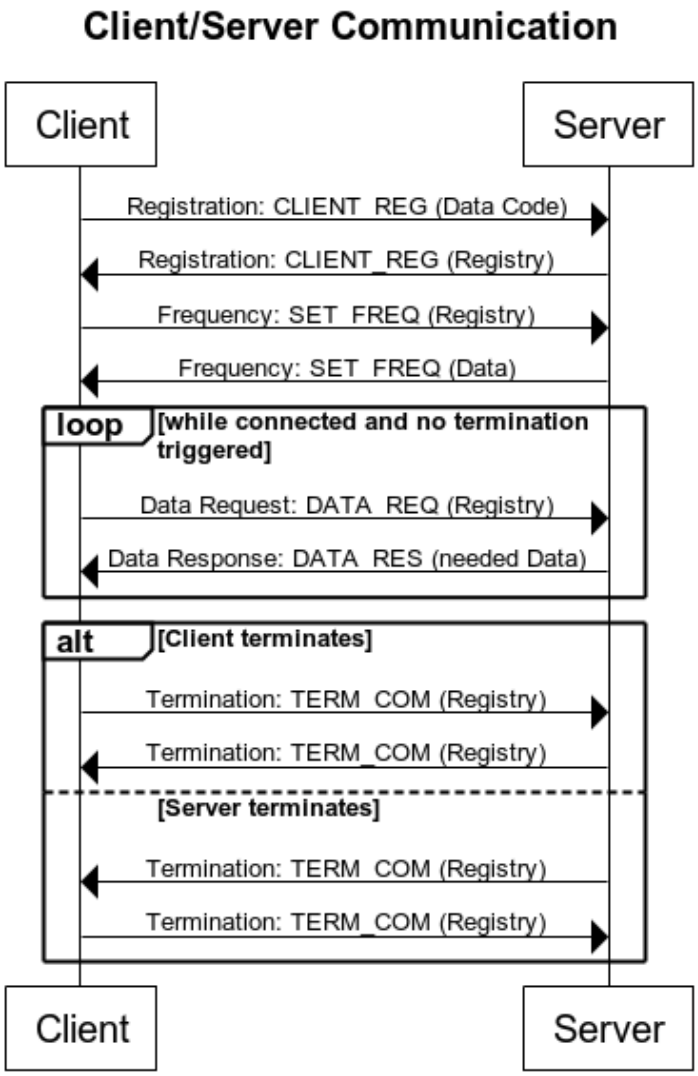

Figure 5: protocol flow diagram

3.3 Communication

The flow diagram (5) shows a typical sequence of events, forming a communication between server and client.

First of all the client registers to the server, transmitting a value of datatype long. This value contains the information about which data is needed and which is not. The server answers with a unique registry value.
Afterwards the sampling rates of the hardware are transmitted to the client.
Then the data needed for the graphs is being requested by the client as long as necessary.
The connection is finally terminated by either the client or the server.

Appendices

A Recommended Sampling Rates

Users must specify a sampling rate for each resource which is compiled as libmeasure module. The sampling rate defines how often measurement values are queried from the devices. Low sampling rates can produce substantial CPU load, since all the measurement threads are executed on the CPU. Hence, sampling rates have to be chosen carefully. Moreover, they have an impact on the accuracy of the measurement results and the CPU utilization. We have to find a trade-off between accuracy of the measurements and the CPU utilization which also leads to different CPU power consumption. Therefore we have methodically examined different sampling rate combinations using all five modules runnable on our heterogeneous system. We have used libmeasure in its `VARIANT_FULL` (all sensors are sampled) with high accuracy (`skip_ms_rate` set to `SKIP_NEVER`). We have measured the power consumption and utilization while all resources have been in idle state. The results are shown in Figure 6 and 7. Obviously, the CPU utilization and power consumption is highly dependent on specific system configurations. We hope that our results are helpful anyway.

Figure 6 shows the CPU utilization, sampling all sensors of all currently supported resources as specified in Section **??**. The lines indicate our recommendations to achieve utilizations below a specific thresholds. For example, the blue line indicates that the utilization induced by our measuring library loading all resource-specifc modules stays below 2 %, if the sampling rates CPU: 40 ms, MIC: 50 ms, GPU: 40 ms, FPGA: 70 ms and System 100 ms or higher are used for the measurements.

Figure 7 shows the resulting CPU power consumption induced by sampling all sensors of all currently supported resources as specified in Section **??**. Accordingly, the lines indicate what sampling rates have to be chosen to make sure that the CPU power consumption stays below specific thresholds. For example the sampling rates have to be CPU: 20 ms, MIC: 20 ms, GPU 30 ms, FPGA 40 ms, System 80 ms or higher to get a CPU power consumption of less than 18 W.

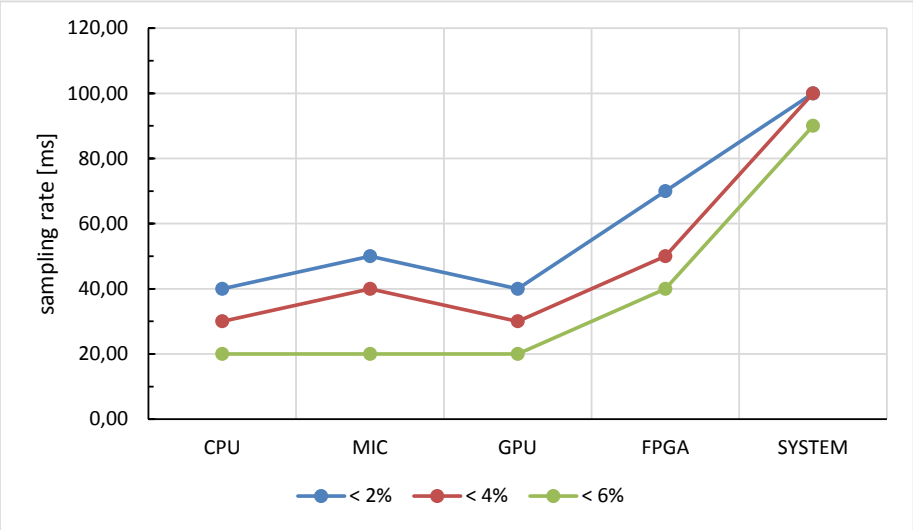

Figure 6: Libmeasure sampling rates and the resulting CPU utilization in percent. The lines indicate the lower boundaries of the sampling rates for which the CPU utilization is not higher than the corresponding threshold (2 %, 4 %, 6 %).

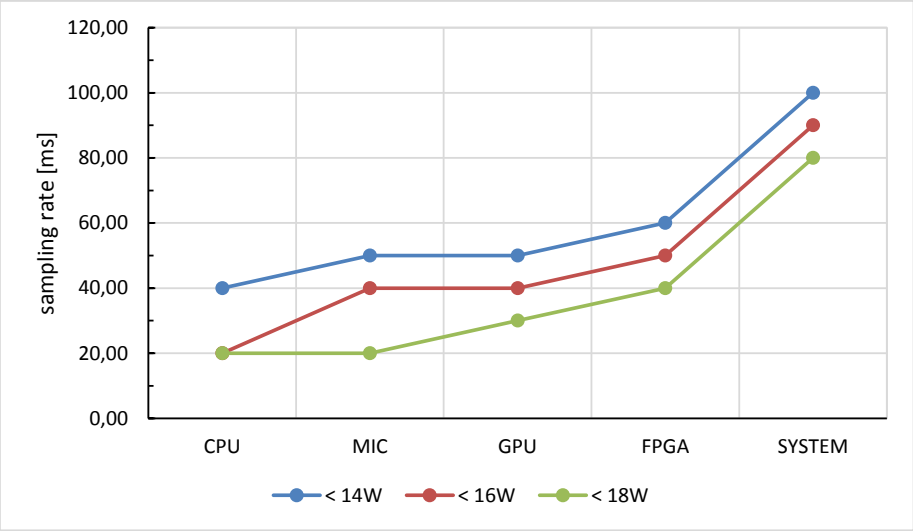

Figure 7: Libmeasure sampling rates and the resulting CPU Power consumption in Watt. The lines indicate the lower boundaries for which the CPU power consumption is not higher than the corresponding threshold (14 W, 16 W, 18 W).