High-Resolution Power Profiling of GPU Functions Using Low-Resolution Measurement

Jens Lang and Gudula Rünger

Department of Computer Science, Chemnitz University of Technology, Germany
{jens.lang,gudula.ruenger}@cs.tu-chemnitz.de

Abstract In order to be able to minimise the energy consumption of an application program, information about the specific energy consumption is required. Modern Nvidia graphics processing units (GPUs) measure their current power consumption and the driver makes the value available to the application every 20 ms. However, for evaluating the energy consumption of GPU kernel functions, such a sampling interval might not be sufficient since the kernels may have a shorter execution time.

This article proposes a method for generating high-resolution power profiles, which is the power consumption of a specific function depending on the progress of its execution. The method uses low-resolution measuring instruments offered by GPUs. Power measurements obtained during several executions of the function are combined into a single power profile. The resulting power profile contains power values in intervals which are much shorter than the sampling interval of the hardware driver so that even short-term power changes can be considered, e.g. for calculating the energy consumption of a single function. The article also shows how to extend the approach to an online generation of power profiles. Furthermore, an overview on the power profiles of some important functions, such as BLAS routines, is given.

Keywords: power profiles, power measurement, GPU computing.

1 Introduction

Energy efficiency of codes will become more and more important in scientific computing. Especially general-purpose graphics processing units (GPUs) promise to be a potentially more energy-efficient alternative to CPUs [3,9,21]. Measuring the energy consumption of a hardware is necessary in order to be able to minimise the energy consumption of an application. For quantifying the energy consumption of a piece of code, basically two approaches exist: hardware-based and software-based energy measurement. For hardware-based measurement of the energy consumption, a measuring device is installed between the power supply and the computational device. From the current and the voltage measured, the electrical power P can be computed. By integrating the power over a time interval, the electrical energy consumption E is determined. The observation of influences evoked by short sections of code, requires a device with a high sampling frequency, e.g. 500 Hz [21], 1 kHz [12] or even 50 kHz [7]. For the software-based

F. Wolf, B. Mohr, and D. an Mey (Eds.): Euro-Par 2013, LNCS 8097, pp. 801–812, 2013.

measurement, modern CPUs, such as Intel's Sandy Bridge or AMD's Bulldozer CPUs, offer an interface for retrieving the value of the energy consumption since some starting time [11, Vol. 3B, Chap. 14.7], [4, Chap. 3.13]. Similarly, the latest GPUs of the manufacturer Nvidia provide the possibility to read the current power consumption by software [19].

The Nvidia Management Library (NVML) provides power measurement [19] for Nvidia GPUs. The power value is updated every 20 ms; this is described in Sect. 3.2. However, such a sampling interval is not sufficient for a precise evaluation of the energy consumption of a GPU kernel function, especially for those with a short execution time. Such kernel functions occur in many areas, such as the solution of linear systems of equations in FEM simulations or Monte Carlo simulations in the field of numerical mathematics as well as, for example, scan algorithms in the field of data processing. Therefore, a method is needed which obtains an accurate, high-resolution power-consumption profile of a GPU function. Furthermore, the availability of high-resolution power profiles are needed to extend approaches which use RAPL to verify energy models, such as [20], to GPUs.

The contribution of this article is to provide such a method for generating high-resolution power profiles of GPU kernel functions, which is based on the measurement interface with a low sampling frequency. These profiles can be used for evaluating and improving the energy consumption of GPU kernel functions. Both, an offline method and an online method for generating the power profiles, are presented. The online method has a low overhead so that it can be used during the actual execution of, e.g., simulation programs which select code paths depending on the energy consumption of specific routines.

The rest of this article is organised as follows: Section 2 presents related work. Section 3 gives an overview on measuring the power consumption on Nvidia GPUs. Section 4 presents the method for generating power profiles from low-resolution measurements, and Sect. 5 gives some sample profiles. The method is extended to an online method in Sect. 6, and Sect. 7 concludes the article.

2 Related Work

Measuring and evaluating the energy consumption of GPUs has been in the focus of research for several years. For example, Huang et al. [9] compare the energy consumption of a biological code for creating an electrostatic potential map on a CPU and on a GPU. Rofouei et al. [21] investigate the energy consumption for some applications from the CUDA SDK executed on a GPU and on a CPU. Both report that the GPU executes the respective codes more efficiently concerning the energy. In contrast, Chen and Singh [5] find no huge difference in the energy consumption for the CPU and the GPU for a document filtering code. Abe et al. [3] measure the effects of frequency and voltage scaling in CPUs and GPUs.

Nagasaka et al. [16] and Chen et al. [6] propose statistical power consumption models for GPUs: Both models execute a set of benchmark functions, measure their power consumption, and record some hardware performance counters of the

GPU. They derive a model which allows the prediction of the power consumption of a kernel function by using the values of the hardware performance counters obtained during its execution. Nagasaki et al. set up this model using linear regression, Chen et al. use a tree-based method.

Collange et al. [7] analyse the influence of computations and memory accesses on the power consumptions of different GPUs. Hong et al. [8] develop an empirical model for predicting the power consumption of a GPU by counting specifics of the code executed, such as the number of memory accesses, the number of streaming multiprocessors used or the use of the computational units. Li et al. [14] extend this model by a possibility to also take streaming multiprocessors executing different workload into account. A similar model is proposed by Kasichayanula et al. [13]: They measure the energy consumption of GPU components, such as global memory, shared memory, floating point unit, etc., and multiply it by the time a given function activates the respective component. Furthermore, they investigate some of the details of the software interface for reading the power consumption of Nvidia GPUs. For the sampling frequency f of the Tesla C2075, they report a value of 62.5 Hz, while the measurements as presented in Sect. 3.2 of this article, resulted in a value of $f = 50$ Hz, possibly due to a different methodology. Weaver et al. from the same research group report a sampling frequency of "roughly 60 Hz" [22].

Hähnel et al. deal with measuring the energy consumption of functions that are shorter than the measuring interval [10]. They measure the energy consumption of the CPU using the Intel RAPL machine-specific registers [11]. They overcome the issue that large sampling intervals might lead to inaccurate results by starting the function exactly at the beginning of a sampling interval of 1 ms and executing a workload with a well-known energy consumption after its return until the end of the sampling interval. Compared to the Nvidia NVML interface, the Intel RAPL has the advantage that the user can retrieve the energy consumption directly and thus does not need to integrate power values. Furthermore, there is no need for clock synchronisation when only measuring on the CPU.

All statistical and empirical models described above may deliver a sufficient temporal resolution for also analysing the energy consumption of short kernel functions. But if such models are used for cases their designers were not aware of, the results might become inaccurate: McCullough et al. [15] indicate that power consumption predictions based on hardware performance counters are inaccurate in complex situations. Therefore, measuring the real power consumption is essential. For all users that do not have access to dedicated measurement hardware, the software-based method proposed in this article might be suitable.

3 Power Consumption of GPUs

Modern GPUs suitable for general-purpose computing implement dynamic frequency scaling in order to save energy in idle mode and to comply with their specified thermal design powers during phases of energy-demanding computations [1,2]. For Nvidia's GPUs of the latest generation, the Nvidia Management

Library (NVML) [19] offers a software interface for reading the current electrical power consumption.

3.1 Retrieving the Power Consumption

Figure 1 shows the measuring of power consumptions on Nvidia GPUs: There is a specific timer which is triggered in intervals of size T, called the *sampling interval* in the following. At the beginning of each sampling interval, the GPU driver reads the current power consumption P and stores the value. The figure shows the power consumption for the case that the GPU has a power consumption of 70 W in idle mode and of 130 W when executing the user-specified kernel function. However, if the user reads the power value from the GPU, he or she gets data as shown in the line "claimed power consumption", which does not reflect the real power consumption of the GPU.

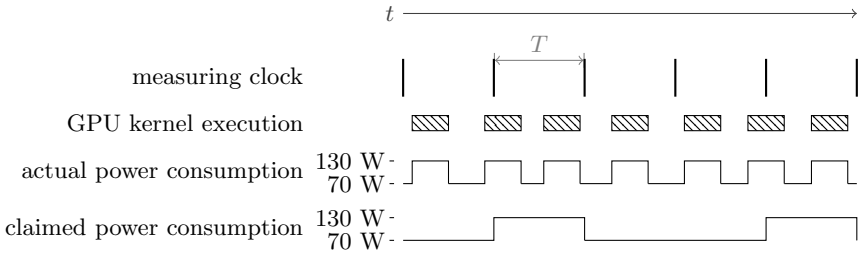


Fig. 1. Repeated execution of a GPU kernel function and measuring its power consumption

The software interface for retrieving the power value is provided by the routine `nvmlDeviceGetPowerUsage(nvmlDevice_t device, unsigned int* power)` of the NVML. The value is measured in milliwatts. It reflects the current electrical power of the whole GPU board including memory, etc. and has an error of ±5 W [19]. All measurements of this article were conducted on a Nvidia Tesla C2075 GPU [17] built into a machine with two octacore Sandy-Bridge CPUs E5-2650 having a clock speed of 2.0 GHz. The machine runs the Linux kernel version 3.2 with the Nvidia GPU driver version 304.64 and CUDA version 4.2.9.

3.2 Measuring the Sampling Interval *T*

The experiment for measuring the sampling interval T of the GPU makes use of the fact that the power value retrieved is very noisy so that two subsequent measurements normally differ even if the state of the GPU has not changed: The measurement algorithm continually retrieves the current power value P_{curr} from the GPU. If P_{curr} differs from the power value P_{last} retrieved before, it can be inferred that the hardware has updated its power value. Then, the current

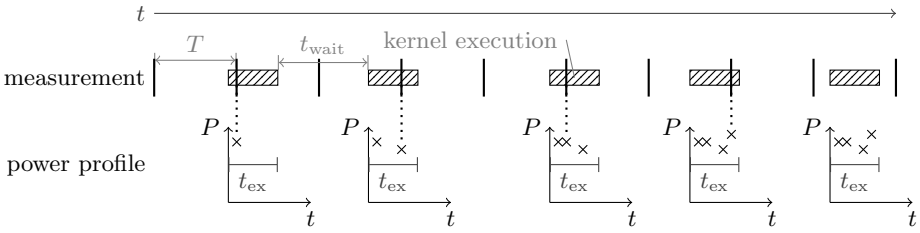


Fig. 2. Illustration of the creation of a power profile from power consumption values measured during repeated GPU kernel execution

time is taken and the time interval elapsed since the last update is emitted. This procedure is repeated for 100 000 times so that one gets 100 000 values for the sampling interval T. In rare cases (0.5 ‰), the power value P_{curr} did not change in two consecutive periods which resulted in a value two or three times as big as the other values. These values have been eliminated. T is then calculated as the arithmetic mean of the remaining values.

For measuring the times, the POSIX function `clock_gettime` is used, which measures the wall-clock time with an accuracy of 1 ns. A value of $T = (20.0082 \pm 0.0008)$ ms, i.e. a relative error of roughly 0.03 ‰, has been determined. The value for the sampling interval of $T \approx 20$ ms results in a sampling frequency of $f = \frac{1}{T} \approx 50$ Hz.

4 The Generation of Power Profiles

For the generation of high-resolution power profiles, i.e. a diagram which shows the electrical power consumption of the GPU during the execution of one kernel function, a statistical method is used to overcome the restrictions imposed by the low sampling frequency of the measurement instrument: The GPU kernel function to be evaluated is executed a large number of times in the range of some hundreds, each time starting at a different phase in the sampling interval. The method is illustrated in Fig. 2: Multiple executions of the kernel function are performed with a random waiting time t_{wait} between them. After starting the execution, the power value is monitored constantly and for each update, this value is emitted together with the corresponding update time relative to the starting time of the function. The two values are used for a step-wise creation of a diagram as shown at the bottom of the figure. If no power value update happens during the execution time of the kernel t_{ex}, no value is emitted as illustrated in the last diagram at the bottom.

Using this approach, the generation of power profiles is performed as shown in Alg. 1: At first, the GPU kernel is executed once in order to determine its execution time t_{ex} (Lines 1 to 5). The algorithm assumes that the execution time does not vary between two calls, i.e. the kernel should always be called with the same parameters. Then, the function is called n_{ex} times. For the experiments

```
1  retrieve current time t_start
2  execute GPU kernel asynchronously
3  wait for GPU kernel to finish
4  retrieve current time t_ret
5  t_ex := t_ret − t_start
6  for k = 1 to n_ex do
7      wait a random time t_wait
8      retrieve GPU power P_last
9      execute GPU kernel asynchronously
10     retrieve current time t_start; t_curr := t_start
11     while t_curr < t_start + t_delay + t_ex + T do
12         retrieve GPU power P_curr
13         if P_curr ≠ P_last then
14             retrieve current time t_update
15             emit t_update − t_start, P_curr
16             P_last := P_curr
17         retrieve current time t_curr
```

Algorithm 1: Retrieving power measurements for a GPU kernel function

presented in Sect. 5, $n_{\text{ex}} = 100$ was chosen, which results in an average of 100 values per sampling interval, i.e. 5000 values per second. Before the execution of the GPU kernel, the CPU is halted for a randomly chosen time interval (Line 7) in order to ensure that the power value is determined at a random point of the execution of the function. The waiting time t_{wait} is distributed uniformly in the interval $T \leq t_{\text{wait}} < 2T$ to avoid that the power value of the previous execution is read. Then, the GPU kernel is executed asynchronously (Line 9). Experiments, see Sect. 5, have shown that reading the starting time of the kernel t_{start} right after the return of the routine starting the kernel leads to a constant delay of $t_{\text{delay}} \approx 6$ ms between the starting time and the increase of the measured power. The while loop in Line 11 continually retrieves the power value as long as the GPU kernel is running. Each time the value P_{curr} changes (Line 13), it is emitted along with the time elapsed since the start of the kernel execution (Line 15).

The output of Alg. 1, i.e. a sequence of pairs (P, t), is then processed further to create a diagram. The P values are emitted in milliwatts and are in the range from the *long idle power* of 35 W [13] to the *thermal design power* of 225 W [17]. The t values indicate how many nanoseconds elapsed since the start of the GPU function. They are in the range from 0 to $t_{\text{ex}} + T$. A power profile diagram comprises the points resulting from all (P, t) pairs of one measurement. Results obtained with this method are presented in Sect. 5.

The algorithm presented in this section is only suitable for offline generation as it completely occupies one CPU core in the time interval beginning at the start of the GPU kernel execution until the power update which succeeds the termination of the kernel. However, the continual polling of the power value did not cause any observable effects on the execution of the GPU kernel.

5 Sample Power Profiles

The diagrams in Fig. 3 (a)–(e) show the power consumptions for various call configurations of `xgemm`, wich are matrix-matrix multiply BLAS functions from the CUBLAS package [18], and one further function, `vectid`. The `vectid` routine performs arithmetic operations and memory accesses in a way such that the GPU threads have different execution times. Each point in a diagram stands for one *time–power* (P, t) pair emitted by Alg. 1.

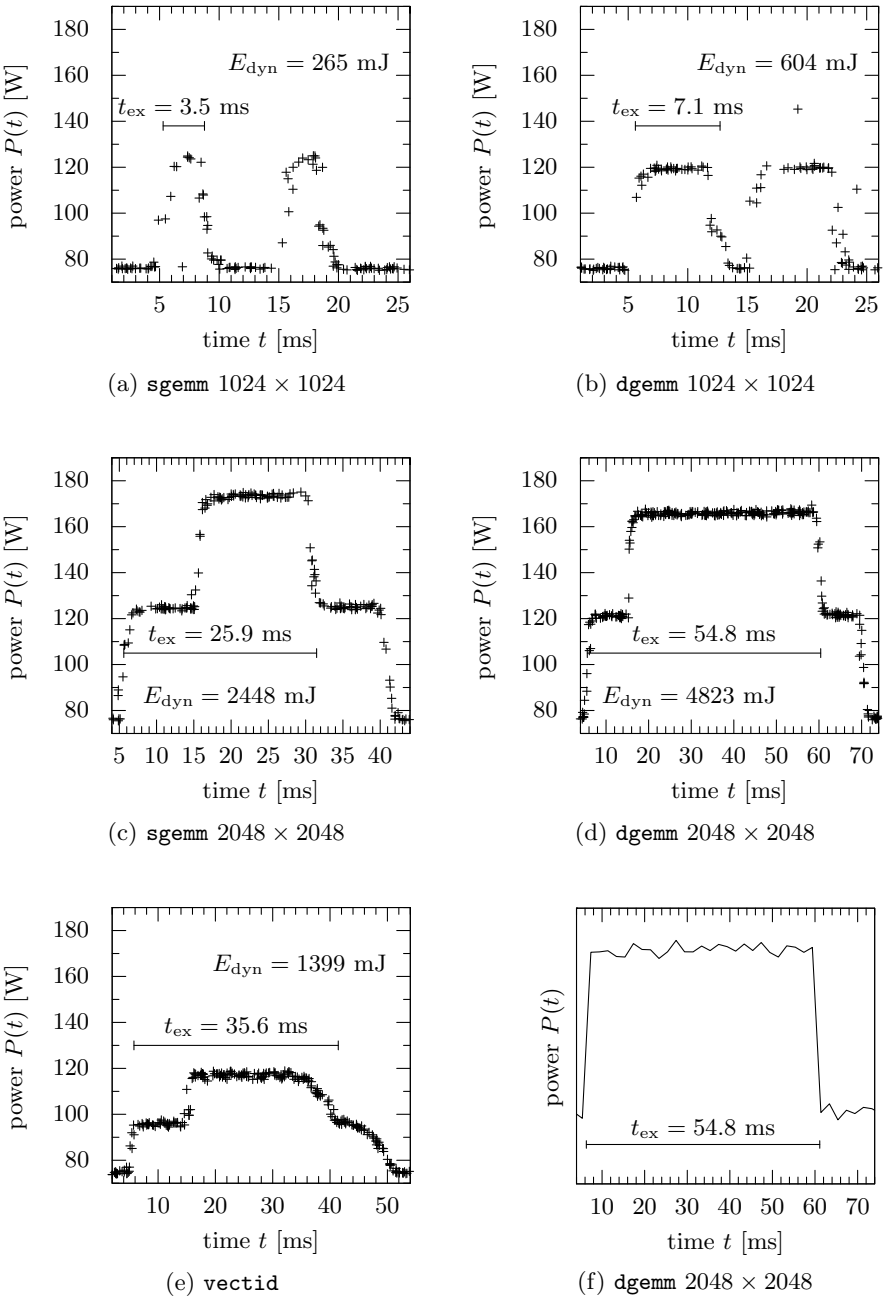
Splitting up the total power P_{tot} into static power P_{stat} and dynamic power P_{dyn} as

$$P_{\text{tot}} = P_{\text{stat}} + P_{\text{dyn}}$$

is often suggested, e.g. in [14]. For the static power in the diagrams, a value of $P_{\text{stat}} = 76$ W can be determined. The dynamic power depends on the specific workload to be processed on the GPU. For the `sgemm` call with a matrix size of 1024×1024, one can see that the power consumption increases up to 123 W during the execution of the routine, and then decreases when the routine is finished. After 5 ms, there is a second peak of roughly the same shape. This effect is closely related to the staircase effect that can be seen in the 2048×2048 case: The power consumption increases to 124 W starting at second 6.3 and then again increases to 172 W starting at second 16.3. The behaviour upon the termination of the routine is analogous. As all measurements show a similar effect, it looks as if one half of the power consumption of the GPU might have a delay of 10 ms in the measurement. This means, that integration over the whole interval leads to a correct value for the energy; the actual value for P_{dyn}, however, is twice as large as suggested by the figure during the first 10 ms of the execution of a function. Consequently, to compute the actual power consumption for the two 1024×1024 `xgemm` calls, the dynamic power suggested by the figure has to be doubled, i.e. for `sgemm`, the value of P_{tot} is 172 W instead of 124 W with $P_{\text{dyn}} = 96$ W. This corresponds to the 2048×2048 `sgemm` case in which P_{tot} is 172 W as well. For the `dgemm` cases, the dynamic power is $P_{\text{dyn}} = 92$ W with a P_{tot} of 166 W.

The `vectid` test function is a hand-written CUDA function. It mainly consists of a for loop running from 0 to $10 \cdot tid$, where *tid* is the thread id of the CUDA thread in the thread block. There are 32 blocks and 1024 threads per block. Inside the loop, 5 floating point operations are performed. Furthermore, two array entries are read from global memory and one is written. The different execution times of the different threads create imbalance which results in a smoother decrease of the power consumption than for the `xgemm` routines. Such an effect would not have been visible with measuring methods that have a coarser resolution.

The values for the dynamic energy consumption E_{dyn} shown in Fig. 3 have been obtained by integrating the dynamic power consumption using the trapezoid rule with the values shown in the respective profile. Fig. 3 (f) was obtained by measuring the electric current flowing through the external PCI Express power connectors of the GPU card. The current multiplied by the voltage gives the



(a) `sgemm` 1024×1024 (b) `dgemm` 1024×1024

(c) `sgemm` 2048×2048 (d) `dgemm` 2048×2048

(e) `vectid` (f) `dgemm` 2048×2048

Fig. 3. (a)–(e) Power profiles of CUBLAS `x`gemm and `vectid` kernels; (f) Power profile obtained by hardware measurement

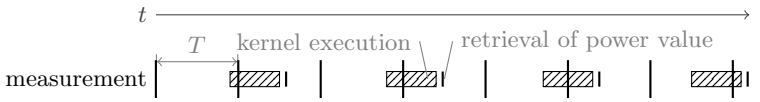


Fig. 4. Creating a power profile using the online method

electric power. Since further current flows through the PCI express socket, the actual values are inaccurate and they have been omitted. Nonetheless, one can see an increase in the power consumption for the duration of the execution of the kernel in the correct time interval.

The thermal design power of 225 W could not be reached with any `xgemm` experiment. The CUBLAS `dgemm` with a matrix size of $14 \cdot 2^{10} \times 14 \cdot 2^{10}$, which occupies all available streaming processors, consumed only 163 W, or 171 W if the GPU was pre-heated to roughly 85 °C. This corresponds to the results of Kasichayanula et al. [13] who report a value of 180 W for the average power consumption of the MAGMA `dgemm` kernel with a matrix size of 8192×8192, which is also significantly lower than the thermal design power.

6 Online Generation of Power Profiles

The approach for offline generation of power profiles as presented in Sect. 4 is adapted in order to allow the online generation of power profiles. This online method has less CPU usage and does not extend the execution time of the GPU kernel. The method is illustrated in Fig. 4. If the GPU kernel to be evaluated has an execution time lower than T, it is sufficient to retrieve one value of the power P_{curr} immediately after the termination of the GPU kernel. One can calculate the time of the last update of the power value precisely from the base time t_{sync} of the sampling interval and its period T. If the power value was updated during the execution of the GPU function, this value can be used for the power profile. This approach works on the condition that the machine has very precise clocks so that the times can be calculated exactly.

Experiments, however, have shown that the CPU and the GPU clocks slightly drift apart: This results in power profiles showing no clear peak unless a re-synchronisation is performed at least every 0.2 s. If this synchronisation is performed during the execution of a GPU kernel, not too much overhead is added as often the CPU is not fully occupied during GPU execution. By predicting the beginning of the next sampling interval and the termination time of the GPU kernel, one can estimate if the next update of the power value occurs during the execution.

The online generation of power consumption profiles is conducted as shown in Alg. 2: The function `syncClocks` (Lines 1 to 8) waits for the update of the power value and then returns the current time. The function `main` represents the structure of a general application algorithm of which the GPU energy consumption is to be evaluated. The while loop in Line 19 repeatedly calls the GPU kernel (Line 22). Before the while loop is started, the time t_{sync} at which the power value is updated, is determined (Line 16).

```
1  function syncClocks()
2      retrieve GPU power P_curr
3      P_last := P_curr
4      while P_last = P_curr do
5          P_last := P_curr
6          retrieve GPU power P_curr
7      retrieve current time t_curr
8      return t_curr
9  function syncIfPossible()
10     t_expected_finish := t_start + β t_ex
11     t_next_update := ⌊(t_start − t_sync)/Δt + 1⌋ Δt
12     if t_expected_finish > t_next_update then
13         t_sync := syncClocks()
14         n_last_sync := n_call
15 function main()
       // main algorithm
16     t_sync := syncClocks()
17     n_call := 0
18     n_last_sync := 0
19     while ... do
           // main algorithm
20         n_call := n_call + 1
21         retrieve current time t_start
22         call GPU kernel
23         syncIfPossible()
24         wait for GPU kernel to finish
25         retrieve current time t_finish
26         retrieve GPU power P_curr
27         t_φ := t_next_update − t_start
28         emit t_φ, P_curr
29         if n_call − n_last_sync > γ then
30             t_sync := syncClocks()
31             n_last_sync := n_call
           // main algorithm
```

Algorithm 2: Online generation of a power profile

The starting time t_{start} of the GPU kernel is taken in Line 21 and the kernel is executed asynchronously in Line 22. In the function `syncIfPossible`, the clock synchronisation takes place concurrently to the GPU kernel execution: If the expected termination time of the GPU kernel $t_{\text{expected\_finish}}$ is greater than the expected next update of the power value at time $t_{\text{next\_update}}$, the actual time of the update is determined by the function `syncClocks`. For β in Line 10, a value of $\frac{4}{5}$ has been used in order to avoid a synchronisation in the late phase of the GPU kernel execution. After the termination of the GPU function (Line 24), the current power value P_{curr} is retrieved (Line 26). In Line 27, t_φ, i.e. the phase of the sampling interval in which the kernel function has been started, is calculated. Next, the values t_φ and P_{curr} are emitted.

As mentioned above, the clocks drift apart too much after roughly 0.2 s. Therefore, a re-synchronisation is forced in Lines 29 to 31 if there has been no synchronisation in the function `syncIfPossible` for a while. The value of γ should be set in a way that the re-synchronisation takes place at least every 0.2 s.

The overhead introduced by the measurement, i.e. the extension of the total execution time of the algorithm on the CPU, was not above 10 % in the test cases. A profile created using online generation can be seen in Fig. 5. The 1024×1024 case shows one peak similar to that in Fig. 3(a). Its power consumption corresponds to that measured in Sect. 5. However, the second peak is not visible, possibly it occurs too late after the execution of the function. Due to this effect, currently only GPU functions with an execution time less than 10 ms can be properly evaluated using the online method.

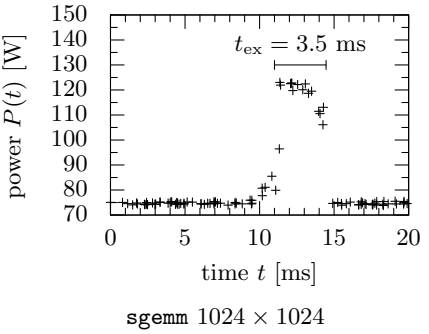


Fig. 5. Online-generated power profile

7 Conclusion

The article has presented a method which can generate high-resolution power profiles for GPU functions even if there are only measurement instruments with a low temporal resolution available. The sampling interval of the measurement offered by the NVML has been determined to be 20 ms. The method presented allows the generation of high-resolution power profiles of GPU functions without requiring any additional hardware. Such power profiles can for example be used by developers to optimise the power consumption of their code. By integrating the power values, the energy consumption of a specific function can be calculated, which can, e.g., be used for auto-tuning its energy consumption.

The offline method for generating power profiles works very accurately, so that some interesting effects could be demonstrated at the sample profiles shown, such as the smooth decrease of the power consumption for unbalanced loads. The method has been extended to an online method in order to enable the generation of power profiles during the execution of simulations etc. There is only a low overhead. Such profiles can, e.g., be used for auto-tuning at runtime. Restrictions of the online method are that it is currently only suitable for kernels with an execution time of less than 10 ms and that its accuracy is lower than the accuracy of the offline method. In general, the method presented is not restricted to GPUs but can also transferred to other measurement instruments whose sampling interval is too large for the targeted purpose.

Acknowledgements. This work is supported by the cluster of excellence *Energy-Efficient Product and Process Innovation in Production Engineering* (eniPROD) funded by the European Union (ERDF) and the Free State of Saxony.

References

1. Powermizer 8.0: Intelligent power management technology. Tech. rep., Nvidia (June 2008), tB-04051-001_v01
2. AMD PowerTune technology. Whitepaper, AMD (December 2010), `http://www.amd.com/uk/Documents/PowerTune_Technology_Whitepaper.pdf`

3. Abe, Y., Sasaki, H., Peres, M., Inoue, K., Murakami, K., Kato, S.: Power and performance analysis of GPU-accelerated systems. In: Workshop on Power Aware Computing and Systems, HotPower 2012 (2012)
4. Advanced Micro Devices: BIOS and Kernel Developer's Guide (BKDG) for AMD Family 15h Models 00h-0Fh Processors, rev. 3.12 (October 2012)
5. Chen, D., Singh, D.: Using OpenCL to evaluate the efficiency of CPUs, GPUs and FPGAs for information filtering. In: 22nd Int. Conf. on Field Programmable Logic and Applications (FPL), pp. 5–12 (2012)
6. Chen, J., Li, B., Zhang, Y., Peng, L., Peir, J.K.: Statistical GPU power analysis using tree-based methods. In: Int. Green Computing Conf. and Workshops (IGCC), pp. 1–6 (2011)
7. Collange, S., Defour, D., Tisserand, A.: Power consumption of GPUs from a software perspective. In: 9th Int. Conf. on Computational Science (2009)
8. Hong, S., Kim, H.: An integrated GPU power and performance model. SIGARCH Comput. Archit. News 38(3), 280–289 (2010)
9. Huang, S., Xiao, S., Feng, W.: On the energy efficiency of graphics processing units for scientific computing. In: IEEE Int. Symp. on Parallel Distributed Processing (IPDPS 2009), pp. 1–8 (2009)
10. Hähnel, M., Döbel, B., Völp, M., Härtig, H.: Measuring energy consumption for short code paths using RAPL. In: GREENMETRICS 2012 (2012)
11. Intel Corporation: Intel 64 and IA-32 Architectures Software Developer's Manual (May 2012)
12. Isci, C., Martonosi, M.: Runtime power monitoring in high-end processors: Methodology and empirical data. In: 36th IEEE/ACM Int. Symp. on Microarchitecture (2003)
13. Kasichayanula, K., Terpstra, D., Luszczek, P., Tomov, S., Moore, S., Peterson, G.: Power aware computing on GPUs. In: 2012 Symp. on Application Accelerators in High-Performance Computing (2012)
14. Li, D., Byna, S., Chakradhar, S.: Energy-aware workload consolidation on GPU. In: 40th Int. Conf. on Parallel Processing Workshops, pp. 389–398 (2011)
15. McCullough, J.C., Agarwal, Y., Chandrashekar, J., Kuppuswamy, S., Snoeren, A.C., Gupta, R.K.: Evaluating the effectiveness of model-based power characterization. In: 2011 USENIX Conf. (2011)
16. Nagasaka, H., Maruyama, N., Nukada, A., Endo, T., Matsuoka, S.: Statistical power modeling of GPU kernels using performance counters. In: Int. Green Computing Conf. (IGCC), pp. 115–122 (2010)
17. Nvidia: Tesla C2075 Computing Processor Board. Board Specification (2011), http://www.nvidia.com/docs/IO/43395/BD-05880-001_v02.pdf, bD-05880-001_v02
18. Nvidia: CUBLAS Library – User Guide, version 5.0 (October 2012), http://docs.nvidia.com/cuda/pdf/CUDA_CUBLAS_Users_Guide.pdf
19. Nvidia: NVML API Reference Manual, ver. 3.295.45 (2012), http://developer.download.nvidia.com/assets/cuda/files/CUDADownloads/NVML/nvml.pdf
20. Rauber, T., Rünger, G.: Towards an energy model for modular parallel scientific applications. In: Green Computing and Communications (GreenCom), pp. 523–532 (2012)
21. Rofouei, M., Stathopoulos, T., Ryffel, S., Kaiser, W., Sarrafzadeh, M.: Energy-aware high performance computing with graphic processing units. In: Workshop on Power Aware Computing and Systems, HotPower 2008 (2008)
22. Weaver, V., Johnson, M., Kasichayanula, K., Ralph, J., Luszczek, P., Terpstra, D., Moore, S.: Measuring energy and power with PAPI. In: Int. Workshop on Power-Aware Systems and Architectures, PASA 2012 (2012)