Towards Application Energy Measurement and Modelling Tool Support

Kenneth O'Brien[1(✉)], Alexey Lastovetsky[1], Ilia Pietri[2], and Rizos Sakellariou[2]

[1] Heterogeneous Computing Laboratory, School of Computer Science and Informatics, University College Dublin, Dublin, Ireland
kenneth.obrien@ucdconnect.ie, alexey.lastovetsky@ucd.ie
http://hcl.ucd.ie
[2] The University of Manchester, Manchester, UK

Abstract. We present a prototype toolkit for researchers to accurately measure and model their application's power and energy usage. We provide an analysis of a matrix multiplication application using our api *libhclenergy*.

Keywords: Energy efficiency · Energy measurement · Software tools · High performance computing

1 Introduction

Energy has emerged as a new finite resource that must be considered by application developers. Currently, developers optimise their applications for performance by making the most efficient use of processor clock cycles, memory hierarchies and network bandwidth, in order to reduce execution time.

In recent years Dennard scaling has ended. Dennard scaling was a law that stated as transistor sizes decreased, the power a processor constructed of these transistors requires, remained proportional to the area of that chip.

The practicalities of this breakdown are that processor manufacturers cannot develop processors consisting of smaller transistors without drawing more power and producing more heat. These factors have resulted in the stagnation of processor clock frequencies and the rise of multicore and accelerator computing, as performance improvements can be achieved by adding more cores without shrinking transistors. This is not a complete solution as increasing the number of cores in a processor increases the overall power consumption, more so than what could be achieved if Dennard scaling had held.

This research is supported by the Structured PhD in Simulation Science which is funded by the Programme for Research in Third Level Institutions (PRTLI) Cycle 5 and co-funded by the European Regional Development Fund. This work is partially supported by EU under the COST Program Action IC1305: Network for Sustainable Ultrascale Computing (NESUS).

V. Malyshkin (Ed.): PaCT 2015, LNCS 9251, pp. 91–101, 2015.
DOI: 10.1007/978-3-319-21909-7_10

Our background is in application performance optimisation, therefore we focus our efforts to application power monitoring and analysis. Efforts at the hardware and data center configuration levels are beyond the scope of our study.

In order to optimise for this new resource we require tool support similar to that which exists for performance optimisation. We first need the ability to measure an application's energy usage, as well as the application's power usage, which is the rate of consumption over time.

Advanced mathematical modelling methods and tool support exists for applications with respect to performance [9]. For application energy modelling to develop further, we require equally advanced tool support.

In Sect. 2 we introduce our api for application power and energy monitoring. In Sect. 3 we introduce a tool for power and energy model construction. In Sect. 4, we demonstrate the capabilities of our tools. In Sect. 5 we describe related efforts and in Sect. 6 we conclude and describe our future works.

2 *Libhclenergy*

2.1 Measurement Infrastructure

There are two application metrics we wish to study, power and energy. Power is a rate of consumption of electricity, measured in units of Watts. Energy is a measure of work done. It is measured in Watt Hours (Wh) or Watt Seconds, commonly known as Joules (J). Power and energy are related by Eq. 1, where E is energy in Joules, P is power in Watts and t is time in seconds.

$$E = P * t \tag{1}$$

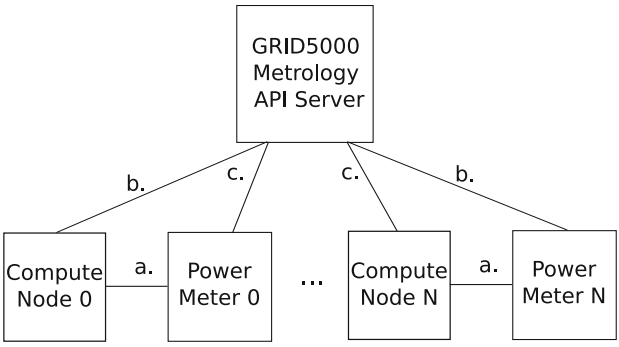

Fig. 1. Power measurement infrastructure

For our measurements we assume the existence of an external measurement device that records the power consumption of our node at regular intervals without affecting the nodes consumption. Figure 1 shows the mechanism by which we obtain our data.

Our *Compute Node 0* is instrumented by an external power measurement device (*Power Meter 0*) which records the power draw through the mains electricity socket (shown as a). This data is reported from all compute nodes to a centralised server(c), which can then be requested via HTTP, by any web capable device (b).

Given the hostname of node and a time frame, the API returns a series of measurements and the resolution at which the measurements were made. We use this infrastructure at GRID5000 as the basis for our api *libhclenergy*.

2.2 Experimental Platform

We chose GRID5000 [4] as our prototyping platform. It is large scale testbed for parallel computing. It provides a highly heterogeneous platform that can be configured by the researcher for their experiments. Many of the nodes contain accelerators and high speed interconnects representative of the current landscape in supercomputing. These include Nvidia GPU, Intel Xeon Phi and Infiniband. We believe our work here to be easily portable to other infrastructure. Recently the HPC job scheduling software Torque [1] added support for power measurement and management providing a suitable basis for production implementations of our *libhclenergy*.

2.3 Measurement of Distributed Applications

In the case of a single process, our API will measure energy and power between start and end time events as specified by the programmer. There can be multiple events and they may overlap, providing the programmer with the granularity of measurement that they require.

For the case of an application with multiple processes, we expose the total value for the node only at present.

In the case of an MPI application, we produce an energy reading for each participating node executing the application.

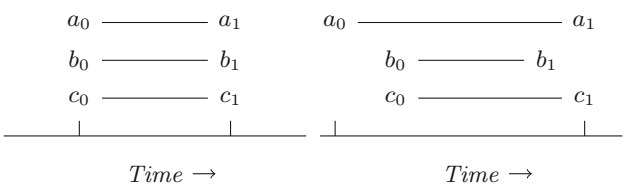

Fig. 2. Multiple processes on single node, both cases

There are two cases to be considered when measuring multiprocess or distributed applications. Firstly we can measure the power of all processes from the launch of the application until the final process exits. This is useful in the

context of a grid, where resources are reserved and the idle machines must be taken into account. Secondly we can measure each node only when it's executing our application. This is the actual power of our application. We do not want to measure a node with a high power draw if we're only using it for a small unit of time. Both cases are shown graphically in Fig. 2 for processes a, b and c. In the first diagram, the energy calculated will be the same for either method of calculation. For the second diagram however, the energy could be calculated for all processes a, b and c for the duration of process a's execution. Our alternative measurement method only accounts for energy of processes while they're executing.

2.4 API Features

Power consumption of a node can be characterised by the static(or idle) power of the components powered on, and the dynamic power, which is the power consumed by devices performing work. Our idle power consumption function allows the user to measure both the static and dynamic power of their application by simple subtraction.

We provide a utility function to calculate the idle power consumption of a node. This function puts the calling process to sleep, after which it requests the consumption for that period. The idle power is the baseline from which deviations are considered characteristic of the running application. In addition, we provide the energy cost in euros of running the application for a given price per kWh, provided by the utility company.

Table 1 demonstrates our API's key functions. Functions 1–3 provide a way to initialise, start and stop a measurement. Functions 4–6 provide measurements. The Raw variant of the PowerSeries function captures measurements for all participating processes at all times, as opposed to only when executing. Functions 7 and 8 are only available when instrumenting MPI applications. Function 7 gathers all measurements from all compute processes to the root process. Functions 9–11 are utilities, providing idle power of a compute node, average power of a series of power measurements and cost of electricity for a given event.

As researchers we want to be confident in the accuracy of our measurements. As such the developer may specify a confidence level and a tolerance for both power and energy measurements. For a segment of instrumented code, the measurements are repeated until the confidence level falls within tolerance or a set maximum iterations is reached.

Figures 3 and 4 show examples of our API in use. Each time an application is measured, the raw data is written to a file. In order to attain our required accuracy we script a repetitive execution until the confidence interval is below tolerance.

3 *Greenman*

We have developed a tool to profile applications, measure their energy consumption and fit existing state of the art statistical models. To the authors' knowledge,

```
int main() {
hclenergy_t *event = hcl_init();

hcl_start(event);
// Execute code for instrumentation
hcl_stop(event);

double energy = energy_consumed(event);
}
```

Fig. 3. Libhclenergy example of energy measurement

```
int main() {
hclenergy_t *event = hcl_init();

hcl_start(event);
// Execute code for instrumentation
hcl_stop(event);

struct host_power_series *power = getPowerSeries(event);
}
```

Fig. 4. Libhclenergy example of power measurement

these are representative of the current models found in the literature. This tool builds on our hclenergy API to gather power and energy measurements.

Presented in Table 2 are some of the existing models we have implemented in the tool. They are all statistical regression models. U components of the models denote utilisation as a percentage of clock cycles for CPU, and total bytes written and read in the cases of memory, disk and network.

The models parameterise the power consumed by a compute node. Energy can be derived from Eq. 1 when execution time is known.

All models are fitted using a variety of standard and robust methods (bisquare, cauchy, fair, huber, welsch and ordinary) from GSL [14]. Robust methods are used to counteract the effect of outliers in the data. As a system is composed of many processes, another scheduled process may interfere with data collection. Robust methods dampen their effect on our model fitting.

We collect statistics at a per core granularity for c-state and p-state occupancy, as well as percentage of clock cycles spent in our application. Modern processor cores operate in various states of alertness known as c-states. In the highest c-state *C0*, all features of the processor including clocks, caches and voltages are at maximum capacity. In the lowest c-state *C6* in the case of the Core 2 Duo, a processor can reduce the voltage of it's cores as low as 0 volts, with all internal clocks and caches disabled. There are gradual steps between these two extremes. Processors cores also have p-states representing each of the discrete frequencies a processor core can execute. Lower frequencies mean lower power consumption, but also lower performance. An application running at a low

Table 1. Key hclenergy API calls

No	Function
1	hclenergy_t *hcl_init();
2	void hcl_start(hclenergy_t *event);
3	void hcl_stop(hclenergy_t *event);
4	double energy_consumed(hclenergy_t *event);
5	struct host_power_series *getRawPowerSeries(hclenergy_t *event)
6	struct host_power_series *getPowerSeries(hclenergy_t *event);
7	struct host_power_series *gatherHostSeries(struct host_power_series *local);
8	double *energy_per_host(hclenergy_t *event);
9	double idlePower();
10	double avg_series(struct timeseries *series);
11	double cost(hclenergy_t *event, double price);

Table 2. Selection of models currently implemented

Model
$P = C_{base} + C_1 * U_{cpu}$, [13]
$P = C_{base} + \sum_{core_i=1}^{n} P_{core_i}$, [6]
$P = C_{base} + C_1 * U_{cpu} + C_2 * U_{cpu}^r$, [20]
$P = C_{base} + C_1 * U_{cpu} + C_2 * U_{I/O}$, [21]
$P = C_{base} + C_1 * U_{cpu} + C_2 * U_{disk} + C_3 * U_{net}$, [17]
$P = C_{base} + C_1 * U_{cpu} + C_2 * U_{disk} + C_3 * U_{net} + C_4 * U_{mem}$, [12]
$P = C_{base} + \sum_{core_i=1}^{n} C_i * f^3$, [22]

frequency taking a longer time may use more energy. We also record network packets per second per interface, memory footprint of the application, and bytes read and written to disk drives.

The sample rate of our power measurements is relatively low compared to our sample rate of application statistics. We interpolate our power readings in order to approximate the correct measurement for the given point in time. We provide approximation by akima, linear, cspline and polynomial splines.

The tool is executed on the Linux commandline as:

```
greenman <greenmanArguments> <resultsFolder> <Application> <Arguments>
```

As such the application under analysis does not require alteration. Any executable can be non intrusively instrumented. The source code of the application is not required for analysis.

If the user wishes to instrument segments of code in an application, the user must alter their code to tell greenman where to start and stop measuring. Models within this segment are calculated only using measurements collected inside these segments.

For each model we provide the researcher R^2, $R^2 Adj$, F statistic, and p value for each model parameter, χ^2, covariance matrix and correlation.

The tool is available as opensource software and is extensible as newer models arise. New models can be added by implementing a standard interface we provide. All models implemented so far use this interface, providing many examples on which to base new ones. We foresee new measurements to be required in the future and so we implement our measurement code in a similar extensible fashion.

As our tool is built in part on top of the PAPI library [24], any counters exposed now or in the future by it are supported.

When greenman is executed, it calls the fork() and exec() system calls to begin executing the researcher's application. We use the ptrace API which is primarily designed for implementing system call tracing and breakpoint debugging of applications. Ptrace allows us to control the application under analysis, frequently sampling it's application data from the/proc filesystem to build a time-series profile of the applications performance.

4 Applying Our API

Here we demonstrate the use of our tools to analyse a matrix multiplication application. We measure only during the kernel's execution. Allocation and initialisation of memory are not considered. We wish to understand the effect of number of the number of threads used in the computation. Using a non-distributed multithreaded implementation we vary the number of threads and measure the energy consumed separately on two compute nodes (Sagittaire 30 and 72). Both nodes are of dualcore x86_64 architecture(AMD Opteron 250) and are identical, with the exception of Sagittaire 72 having an additional hard disk drive and 16 GB instead of Sagittaire 30's 2 GB of ram.

The results of our experiment are shown in Table 3. We report confidence intervals at the 95 % level. We note that Power while executing the application does not vary with the number of threads used, but that it is heavily influenced by the idle power of the machine. As the CPU is the most power demanding component of most servers [13], the similarity between idle and active power led us to investigate if the CPU was not using frequency scaling features. We confirmed this to be true in our environmental setup. Enabling these features would cause a reduction in power consumption.

We also observe that the execution time for both machines is similar for the given number of threads. Energy however varies dramatically. For the same computation, *Sagittaire-72*, uses 727.17 J and 369.12 J more than *Sagittaire-30* for 1 and 2 threads respectively. This analysis tells us that we should use *Sagittaire-30* for this computation as we will use less energy and not suffer any performance degradation.

The cost of energy is shown in Table 4. Though the costs are low, we must consider how they scale for longer running applications on a greater number of compute nodes.

Table 3. Power and energy measurements

Machine	#Threads	Power (W)	Energy (J)	Idle	Time(s)
Sagittaire-30	1	175.14 ± 0.40	2900.37 ± 6.90	174.58 ± 0.38	16.56
Sagittaire-30	2	175.28 ± 0.32	1560.72 ± 3.33	174.586 ± 0.38	8.90
Sagittaire-72	1	218.53 ± 4.26	3627.53 ± 70.81	215.975 ± 0.69	16.60
Sagittaire-72	2	217.29 ± 0.91	1929.84 ± 8.21	215.975 ± 0.69	8.88

Table 4. Energy costs

Machine	#Threads	Cost(4.125 c/KWh)
Sagittaire-30	1	€ 0.0033
Sagittaire-30	2	€ 0.0017
Sagittaire-72	1	€ 0.0041
Sagittaire-72	2	€ 0.0022

5 Related Works

Related tools for model prediction include JouleTrack [26], a web based tool for application profiling and energy estimation for StrongARM and Hitachi SH-4 processors. Dunkels [11], provides an energy estimation framework for small sensor web devices based on work by [29] which assumes that a larger infrastructure would be able to measure it's own energy via ACPI [2]. While power measurement via ACPI is part of the standard since version 4, we do not have access to machines that support it. Neither of these tools target the architectures and infrastructure that we do.

Barrachina [5] presents pmlib, a software package for measuring energy states on CPUs. This library provides whole node level measurements accurately, but does not capture finer grained measurements such as that of components and accelerators. However, it has the advantageous ability to interface with high frequency external measurement devices.

Cabrera provides EML [8] which are similar contributions to our *libhclenergy*, but lacks the ability to report statistical confidence and also to transparently calculate per node power in MPI applications.

In addition to these software methods, there are such as PowerPack [15] and PowerMon2 [7] that intercept power rails of components to give component level. These methods are difficult to deploy in real systems and are disadvantaged by the complexity of measuring devices with multiple power rails [18].

To the authors' knowledge there is no existing tool for energy profiling and model testing.

5.1 Existing Tools

We limit ourselves to a node level granularity, but should the reader be interested in finer grained measurement at the device level, we advise you to consult the following tools which are performance counter based.

Intel provide the RAPL interface [10] which is a software power model and similarly AMD provides APM [3]. Though easily accessible, both have disadvantages. Intel RAPL fails to provide power measurements, only providing energy with no timestamp data, hindering indepth analysis [16] and AMD APM has been shown to be inaccurate due to assumptions during sleep modes [16].

Likwid [28], a lightweight performance tool offers RAPL measurements from Intel SandyBridge and IvyBridge x86 processors.

Nvidia, through their management library (NVML [25]) provides access to milliwatt power consumption metrics, accurate to 5 %, as well as current Pstate of each graphics card in a system. NVML also provides related metrics such a fan speed and temperature.

6 Conclusion

We have provided a prototype implementation of a power and energy monitoring API for a modern parallel infrastructure as well as a tool for model fitting. These tools allow us to test a variety of schemes for power approximation when the origin sample frequency is low. Both tools will be released under an opensource license in the coming weeks.

Future hardware will likely by necessity include higher precision energy measurement capability. Current accelerator devices are capable of updating their power consumption data as frequently as 10 Hz.

7 Future Works

A current limitation of greenman is an inability to profile MPI applications due to the design patterns used to construct the tool. We aim to resolve this in subsequent releases.

Our api provides us with the ability to instrument sections of code. We plan to use this api to instrument different tasks of workflow applications to best allocate tasks for energy efficiency.

Current generations of Nvidia GPGPU and Intel Xeon Phi accelerators have the facility to report their own board power consumption through their vendor APIs [19,25]. As these are essentially performance counters, we will be adding them to the metrics that greenman can record. From there, we will implement existing accelerator power models [23,27] and provide an extensible interface for researchers to add their own models.

We will be exploring functional models of applications on heterogeneous platforms. The Heterogeneous Computing Lab has produced Fupermod [9] for producing optimal data partitioning in heterogeneous environments. We will be augmenting this software with energy measurements.

Acknowledgment. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

References

1. Adaptive Computing, I: Torque resource manager (2015). http://www.adaptivecomputing.com/products/open-source/torque/
2. H.P.C., et al.: Acpi v4.0a (2010). http://www.acpi.info/DOWNLOADS/ACPIspec40a.pdf
3. AMD: Bios and kernel developerś guide(bkdg) for amd family 15h models 00h–0fh processors (2013). http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/42301_15h_Mod_00h0Fh_BKDG1.pdf
4. Balouek, D., et al.: Adding virtualization capabilities to the Grid'5000 testbed. In: Ivanov, Ivan I., van Sinderen, Marten, Leymann, Frank, Shan, Tony (eds.) CLOSER 2012. CCIS, vol. 367, pp. 3–20. Springer, Heidelberg (2013)
5. Barrachina, S., Barreda, M., Catalán, S., Dolz, M.F., Fabregat, G., Mayo, R., Quintana-Ortí, E.S.: An integrated framework for power-performance analysis of parallel scientific workloads. In: ENERGY 2013, The Third International Conference on Smart Grids, Green Communications and IT Energy-Aware Technologies, pp. 114–119 (2013)
6. Basmadjian, R., Ali, N., Niedermeier, F., de Meer, H., Giuliani, G.: A methodology to predict the power consumption of servers in data centres. In: Proceedings of the 2nd International Conference on Energy-Efficient Computing and Networking, pp. 1–10. ACM (2011)
7. Bedard, D., Lim, M.Y., Fowler, R., Porterfield, A.: Powermon: fine-grained and integrated power monitoring for commodity computer systems. In: Proceedings of the IEEE SoutheastCon 2010 (SoutheastCon), pp. 479–484, March 2010
8. Cabrera, A., Almeida, F., Arteaga, J., Blanco, V.: Measuring energy consumption using EML (energy measurement library). Comput. Sci. Res. Dev. **30**(2), 135–143 (2015). http://dx.doi.org/10.1007/s00450-014-0269-5
9. Clarke, D., Zhong, Z., Rychkov, V., Lastovetsky, A.: Fupermod: a software tool for the optimization of data-parallel applications on heterogeneous platforms. J. Supercomput. **69**(1), 61–69 (2014)
10. David, H., Gorbatov, E., Hanebutte, U.R., Khanna, R., Le, C.: Rapl: memory power estimation and capping. In: 2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED), pp. 189–194, August 2010
11. Dunkels, A., Osterlind, F., Tsiftes, N., He, Z.: Software-based on-line energy estimation for sensor nodes. In: Proceedings of the 4th Workshop on Embedded Networked Sensors, pp. 28–32. ACM (2007)
12. Economou, D., Rivoire, S., Kozyrakis, C., Ranganathan, P.: Full-system power analysis and modeling for server environments. In: Proceedings of Workshop on Modeling, Benchmarking, and Simulation, pp. 70–77 (2006)
13. Fan, X., Weber, W.D., Barroso, L.A.: Power provisioning for a warehouse-sized computer. ACM SIGARCH Comput. Archit. News **35**(2), 13–23 (2007)
14. Galassi, M., et al.: Gnu Scientific Library Reference Manual, 3rd edn. Network Theory Ltd., Bristol (2009). http://www.gnu.org/software/gsl/manual/gsl-ref.ps.gz

15. Ge, R., Feng, X., Song, S., Chang, H.C., Li, D., Cameron, K.: Powerpack: energy profiling and analysis of high-performance systems and applications. IEEE Trans. Parallel Distrib. Syst. **21**(5), 658–671 (2010)
16. Hackenberg, D., Ilsche, T., Schone, R., Molka, D., Schmidt, M., Nagel, W.: Power measurement techniques on standard compute nodes: A quantitative comparison. In: 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 194–204, April 2013
17. Heath, T., Diniz, B., Horizonte, B., Carrera, E.V., Bianchini, R.: Energy conservation in heterogeneous server clusters, pp. 186–195 (2005)
18. Hsu, C.H., Poole, S.: Power measurement for high performance computing: state of the art. In: 2011 International Green Computing Conference and Workshops (IGCC), pp. 1–6, July 2011
19. Intel Corporation: Intel manycore platform software stack (2015). https://software.intel.com/en-us/articles/intel-manycore-platform-software-stack-mpss
20. Jung, G., Hiltunen, M.A., Joshi, K.R., Schlichting, R.D., Pu, C.: Mistral: dynamically managing power, performance, and adaptation cost in cloud infrastructures. In: 2010 IEEE 30th International Conference on Distributed Computing Systems (ICDCS), pp. 62–73. IEEE (2010)
21. Kansal, A., Zhao, F.: Fine-grained energy profiling for power-aware application design. ACM SIGMETRICS Perform. Eval. Rev. **36**(2), 26 (2008). http://portal.acm.org/citation.cfm?doid=1453175.1453180
22. Kim, K.H., Beloglazov, A., Buyya, R.: Power-aware provisioning of virtual machines for real-time cloud services. Concurrency Comput. Pract. Exp. **23**(13), 1491–1505 (2011)
23. Lai, Z., Lam, K.T., Wang, C.L., Su, J.: A power modelling approach for many-core architectures. In: 2014 10th International Conference on Semantics, Knowledge and Grids (SKG), pp. 128–132, August 2014
24. Mucci, P.J., Browne, S., Deane, C., Ho, G.: Papi: a portable interface to hardware performance counters. In: Proceedings of the Department of Defense HPCMP Users Group Conference, pp. 7–10 (1999)
25. Nvidia Corporation: Nvidia management library (2015). https://developer.nvidia.com/nvidia-management-library-nvml
26. Sinha, A., Chandrakasan, A.P.: Jouletrack: a Web based tool for software energy profiling. In: Proceedings of the 38th Annual Design Automation Conference, pp. 220–225. ACM (2001)
27. Song, S., Su, C., Rountree, B., Cameron, K.W.: A simplified and accurate model of power-performance efficiency on emergent gpu architectures (2013)
28. Treibig, J., Hager, G., Wellein, G.: Likwid: a lightweight performance-oriented tool suite for x86 multicore environments. In: 2010 39th International Conference on Parallel Processing Workshops (ICPPW), pp. 207–216. IEEE (2010)
29. Zhao, Y.J., Govindan, R., Estrin, D.: Residual energy scan for monitoring sensor networks. In: 2002 IEEE Wireless Communications and Networking Conference, WCNC 2002, vol. 1, pp. 356–362. IEEE (2002)