A. Summary

- In the area of ***Evaluation and Exploration of Next Generation Systems for Applicability and Performance,*** over the period of 07/01/11 through 09/30/11 the NCSA Innovative Systems Lab team worked on two projects: development of the software infrastructure for power monitoring on GPU HPC clusters and a reference design and implementation of pattern matching algorithm.

B. Evaluation and Exploration of Next Generation Systems for Applicability and Performance (Volodymyr Kindratenko, Guochun Shi)

1 Software infrastructure for power monitoring

To quantify the energy usage by the applications, we use a power monitoring system developer earlier in our lab (described in 2010 Q4 report). Even though the hardware has been available, up until now we have not fully implemented the software stack necessary to utilize the capabilities of this system. Our goal for the remainder of the project is to completely re-design and re-implement the software to support seamless integration of the power monitoring hardware in production HPC environments, particularly those using GPUs as stand-alone computing units. Figure 1 shown the architecture of the software under development.

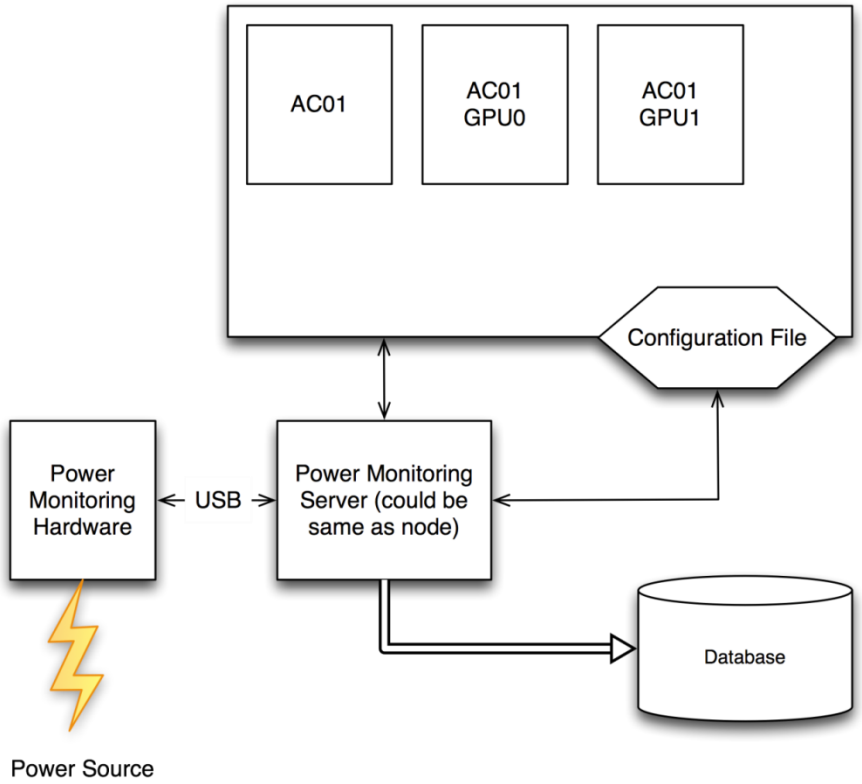

Figure 1. Power monitoring software infrastructure.

1.1 Functional Overview of Power Monitoring

Each group (and all of its associated hardware, GPUs, etc.) has a unique configuration file which contains what devices are being monitored by the power monitoring hardware; what computer/server is connected to the power monitoring hardware (and therefore able to read data from it).

When a job is run on a node, the program may call an API to begin power monitoring. There are four calls in the API:

- Start
- Stop
- Open mark
- Close mark

The (Start) call begins power monitoring. At this point, the API reads the configuration file for the group, and determines using that configuration which server contains the power monitoring hardware. A start signal is sent to that location, and the server begins to poll the hardware for amperage measurements. Each power monitoring box has several channels available to measure. The configuration file describes what is connected to each channel.

For each (Start) call, a unique id is generate to store the results of this session's power monitoring data, in order to facilitate future retrieval. The user does not need to know what this unique id is – it is an internal identifier. Instead, a user can use his job id, node name, and/or the time his program was run to find his associated power monitoring data.

The user may choose to call (open mark) and (close mark) API from his program. These markers (which have names, not necessarily unique) indicate significant points of time, such as the beginning or ending of a loop, et cetera. Although the terminology of these markers suggests behavior similar to parenthesis, the markers need not be nested.

Finally a (Stop) call is made at the end of a program, and at this point no more data is collected from the power monitoring hardware. Should the program abruptly terminate, there is a mechanism for preventing the power monitoring from endlessly continuing.

Once the data has been collected and stored in the database, it can be retrieved in an interactive graphical form from a display server. The display server can be located anywhere on the network so long as it has access to the database.

1.2 Overview of Power Monitoring Server

The power monitoring server has two functions:

- to read data from the connected power monitoring hardware, and
- to handle requests from the power monitoring API

It will coordinate these two functions and upload the resulting data collected to an external database.

One thread is created for each power monitoring box connected to the server (one per USB port). One thread created for a TCP server, and another for a UDP server – the reason for having both will be elaborated.

As the power monitoring server may be the same node on which the monitored program is run, it is important to design the power monitoring server to use as little resources as possible. Most importantly, the server must operate asynchronously, and must avoid blocking I/O.

To this end, a UDP server is created for processing (open mark) and (close mark) requests. UDP incurs very little overhead. Important calls like (start) and (stop), which will only ever be called once in a program's lifetime, are relegated to the TCP server, to ensure that they are not missed by accident.

To address the issue of programs exiting without sending a (stop) signal, there is a keep-alive interval at which the API must send (ping) signals to the server. Should a program die, it will no longer send a (ping) signal and when the interval elapses, the server will automatically stop the data collection.

2 Acceleration of pattern matching

Pattern matching algorithms based on regular expression matching are key in searching text files and databases, deep network packets analysis, computer virus scanners, and bioinformatics applications, to name a few areas of potential impact. Existing CPU-based implementations run at MB/sec data rates. Their port to a GPU-based platform, if successful, can run in GB/sec data rates range.

We have implemented a brute force pattern matching algorithm that performs a sequential search of user-specified patterns in a text file. The implementation serves as a reference design for other, more efficient implementations, but it is not a good candidate for GPU port by itself as it is not efficient.

The technique that we think is appropriate for GPU implementation is the one based on the Deterministic Finite State Machine (FSM). The idea is to pre-compute State Transition Table (STT) and state-pattern (output) table (SPT) for an array of patterns to be searched for and to use this information during the actual search. This approach has a relatively expensive startup time – pre-computation of STT and SPT tables – but once these tables are available, pattern matching complexity becomes O(n) where n is the size of the text file to be searched. Knuth-Morris-Pratt algorithm is one instance of this approach which we are using as the base for our implementation. We are now in the process of implementing the code for generating STT and SPT tables on the host CPU. Once such tables are available, a GPU-based implementation of the actual search algorithm will follow.

3 Future Work

For the remainder of the project term, we plan to finish the two projects currently under development. We will release the power monitoring software solution under NCSA-UIUC Open Source license.