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

UNIVERSITY OF PADERBORN, GERMANY

Ampehre v0.8.0

Client-Server Implementation of MSMonitor

Authors:
Achim LÖSCH
Ahmad EL-ALI

August 18, 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 multithreaded 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, which moves the graphical rendering as well as other computations from the heterogeneous node to a client.

2 User Manual

MSMonitor is a tool to measure and monitor the power consumption, temperature, utilization and memory usage of heterogeneous nodes. Our new version features a client/server implementation, which enables a more precise measurement with less overload by moving high computational effort e.g. graphical rendering, to a connected client. In the following the client and server options are to be explained.

2.1 Server

The server is supposed to run on the heterogeneous node itself, in order to measure the needed information. It awaits incoming clients and responds to them by sending back all requested data. 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 client requests.
-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 and thus takes care of the rendering and other additional calculations. This reduces the load on the server, leading to more precise measurements compared to the standalone implementation.
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. A big and empty mdi area is in the center while a menu bar at the top (Figure 1).
The settings window and all the plots can be opened from the menu bar or via keyboard shortcuts. They appear on the mdi area, where users can move them around.
The settings window contains all controls to connect to the server, set up the GUI refresh rate and data frequency. This can be seen in Figure 2. The data frequency sets the interval of time, in which new data is being requested from the server. The Save Data spinbox sets the amount of values to be drawn. Setting this to 1 would only draw a point. If "Save Data" is set to 100 and the amount of available data exceeds 100, then the oldest data will be erased until the amount of available data is less or equal 100.
The GUI refresh rate sets the interval of time, in which the graphs are being redrawn, no matter if new data is already available or not.
Tooltips are available for each setting, in order to receive further information.
In order to connect to the server, its IP address and port need to be given. In case a firewall is set up on the heterogeneous node, a direct connection is

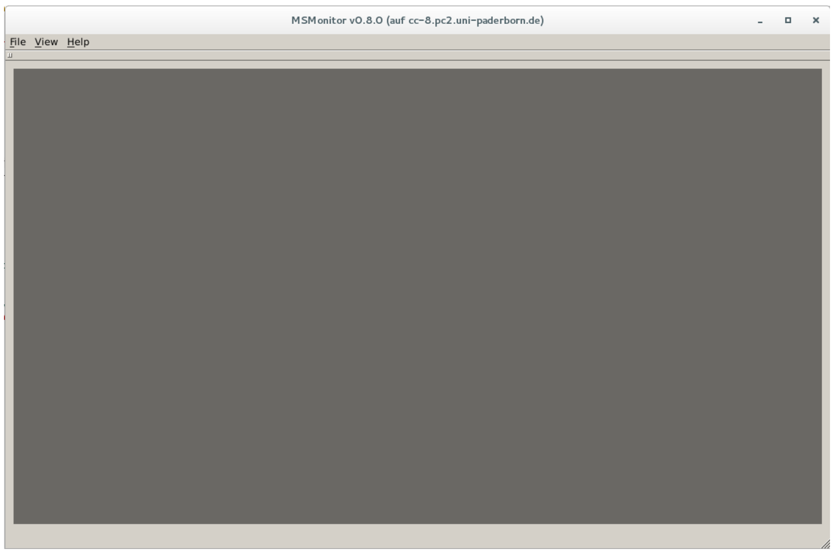


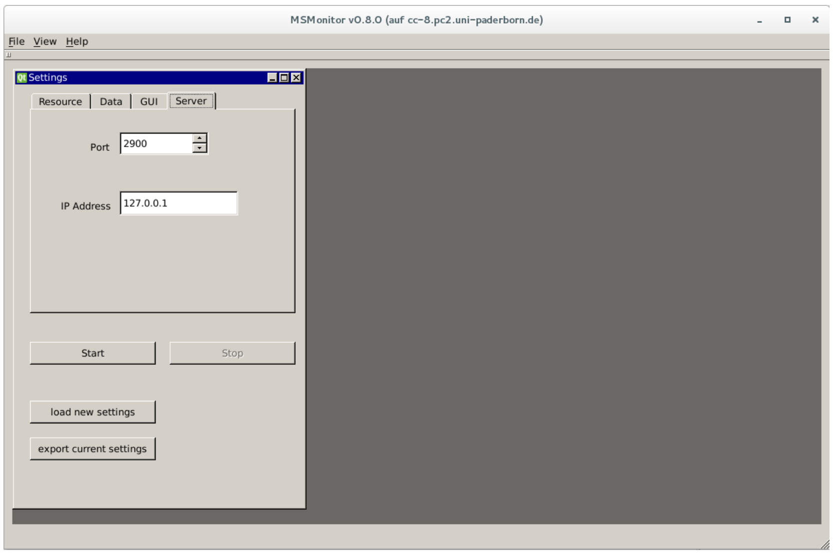
Figure 1: Starting user interface of MSMonitor

prohibited. In this case a ssh tunnel can be used as a workaround.
The packages needed to successfully compile the client are `Qwt 6.1.3` and `Qt4` or later. After compiling the client, run the client script to set up the tunnel:

```
git clone https://github.com/akiml/ampehre.git
cd ampehre
make client
./msmonitor_client.sh "<IP_ADDRESS>" <PORT>
```

Listing 1: install the client

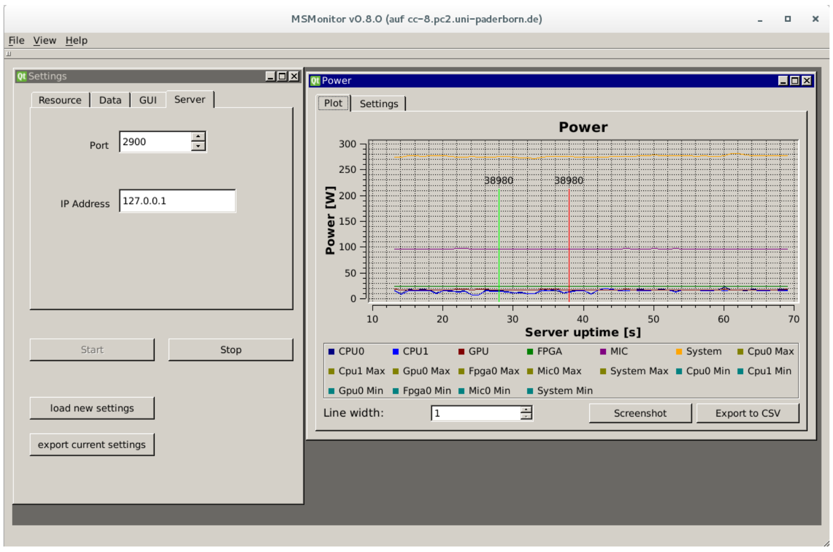
The first argument is the IP address of the server and the second argument is the port number.
This has been tested on Ubuntu 16.04 and on Debian 9.
The settings can be exported to a configuration file, which can be loaded on the next session. After connecting to the server, the plots will be drawn.
Each consists of two tabs. The first tab (plot) contains the coordinate system itself, a legend, a spin box to choose the thickness of the curves , 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 Figure 3. The second tab (settings), shown in Figure 4, gives the option to enable or disable currently drawn graphs by simply clicking on the corresponding labels. Recorded global minimum and maximum values are listed next to the labels and can also be shown inside the graph.
If the curves have high fluctuation, median and mean filter can be applied over a user-specified interval, in order to clarify tendencies. Please note, if the chosen interval is higher than the actual amount of values, nothing will be plotted 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 sending UNIX signals SIGUSR1 and SIGUSR2 to the server.



Port 2900

IP Address 127.0.0.1

Figure 2: User interface after enabling the settings window



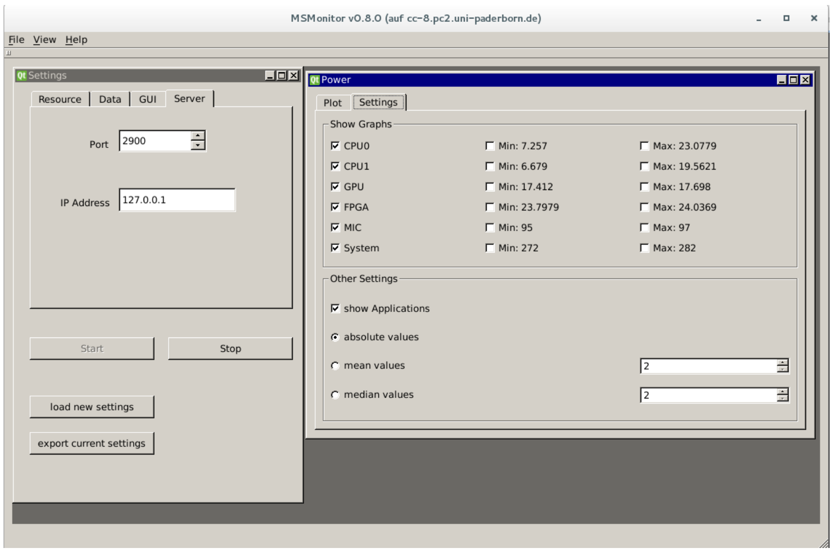
Port 2900

IP Address 127.0.0.1

Line width: 1

Figure 3: Active graph in msmonitor (right)

If SIGUSR1 is signaled to the server prior to starting, it informs the clients about a newly started application. Sending SIGUSR2, informs all registered clients about the termination of an application. The unique process ID ensures that the user can identify the lifetime of an application.
Besides the mentioned graphs and curves there are also other forms of visualization available. The heatmaps give a feedback using a defined color-scale. For example high temperature will be presented in red and low temperature in blue.



Port: 2900

IP Address: 127.0.0.1

Show Graphs

☑ CPU0	☐ Min: 7.257	☐ Max: 23.0779
☑ CPU1	☐ Min: 6.679	☐ Max: 19.5621
☑ GPU	☐ Min: 17.412	☐ Max: 17.698
☑ FPGA	☐ Min: 23.7979	☐ Max: 24.0369
☑ MIC	☐ Min: 95	☐ Max: 97
☑ System	☐ Min: 272	☐ Max: 282

Other Settings

☑ show Applications

[✓ absolute values]

mean values: 2

median values: 2

Figure 4: Second tab of a plot (right)

The system overview gives a summary about current processes on the heterogeneous node and gives an overview over most of the currently measured data.

3 Protocol

In order to enable communication between client and server, a protocol had to be defined. Our protocol is based on `http`. Connections are established via TCP/IPv4 network protocol.

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 are currently implemented:

Option	**Description**
`CLIENT_REG`	The first contact is initiated by the client. This is an attempt to register a client to the server. Skipping this step leads to the server refusing to answer.
`SET_FREQ`	The client receives the sampling frequencies for each computing resource 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 specific data request.
`TERM_COM`	The communication is terminated from either side.

3.2 Communication

The flow diagram (Figure 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 frequencies 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.

3.3 Message Structure

A message consists of a header, which contains the message type and protocol version. The second part is the body, which contains the message. A tail is not defined.

3.3.1 Header

The following shows the header for a client registration:

```
CLIENT_REG / MSMP/0.1
```

Listing 2: A MSMonitor Protocol Header

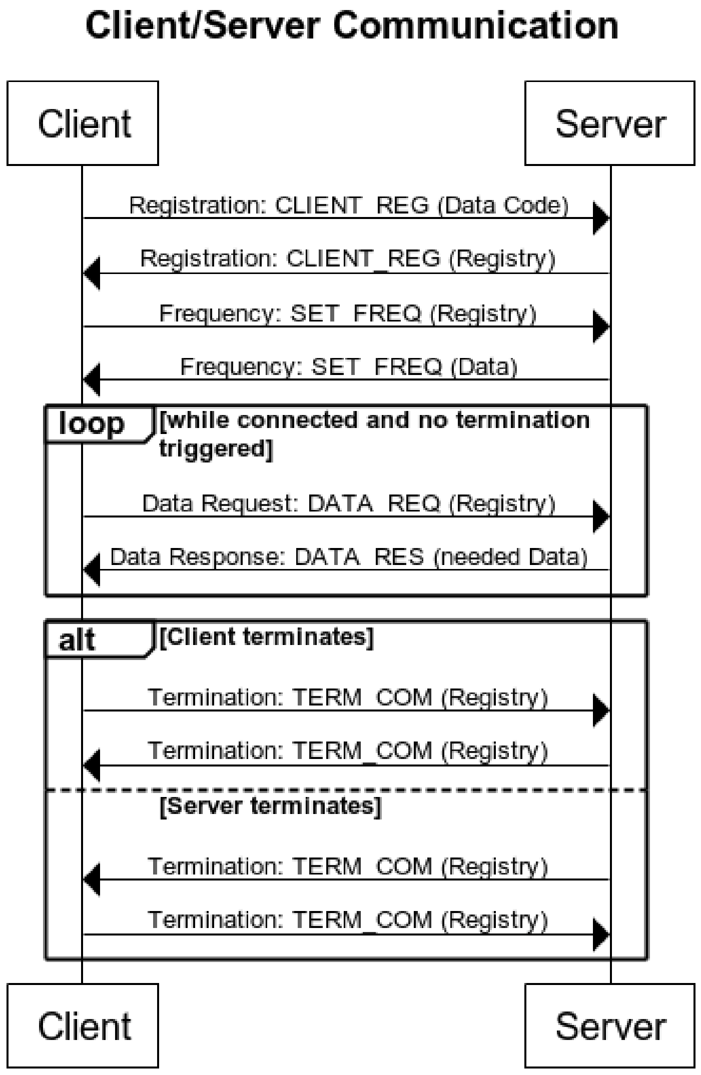

Client/Server Communication
Client
Server
Registration: CLIENT_REG (Data Code)
Registration: CLIENT_REG (Registry)
Frequency: SET_FREQ (Registry)
Frequency: SET_FREQ (Data)
loop
[while connected and no termination triggered]
Data Request: DATA_REQ (Registry)
Data Response: DATA_RES (needed Data)
alt
[Client terminates]
Termination: TERM_COM (Registry)
Termination: TERM_COM (Registry)
[Server terminates]
Termination: TERM_COM (Registry)
Termination: TERM_COM (Registry)
Client
Server

Figure 5: Protocol flow diagram

It consists of the message type, followed by the protocol name and version. In general message types give the client and server information about how to interpret the message body of the current message.
Please notice the resemblance to the widely known HTTP header:

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

Listing 3: A HTTP Header

3.3.2 Body

The message body generally highly depends on its type. This is the message body of the CLIENT_REG message.

```
CLIENT_REG / MSMP/0.1
0b1101000000000000010000000000000000000000000000000000000000000000
```

Listing 4: 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. Please note, that it is a long value which is being transmitted. It is presented human readable in order to clarify the principle.
Please refer to *Protocol.c* in order to understand the bitfield.

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

Listing 5: Server registers client

The server accepts the client and registers it with the unique id '0'. Now the client can request data by transferring its unique registration ID.

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

Listing 6: Client requests data

As the client is registered to the server, the server sends data to the client according to the data request bitfield mentioned in Listing 4.

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

Listing 7: Server responds with needed data

The server responds with the requested data in a defined order, which is equally known by server and client. The data is of datatype double. Please note, the message is usually not human readable.
The DATA_REQ and DATA_RES messages are sent/received in a loop until one of the subscribers terminates the communication by sending a TERM_COM message and disconnects.

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

Listing 8: Client terminates communication