Autotuning based on frequency scaling toward energy efficiency of blockchain algorithms on graphics processing units

Matthias Stachowski, Alexander Fiebig & Thomas Rauber

The Journal of Supercomputing
An International Journal of High-Performance Computer Design, Analysis, and Use

ISSN 0920-8542

J Supercomput
DOI 10.1007/s11227-020-03263-5

Springer

The Journal of Supercomputing
https://doi.org/10.1007/s11227-020-03263-5

Autotuning based on frequency scaling toward energy efficiency of blockchain algorithms on graphics processing units

Matthias Stachowski[1] · Alexander Fiebig[1] · Thomas Rauber[1]

Abstract
Energy-efficient computing is especially important in the field of high-performance computing (HPC) on supercomputers. Therefore, automated optimization of energy efficiency during the execution of a compute-intensive program is desirable. In this article, a framework for the automatic improvement of the energy efficiency on NVIDIA GPUs (graphics processing units) using dynamic voltage and frequency scaling is presented. As application, the mining of crypto-currencies is used, since in this area energy efficiency is of particular importance. The framework first determines the energy-optimal frequencies for each available currency on each GPU of a computer automatically. Then, the mining is started, and during a monitoring phase it is ensured that always the most profitable currency is mined on each GPU, using optimal frequencies. Tests with different GPUs show that the energy efficiency, depending on the GPU and the currency, can be increased by up to 84% compared to the usage of the default frequencies. This in turn almost doubles the mining profit.

Keywords DVFS · GPU · Blockchain · Energy · HPC

1 Introduction

Modern supercomputers provide massive computing power, but they also require large amounts of energy for computing. As an example, the current number one supercomputer in the June 2019 TOP 500 listing has 4608 computing nodes, each consisting of

✉ Matthias Stachowski
stachowski@uni-bayreuth.de

Alexander Fiebig
alexander.fiebig@uni-bayreuth.de

Thomas Rauber
rauber@uni-bayreuth.de

[1] University of Bayreuth, Bayreuth, Germany

Published online: 02 April 2020

two IBM POWER9 processors and six NVIDIA Tesla V100 GPUs. Each GPU has a maximal power consumption of 300 W. The maximal power consumption of the entire system is 13 MW, 8.29 MW of which can be attributed to the power consumption of the GPUs [1, 2]. This indicates the importance of GPUs in high-performance computing and the tremendous energy consumption of supercomputers.

The energy consumption can be influenced by the operational frequency used by the hardware, and modern CPUs (central processing unit) and GPUs (graphics processing unit) provide DVFS (dynamic voltage and frequency scaling) to reduce the energy consumption by reducing the operational frequency. However, reducing the frequency usually increases the execution time. Therefore, it is important to select a frequency that reduces the energy consumption without increasing the execution time by a significant amount. The automatic optimization of energy efficiency during the compute-intensive execution of application programs is especially desirable for GPUs due to their large power consumption.

Blockchain mining algorithms play an increasingly important role due to the use of crypto-currencies such as Bitcoin, which are used to secure financial transactions, especially in the Internet. When mining crypto-currencies, solving a mathematical puzzle earns a reward in coins, which can then be exchanged for real money. The probability to win the reward is proportional to the computing power invested, whereas the costs correspond to the energy consumption of the hardware. Blockchain mining algorithms are often executed on GPUs, since GPUs are more effective than CPUs for these algorithms.

The goal of this article is to investigate how the energy consumption of blockchain mining algorithms can be optimized for NVIDIA GPUs. For the investigation, we propose and use a framework that automates the energy reduction using a systematic exploration. The framework consists of three phases. The first two phases determine optimal frequencies for each currency and each GPU using an offline selection process and an online optimization process. The third phase is a monitoring phase, which ensures that on each GPU, the most profitable currency is mined at each point of execution time. Tests with different GPUs show that the energy efficiency, depending on the GPU and the currency, can be increased by up to 84% compared to the usage of the default frequencies. This in turn almost doubles the mining profit.

The rest of the article is structured as follows. Section 2 gives a brief introduction to blockchain technology and the underlying algorithms. Section 3 explains some technical background, e.g., how the energy measurement and frequency scaling are performed. Section 4 introduces the energy-oriented autotuning framework. Section 5 presents an experimental evaluation with different currencies for different NVIDIA GPUs. Section 6 describes and discusses related work. Section 7 concludes the article.

2 Blockchain technology

In this chapter we shortly introduce how a blockchain works in general and introduce the three main blockchain algorithms used in this article, Ethereum (ETH), Monero (XMR) and ZCash (ZEC).

2.1 Introduction to blockchains

A blockchain is a list of data (blocks) which are connected through cryptographical hashes. Each block consists of a header and a body. The body contains an amount of finished transactions and their hashes which are stored in a hash tree (Merkle Tree). The header contains the root of this tree, a time stamp and a random number (*Nonce*). Additionally, the header includes the hash value of the previous block. This ensures a non-modified belated transaction. Depending on the blockchain technology, the header can include more information. Also the hashing function used can differ.

A blockchain is managed decentrally, meaning that there are no central servers where the blockchain is stored. Instead, a peer-to-peer network is used. Each participant of this network stores the complete blockchain. If a new block is generated by a set of transactions, every node verifies these transactions locally before sending this block to the other nodes in the network. To be prepared for failures or attacks, different blockchains offer different possibilities for protection. One of them is Proof of Work (PoW).

PoW participants are called miners. A miner can only add a new block of transactions, when he solves a *Nonce*. Simplified, the miner has to solve the following equation:

$$hash(block\_data||X) < Y, \tag{1}$$

where X is the *Nonce* and Y is the given difficulty *target*. The target changes dynamically with the combined computing power of all miners to achieve the same time to solve this equation every time. The *target* corresponds to the *difficulty* to mine a block. Solving Eq. (1) is called mining a block. This procedure can only be done with a brute-force approach. If a miner mines a block, he gets a block reward [3]. This computation provides a large potential of parallelism, and GPUs can be used efficiently, as described in Sect. 3.1. However, a lot of energy may be used. Since it is almost impossible to find a block alone, miners are connected through a so-called mining pool. These pools concentrate the computing power of each miner who subscribes to this pool. The miner with the highest computing power contributed earns the most reward.

The crypto-currencies Ethereum (ETH), Monero (XMR) and ZCash (ZEC) are shortly described in the following subsection.

2.2 Overview of the algorithms of ETH, XMR and ZEC

This chapter explains the main concepts of the algorithms of the crypto-currencies ETH, XMR and ZEC used in this article. Ethereum uses the Ethash algorithm, Monero uses the CryptoNight algorithm, and ZCash uses the Equihash algorithm. These algorithms will be shortly explained in the following. A more detailed description can be found in [4].

2.2.1 ETH: the Ethash algorithm for Ethereum

Ethereum mining is based on the Ethash algorithm, also known as the Dagger-Hashimoto algorithm. The simplified flow diagram in Fig. 1 shows the main structure of the algorithm [5].

The Ethash algorithm uses a pseudo-generated data set [called DAG (directed acyclic graph)] based on all blocks generated so far by all participants and a nonce counting all confirmed transactions contributed by the specific participant. To generate a new block, the header from the previous block and the current nonce is hashed by a hashing algorithm similar to SHA-3 to get the initial 128-byte mix, also called Mix 0 (step 1 in Fig. 1). By using this Mix 0 the 128-byte site of the DAG can be determined (step 2). Then in step 3, Mix 1 is calculated based on this site and Mix 0. Steps 2 and 3 are repeated 64 times until Mix 64 is generated. In step 5, Mix 64 is compressed to a 32-byte mix, also called Mix Digest. This Mix Digest is then compared to the target threshold (also 32 bytes). If the value of Mix Digest is smaller or

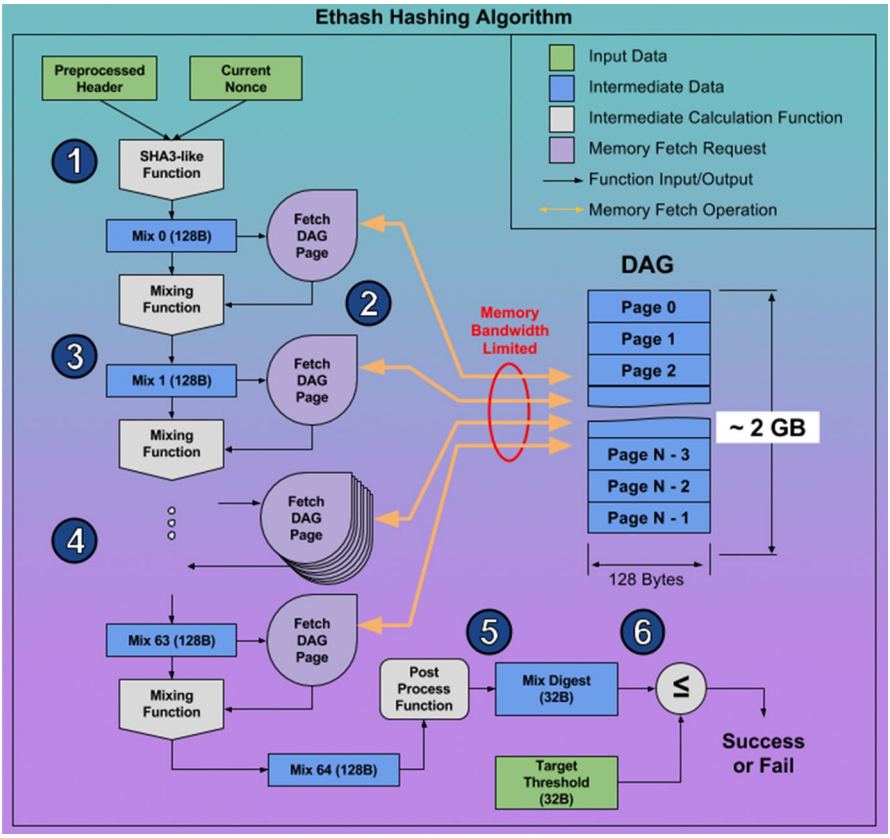

Fig. 1 Flow diagram of the Ethash algorithm used by Ethereum mit a DAG size of 2.37 GB of late 2018 [5]

equal to this target threshold, the nonce is accepted and transferred to the Ethereum network. If this is not the case, the nonce is dismissed and a new nonce is used for the comparison, which is usually a newly generated random number, see step 6.

Analyzing the different steps of the algorithm, it becomes clear why memory bandwidth is the limiting factor: Each read of a new mix needs 128 byte from the DAG. Hashing just one nonce needs 64 mixes which corresponds to 8 KB of data. But reading one site of the DAG is random, so caching the DAG in a small CPU cache would be of no benefit because reading the next site of the DAG would be probably out of the cache. When benchmarking Ethereum for this article, the DAG size was around 2.37 GB, while the biggest CPU cache was around 128 MB at this time [6].

In conclusion, the only way to improve the performance is to decrease the access time to the DAG, which is equivalent to an increase in the overall memory bandwidth.

2.2.2 XMR: the CryptoNight algorithm

CryptoNight is the algorithm used by the crypto-currency Monero [4]. The algorithm basically consists of three main steps: the scratchpad initialization, the memory-hard loop and hashing operations.

In the first step, a large scratchpad is initialized with pseudo-random data. To do so, input data are hashed with Kecccak-1600 (from which the well-known SHA-3 (Secure Hash Algorithm 3) is a subset), which results in 200 bytes of pseudo-random data. By applying AES-256 (Advanced Encryption Standard) encryption to these 200 bytes, a 2-MB buffer of pseudo-random data is seeded. Bytes 0 to 31 of the Keccak-1600 hash are used as AES key. The encryption is performed on 128-byte payloads until 2 MB is reached. The Keccack-1600 bytes 66 to 191 are used as the first payload. The next payloads are encrypted on the results of the previous ones. Finally each 128-byte payload is encrypted 10 times.

The second step, the so-called memory-hard loop, basically consists of 524,288 iterations of a simple stateful algorithm. All iterations read and write the scratchpad at pseudo-random locations. It is not possible to calculate the state of future iterations directly.

The last step performs a hashing of the entire scratchpad to produce the resulting value. The step combines the original Keccak-1600 data with the entire scratchpad. Then, the algorithm picks one of four hashing algorithms (BLACK-256, Groestl-256, JH-256 or Skein-256) and hashes the result with the selected hashing function. The resulting 256-bit hash is the final output of the CryptoNight algorithm [7]. To make this algorithms ASICS (application-specific integrated circuit) safe, the algorithm is changed slightly every 6 months [8, 9].

2.2.3 ZEC: the Equihash algorithm for ZCash

Equihash is a Proof-of-Work algorithm which is based on the generalized birthday problem and the enhanced Wagner's algorithm. The algorithm uses three parameters n, k, and d [10, 11]. The values of these parameters determine the time and memory

requirements of the algorithm. The Equihash algorithm solves a modified generalized birthday problem: Given a single list L of n-bit strings x_i with $|L| \ll 2^n$, find exactly 2^k distinct strings $x_1, x_2, \ldots, x_{2^k}$ from L such that

$$H(x_1) \oplus H(x_2) \oplus \cdots \oplus H(x_{2^k}) = 0 \tag{2}$$

and $H(x_1||x_2|| \ldots ||x_{2^k})$ has d leading zeros. H is a given hash function and $\oplus$ denotes the XOR on bit strings. ZCash uses the Equihash algorithm with $n = 200$ and $k = 9$ (d is set to zero), see Fig. 2 for an illustration. With these parameters the birthday problem has a minimum memory size of 522 MB. The algorithm also uses a seed I which is obtained by a hash transfer. V is a 160-bit nonce to be determined. The Equihash algorithm solves the modified generalized birthday problem by finding $x_1, x_2, \ldots, x_{2^k}$ with all numbers smaller than $2^{\frac{n}{(k+1)}+1}$ to solve Eq. (2) as stated above. H is a Blake2b hashing function [11]. For a detailed explanation it is recommended to read the original publication [10].

2.3 Pre-profiling ETH, XMR, ZEC

The currencies employed in the evaluation in Chapter 5 are Ethereum (ETH), Monero (XMR) and ZCash (ZEC) which are all Proof-of-Work (PoW) algorithms. Ethereum, Monero and ZCash were chosen due to their different hashing functions, which lead to different computing characteristics. Ethereum uses the Ethash-based hashing function. Monero utilizes the CryptoNight protocol, and ZCash is Equihash based. At the time of writing this paper, CryptoNight in version 7 has been available. For the mining of ETH, the *ethminer* [12] in version 0.15.0.dev11 has been used, for XMR the *xmr-stak* [13] in version 2.4.5, and for ZEC the *excavator* [14] in version 1.1.0a.

The hash function used in the blockchain of the corresponding currency is essential for the performance: To quantify this effect, the mining of different currencies has been profiled in Windows 10 with the NVIDIA-Driver in version 391.24 and the CUDA toolkit in version 9.1. Figure 3 shows the profiling results of ETH, ZEC and XMR on a NVIDIA Titan X (Pascal) with the *NVIDIA Visual Profiler*, NVVP [15]. The utilized memory bandwidth for reading and writing access, as well as the number of instructions per clock cycle (IPC), is measured. The memory bandwidth measured indicates how much a program is memory-bound, while the IPC measured indicates how much a program is compute-bound. As the miner uses several

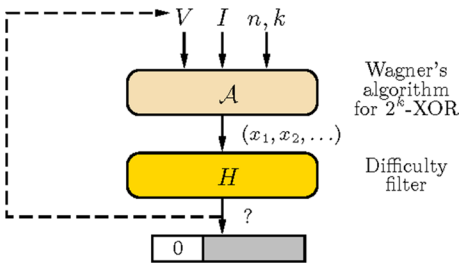

Fig. 2 Flow diagram of the Equihash algorithm used by ZCash [10]

CUDA kernels, the weighted mean of the individual kernels after the GPU time is displayed.

The memory bandwidth indicates that ETH is most strongly memory-bound among the currencies examined, followed by XMR and ZEC. The executed IPC shows that ZEC is most strongly compute-bound, followed by ETH and XMR. The stronger a currency is memory-bound, the faster and longer the hashrate will increase when the VRAM frequency is raised. Similarly, the stronger a currency is compute-bound, the faster and longer the hashrate will increase when the core frequency is raised.

3 Technical background

In this section, we will explain some technical background. This includes information about the hardware used for the experimental evaluation, the operating system, the energy measurement and the frequency adaptation.

3.1 How GPUs work in this context

GPUs are massive parallel many core processors, which have, compared to CPUs, a high amount of computing power and a large memory bandwidth. On GPUs a larger number of transistors are assigned to data processing than on CPUs, where a relative high amount of transistors is used for caches and control logic. The reason for this lies in the fact that CPUs and GPUs serve different purposes: CPUs are designed to minimize the latency for one thread, whereas GPUs are designed to maximize the throughput of all threads.

On a GPU a thread corresponds to a sequence of SIMD (Single Instruction Multiple Data) lane operations. As a result, GPUs are well suited for computations in which the same instructions are executed on different data in parallel in SIMD style.

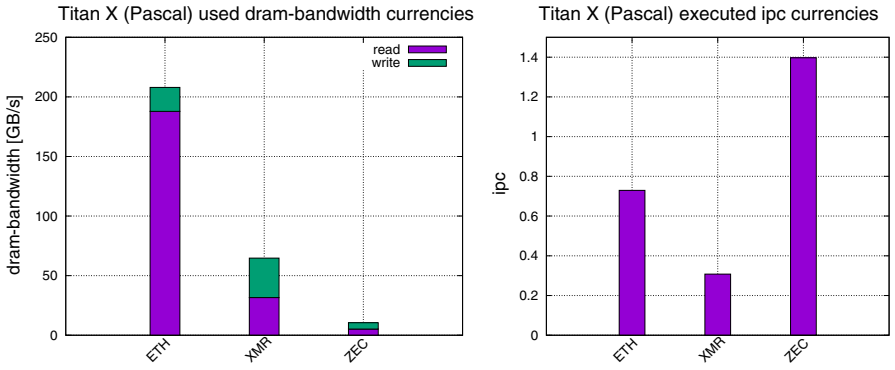

Fig. 3 Measured VRAM memory bandwidth (left) and executed IPC (right) during mining of ETH (ethminer), XMR (xmr-stak) and ZEC (excavator) on the Titan X with standard frequencies

Executing the same instructions on different data allows to hide memory access latencies by computations. This reduces the need for big caches on GPUs [16].

The hash algorithms described in Sect. 2.2 are utilized during the PoW approach described in Sect. 2.1 for the different currencies evaluated. To solve the PoW, different numbers (nonces) can be checked with the hash algorithms in parallel. This corresponds to SIMD computations suitable for GPUs (1:1 mapping of nonces to SIMD lanes). The more nonces can be checked in parallel, the faster a solution to the PoW can be found, earning the block reward. The number of checked nonces per second is called hashrate.

3.2 Energy measurement

The energy measurement is incorporated as an individual module and is activated separately for each GPU as a background process. During the energy measurement, the real-time energy consumption of the specific GPU is measured periodically via an infinite loop using the **NVML** [17] library. The energy data are reported in a file with the corresponding system time. The energy consumption for a given time frame is calculated by reading the energy values for the specific time period and calculating the mean value.

3.3 Frequency adaptation

DVFS enables us to change the operational frequencies of a computing unit dynamically. DVFS is also available for GPUs. The frequency adaptation uses the **NVML** [17] library and, depending on the operating system, the **NVAPI** [18] (Windows) or the **NV-Control X** [19] (Linux) library. The parameters are the device ID, the adjustable VRAM frequency, as well as the core frequency as an index of a vector which holds the available frequencies. There are three ways of determining a suitable frequency, depending on which APIs are supported on the specific GPU:

- **NVML** and **NVAPI/NV-Control X**: The VRAM and the core frequency can be set in the full available range of frequencies.
- **NVML only**: The core frequency can be set in a slightly restricted value range; however, the VRAM frequency cannot be set.
- **NVAPI/NV-Control X only**: The VRAM frequency can be set in the full range, and the core frequency is set in a strongly restricted range.

3.4 Hardware setup

The GPUs used to evaluate the framework are shown in Table 1 along with important hardware information. The table also displays which APIs from Sect. 3.3 are available for overclocking and underclocking, as well as the adjustable frequency area.

The CPU used is an Intel Broadwell 6950X Processor. Since all measurements run on the GPUs, the main task of the CPU is to schedule work on the GPUs and to run the operating systems.

4 Autotuning framework

For our experiments, we have developed a framework with autotuning features. The source code of the framework is available at https://github.com/UBT-AI2/dvfs_gpu.

In this section, we focus on the main modules of the autotuning framework and its implementation. The framework has been developed for a collection of GPUs attached to the same computer. The goal is to reduce the overall energy consumption of the entire system as far as possible with a stable performance rate. The framework developed is organized in three main autotuning phases: (1) an offline frequency (pre-)selection, (2) an online frequency optimization and (3) a monitoring phase. This chapter describes the different phases in Sects. 4.2, 4.3 and 4.4. As usual for autotuning, we also use search strategies to determine whether a setting is better than another setting, see Sect. 4.1.

4.1 Optimization procedure

Three different search strategies have been implemented to find the energy optimal frequencies: **Hill Climbing**, **Simulated Annealing**, and **Nelder-Mead** [20]. The optimization function maps the adjustable frequency range to the amount of hashes per Joule, obtained while mining a currency.

The target function $f : [a, b] \times [c, d] \longmapsto \mathbb{R}_0^+$ with $a, b, c, d \in \mathbb{N}$ maps the adjustable frequency range of a GPU to the amount of hashes per Joule which are achieved while mining a crypto-currency in (3) where $[a, b]$ corresponds to the range of the adjustable core frequencies and $[c, d]$ corresponds to the range of the adjustable VRAM frequencies. The frequencies are expressed as integer values in MHz. The target function is computed by:

Table 1 GPUs used for evaluation

	TITAN X	TITAN V	GTX 1080	P4000
Architecture	Pascal	Volta	Pascal	Pascal
CUDA-Cores	3584	5120	2560	1792
Memory type	12 GB GDDR5X	12 GB HBM2	8 GB GDDR5X	8 GB GDDR5
Memory bandwidth (GB/s)	480	652	320	243
Power cap (W)	250	250	180	105
NVML	Yes	Yes	No	Yes
NVAPI	Yes	Yes	Yes	No
NV-Control X	Yes	No	Yes	No
VRAM-clock range (MHz)	4005–6005	850–1050	4005–6005	3802
Core-clock range (MHz)	139–2011	139–2012	1711–2011	139–1708

$$f(vram, core) = \frac{hashrate(vram, core)}{energyconsumption(vram, core)} \tag{3}$$

The optimization procedure aims to maximize the value of f in the adjustable frequency range. Additionally, as a side condition for the optimization procedure, a minimum hashrate to adhere to can be specified. The value range for the frequency to be adjusted is determined for each GPU at program start. It depends on the APIs supported on the GPU for frequency adjustment. Depending on supported APIs, the target function (3) is zero-dimensional, one-dimensional or two-dimensional.

4.2 Offline phase

The frequency optimization is executed for each GPU in a separate CPU thread which determines the energy optimal frequency for each currency assigned to the GPU. This is performed by using the optimization procedure introduced in Sect. 4.1.

The frequency optimization itself is divided in an offline and an online phase. These two phases differ in the evaluation of the target function from Eq. (3), i.e., in determining the hashrate and energy consumption at the given frequency.

In the offline phase, the frequencies are optimized via short offline benchmarks. In this context, offline is defined as having no connection to the mining pool and a lack of network communication. The performance (hash amounts per second) of the hash algorithm is thus determined under ideal conditions, as network faults and latency are no longer a bottleneck.

During every evaluation of the target function from Eq. (3), the binary of the miner is invoked in benchmark mode for the currency to be optimized. The hashrate achieved is stored in a data structure together with the maximum energy consumption measured during the benchmarking. Information regarding frequencies used in these measurements and the time frame of this benchmarking are also saved.

The duration of an offline phase depends on the run time of a measurement and the applied optimization procedure. For the tested currencies, the measurement duration ranged between 2 and 30 s. The optimization procedures need 10–15 function evaluations, depending on their individual starting points.

This module is also suitable for every kind of benchmarks to determine the optimal frequency setting if an energy optimal setting is considered.

4.3 Online phase

During the online phase, frequencies are optimized under real conditions. The online mining with a mining pool is initiated at the beginning of the online frequency optimization phase as a background process. This background process continuously writes the hashrate currently achieved along with the corresponding system time to a log file.

To evaluate the target function, the desired frequencies must be set and the current system time saved. While waiting for a determinable time frame, which is typically between two and three minutes, the average hashrate can be read off the log file.

Moreover, the average energy consumption during this time can be determined as described in Sect. 3.2. Hashrate, energy consumption and measuring period are stored in a data structure as in the offline phase.

The result of the offline phase serves as a starting point for the online phase. The creation of a measuring point (function evaluation) takes significantly more time in the online phase compared to the offline phase. However, less function evaluations are needed as the starting point is usually already close to the optimum. Moreover, mining incomes can be earned during the online phase, as real mining is running in the background.

4.4 Monitoring

Similar to the frequency optimization phase, the monitoring phase is also executed by a separate CPU thread for each GPU. The monitoring thread executes an infinite loop and is started when there are optimal frequencies available for all currencies on the corresponding GPU. This is the case when the frequency optimization phase is completed for all GPUs of the associated GPU group (see Sect. 4.5).

The monitoring phase is responsible for the periodic calculation of energy costs and mining revenues. The energy costs in Euro per second are computed as follows:

$$energy\_cost\left[\frac{\text{Euro}}{\text{s}}\right] = energy\_consumption\,[\text{Ws}] \cdot \frac{energy\_cost\left[\frac{\text{Euro}}{\text{kWh}}\right]}{1000 \cdot 3600} \tag{4}$$

To compute the mining revenue in Euro per second the following formula is used where *hr* is the abbreviation for hashrate [21]:

$$mining\_reward = \frac{user\_hr}{net\_hr} \cdot \frac{1}{block\_time} \cdot block\_reward \cdot stock\_price \tag{5}$$

Here the user hashrate (*user_hr*) is the hashrate that is obtained by the miner and the network hashrate (*net_hr*) is the total hashrate of all miners of the currency. Moreover, *block_time* is the average time needed to mine a new block, *block_reward* is the reward of the mined currency that a miner receives when finding a new block, and *stock_price* is the stock price for one unit of the currency in Euros.

The average block time is given by the currency (e.g., 15 s for Ethereum) and should remain constant, independently from the miner and the current network hashrate. For this reason the block difficulty is continuously adjusted according to the current network hashrate. If it increases (decreases), the block difficulty (*block_df*) will rise (fall). Thus:

$$block\_df = net\_hr \cdot block\_time \tag{6}$$

In the framework, *stock_price* from *CryptoCompare* [22], and *net_hr*, *block_time* as well as *block_reward* from *WhatToMine* [23] are retrieved over their REST-APIs. Subtracting energy costs from the mining revenue gives the profit:

$$mining\_profit = mining\_reward - energy\_cost \tag{7}$$

If the currently mined currency is no longer the most profitable one after the recalculation of energy costs and mining revenues, the background mining of this currency is terminated and the mining of a new, more profitable currency is initiated.

Subsequently, energy-efficient frequencies for the newly mined currency are determined again, trying to find even better frequencies. The search method is identical to the online frequency optimization method in Sect. 4.3. The starting point of the search is the previously used frequency for the particular currency. The previous optimization result is updated with the result of the new optimization.

4.5 Handling multiple GPUs

The device information of the hardware system to be used is read for all GPUs. The GPUs are subdivided into groups for the offline and the online frequency optimization. Identical GPUs will be allocated to the same group. Each GPU group must be able to optimize the frequencies for all available currencies. After the program is started, existing optimization results from previous measurements can be incorporated for the individual groups. This will reduce the optimization effort needed. If the optimization result of a group contains values for all currencies, the complete frequency optimization phase is skipped for the particular group. The currencies with no optimization result available are divided onto the individual GPUs of the group for frequency optimization.

The reason behind the arrangement of GPUs into groups is that the optimum frequencies for the individual currencies are considered to be identical for identical GPUs. Thus, the optimization for the individual currencies can be performed in a parallel fashion within each group, exchanging optimization results between the GPUs of the group. This results in an acceleration of the frequency optimization phase.

Subsequently, a thread is started for every GPU. These threads run for the complete length of the program, i.e., the complete frequency optimization and monitoring phase are executed separately for each GPU in a separate thread. In the monitoring phase, the threads run in an infinite loop until the user stops the program. At program termination the optimization results that have been updated during the monitoring phase are saved as a JSON file. These results contain information about optimal frequencies, hashrates and energy consumption for each currency on each GPU.

5 Experimental evaluation

In this section, we evaluate the framework introduced in Sect. 4. For the evaluation, the framework is tested with different GPUs and different currencies as introduced in Sects. 2.3 and 3.4.

5.1 Energy optimum ETH-Ethash

In order to determine the energy optimum, i.e., the maximum number of hashes per Joule, the hashrates and the corresponding energy consumption are measured for all adjustable frequencies on the individual GPU.

Figure 4 shows the results of the measurement on the Titan X and Titan V for ETH. It can be seen that ETH benefits most from higher VRAM frequencies compared to XMR and ZEC, confirming the profiling result in Sect. 2.3. The energy optimum is usually located in mid-range core frequencies and mid-range to high-range VRAM frequencies.

To detect the efficiency increase, the values at optimum frequencies are compared with those at default frequencies (see Table 2). The default frequencies are

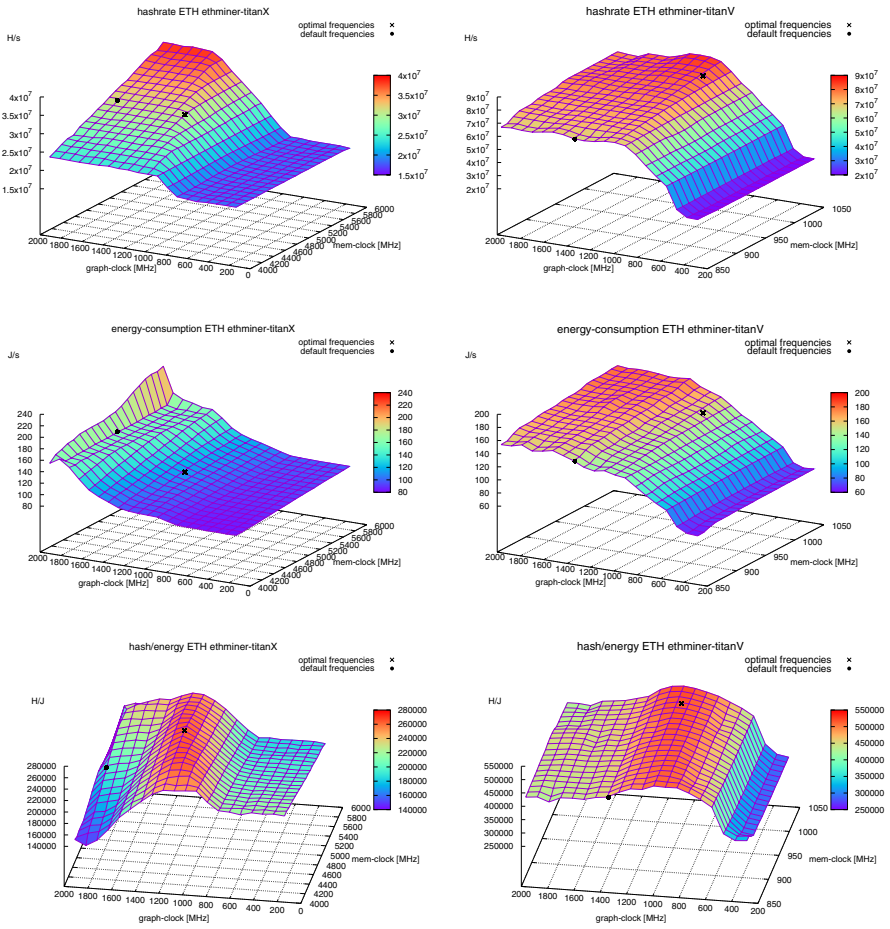

Fig. 4 ETH: hashrate (above), energy consumption (middle) and hashes per Joule (below) for all adjustable frequencies on the Titan X (left column) and the Titan V (right column)

determined by starting the miner on the corresponding GPU and by observing the frequencies adjusted by the NVIDIA-Driver.

5.2 Energy optimum XMR-CryptoNight

As can be seen in Fig. 5, XMR behaves similar to ETH. However, as XMR is less compute-bound and less memory-bound than ETH (see Fig. 3), the increase of the hashrate while raising the frequency level produces a flatter slope. It is also noteworthy that XMR requires relatively low amounts of energy. The efficiency gain via the optimization of frequencies is summarized in Table 3.

Unlike other currencies, the mining of XMR is also worthwhile on CPUs. A possible reason for this is that the higher amount of computing units does not push a GPU to the limits of its capacity, as the low IPC values demonstrate. This also explains the low demand for energy.

5.3 Energy optimum ZEC-Equihash

In comparison with ETH and XMR, ZEC is more compute-bound, as visible in Fig. 3. Hence, the energy optimum is usually located at lower VRAM-clock rates and slightly higher core-clock rates (see Fig. 6). For this reason, mining of ZEC on the Titan V is also not very efficient, as this GPU is characterized by a fast HBM2 memory which cannot be utilized by the ZEC algorithm very well. Moreover, due to the high IPC value (see also Fig. 3) and the corresponding compute-bound characteristic, ZEC requires a higher energy demand because of a high exploitation of the CUDA cores. Table 4 indicates the efficiency gain for ZEC when using optimal frequencies.

Table 2 ETH: optimal versus default frequencies on different GPUs

Currency: ETH	TITAN X	TITAN V	GTX 1080	Quadro P4000
Optimal frequencies				
VRAM-Clock	4955	1010	5805	3802
Core-Clock	1151	1035	1711	1088
Hashrate	31,119,928	82,603,192	25,339,050	26,496,646
Power	113.095	155.207	156.29	77.036
Hashes/Joule	275,166	532,213	162,128	343,951
Default frequencies				
VRAM-Clock	5005	850	5005	3802
Core-Clock	1822	1335	1885	1695
Hashrate	31,532,634	67,375,607	21,387,776	26,630,501
Power	161.008	147.519	135.96	104.775
Hashes/Joule	195,845	456,725	157,309	254,168
Efficiency gain (%)	40.5	16.53	3.06	35.32

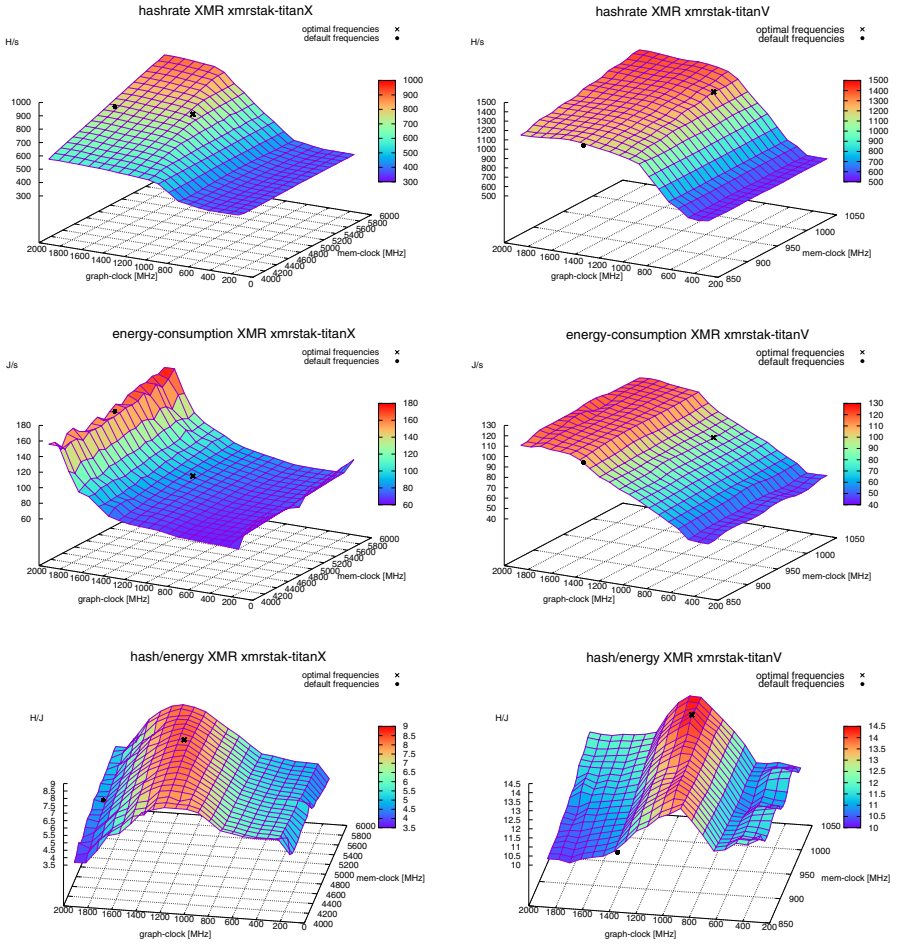

Fig. 5 XMR: hashrate (top), energy usage (middle) and hashes per Joule (below) at all adjustable frequencies on the Titan X (left column) and the Titan V (right column)

5.4 Search strategies evaluation

In this section, we will look at the different optimization algorithms (see Sect. 4.1), which are used during the *frequency optimization phases* (see Sects. 4.2 and 4.3). To do so, we will observe both a 2D optimization (VRAM frequency, core frequency) on the Titan X and a 1D optimization (core frequency) on the Quadro P4000 since the Quadro does not allow us to change the VRAM frequency. ZEC will be used as currency.

Every algorithm (Hill Climbing, Simulated Annealing and Nelder-Mead) is executed three times with different starting frequencies. The different starting points are the maximum, the minimum and the middle frequency. Furthermore all

Table 3 XMR: optimal versus default frequencies on the different GPUs

Currency: XMR	TITAN X	TITAN V	GTX 1080	Quadro P4000
Optimal frequencies				
VRAM-Clock	5155	1010	5502	3802
Core-Clock	1202	1035	1811	999
Hashrate	744.5	1277.6	581.0	554.1
Power	86.42	88.115	85.51	45.589
Hashes/Joule	8.6149	14.4992	6.79453	12.1542
Default frequencies				
VRAM-Clock	5005	850	5005	3802
Core-Clock	1847	1335	1911	1708
Hashrate	753.5	1178.2	523.8	593.9
Power	161.02	106.495	126.34	73.179
Hashes/Joule	4.67954	11.0634	4.14596	8.11572
Efficiency gain (%)	84.1	31.06	63.88	49.76

algorithms are also evaluated with a minimum hashrate as constraint. The maximum number of iterations is set to six for all tests.

The following sections introduce the optimization process and evaluate the performance of the individual algorithms. The performance of the optimization is measured with the following two criteria:

- deviation between the estimated optimum and the real optimum and
- required number of function evaluations to find the best frequency value.

5.4.1 Performance of Hill Climbing

The results of optimizing with Hill Climbing are summarized in Table 5. The first table row shows the optimum found by an exhaustive search (see Table 4). The second table row lists the energy efficiency (number of hashes per Joule) obtained with the associated frequencies in brackets (VRAM frequency, core frequency) for the different starting points. The third table row displays the required number of function evaluations to find the best energy efficiency, as well as the total number of function evaluations until termination in brackets. Both are shown columnwise for the Titan X (2D optimization) and the Quadro P4000 (1D optimization).

Figure 7 shows the associated optimization procedure of Hill Climbing by following the additional black lines. The focus of these figures is more the amount of iterations than the exact path of the Hill Climbing approximation itself. The algorithm process is always shown in combination with the function to be optimized at different starting points for both GPUs.

Amid all starting points, Hill Climbing attains a good result value near the optimum on both GPUs. On average a result value of 3.98 H/J is found on the Titan X. The optimum from Table 4 lies at 4.13 H/J. On the Quadro P4000, the result value of 3.31 H/J found on average is almost identical to the optimum from Table 4, which

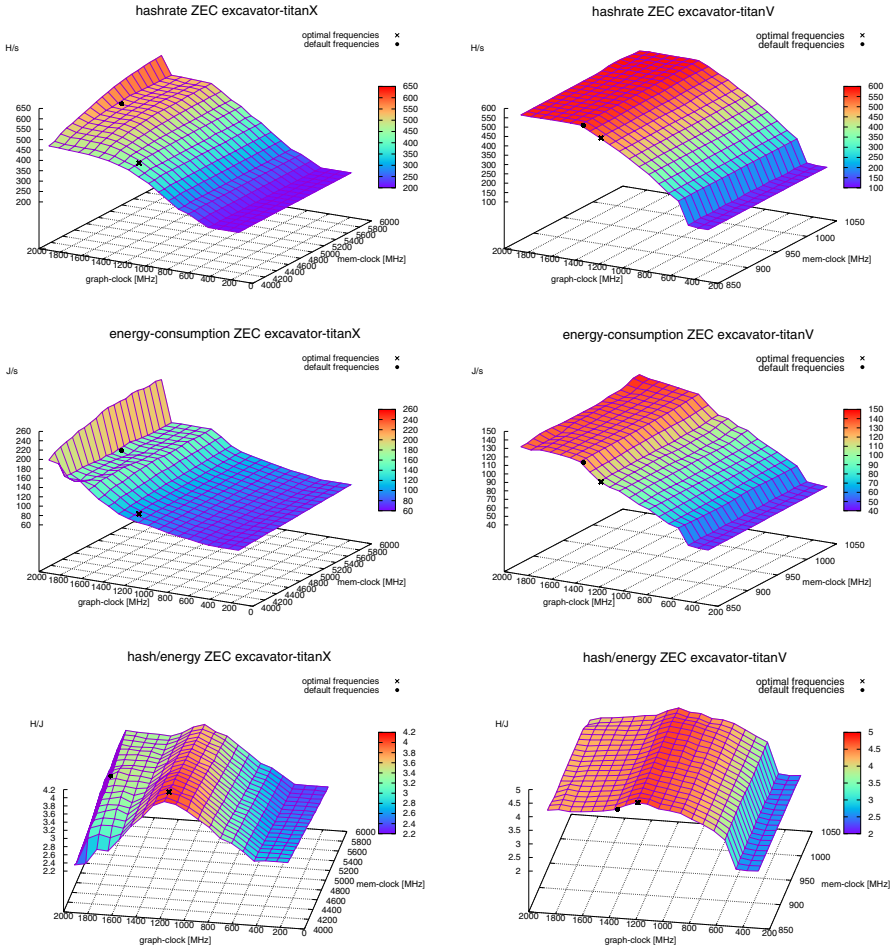

Fig. 6 ZEC: hashrate (above), energy consumption (middle) and hashes per Joule (below) at all adjustable frequencies on the Titan X (left column) and the Titan V (right column)

is 3.34 H/J. These small deviations can be explained by measurement inaccuracies. In general, it is easier to find the optimum at a 1D optimization like on the Quadro P4000. On average it requires five function evaluations to find the best value on the Quadro P4000 and 15 evaluations on the Titan X.

5.4.2 Performance of Simulated Annealing

The performance of Simulated Annealing is similar to that of Hill Climbing on both GPUs. Yet, it is not necessary to escape local optima with Simulated Annealing, since they do not exist. Table 6 shows the result and the required amount of function evaluations for the optimization. The table structure is identical to that of Table 5 explained in Sect. 5.4.1. The corresponding optimization

Table 4 ZEC: optimal versus default frequencies on the different GPUs

Currency: ZEC	TITAN X	TITAN V	GTX 1080	Quadro P4000
Optimal frequencies				
VRAM-Clock	4155	850	5555	3802
Core-Clock	1151	1185	1861	1025
Hashrate	433.525144	525.599705	432.495997	191.152922
Power	104.826	109.25	129.22	57.176
Hashes/Joule	4.13566	4.81098	3.34697	3.34324
Default frequencies				
VRAM-Clock	5005	850	5005	3802
Core-Clock	1784	1335	1885	1708
Hashrate	538.831065	579.105294	395.216173	219.175340
Power	159.64	128.46	156.30	100.79
Hashes/Joule	3.37529	4.50806	2.52857	2.17457
Efficiency gain (%)	22.53	6.72	32.37	53.74

Table 5 Hill Climbing: result and number of function evaluations of optimization with starting points minimum, medial and maximum frequencies

Hill Climbing ZEC	TITAN X	Quadro P4000
Optimum	4.13566 (4155, 1151)	3.34324 (3802, 1025)
Result		
Start-Min	3.97659 (4006, 1126)	3.31102 (3802, 961)
Start-Mid	4.02499 (4248, 1177)	3.31719 (3802, 1088)
Start-Max	3.94662 (4024, 1265)	3.31502 (3802, 999)
Average	3.98273 (4093, 1189)	3.31438 (3802, 1016)
Evaluations		
Start-Min	8 (19)	6 (9)
Start-Mid	18 (23)	3 (6)
Start-Max	18 (19)	6 (8)
Average	14.66 (20.33)	5 (7.66)

procedures at different starting points are shown in Fig. 8. Again, the exact paths shown as black lines are not as important as the amount of iteration of the Simulated Annealing needs to find a point near the optimum.

On average a value of 4 H/J is found on the Titan X. This is equivalent to a deviation of 0.13 H/J from the optimum 4.13 H/J in Table 4. Aside from measurement inaccuracies, the maximal value is also found on the Quadro P4000. The average number of function evaluations is identical to that of Hill Climbing, being 15 on the Titan X and five on the Quadro P4000. This is in part due to the fact that Simulated Annealing uses the same pattern as Hill Climbing to explore new solution candidates.

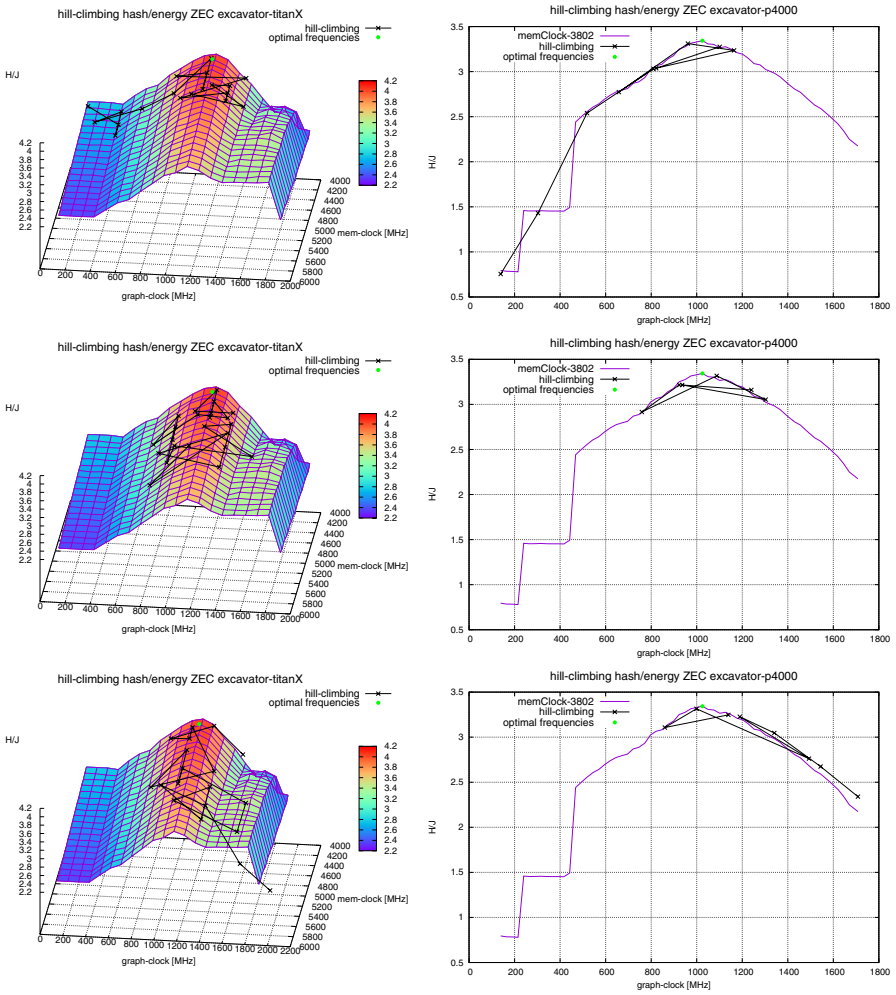

Fig. 7 Hill Climbing: frequency optimization procedure of ZEC with starting points of minimal (above), medial (middle) and maximal (below) frequencies on the Titan X (left column) and the Quadro P4000 (right column)

5.4.3 Performance of Nelder-Mead

The optimization with the Nelder-Mead procedure performs slightly worse than with Hill Climbing and Simulated Annealing. The values obtained and the required number of function evaluations are summarized in Table 7. The table structure is already described in Sect. 5.4.1. Figure 9 shows the corresponding optimization procedures at different starting frequencies. The black lines in this figure represent the path of Nelder-Mead. More important than the exact path is the amount of iterations Nelder-Mead uses to get near the minimum which is represented by the green dot.

Table 6 Simulated Annealing: result and amount of function evaluations of optimization of ZEC, starting with minimal, medial and maximal frequencies

Simulated Annealing ZEC	TITAN X	Quadro P4000
Optimum	4.13566 (4155, 1151)	3.34324 (3802, 1025)
Result		
Start-Min	4.02485 (4127, 1126)	3.32891 (3802, 987)
Start-Mid	4.05375 (4036, 1189)	3.31112 (3802, 1063)
Start-Max	3.9611 (4091, 1126)	3.32744 (3802, 1037)
Average	4.013 (4085, 1147)	3.32249 (3802, 1029)
Evaluations		
Start-Min	9 (16)	10 (14)
Start-Mid	12 (19)	4 (13)
Start-Max	24 (26)	2 (15)
Average	15 (20.33)	5.33 (14)

The energy efficiency averages at 3.88 H/J on the Titan X, corresponding to a deviation of 0.25 H/J from the optimum in Table 4. Especially when starting with maximum frequency, the VRAM frequency area is not explored enough and hence only a value of 3.72 H/J is found. As with Hill Climbing and Simulated Annealing and irrespective of measurement inaccuracies, the maximum value is also found on the Quadro P4000. In order to find the best value, the Nelder-Mead procedure needs on average ten function evaluations on both the Titan X and the Quadro P4000. This makes the Nelder-Mead procedure faster on the Titan X (10 vs. 15 function evaluation), but at the same time slower than Hill Climbing and Simulated Annealing on the Quadro P4000 (10 vs. 5 function evaluations).

In general, the Nelder-Mead procedure is relatively independent of the dimension when exploring new solution candidates. Though the simplex has more or less points depending on the dimension, the worst point of the simplex must be substituted or the simplex must be compressed as a whole in each iteration [24]. Only when calculating the initial simplex or when compressing the simplex the dimension has an influence on the number of function evaluations.

5.4.4 Optimization with constraints

To evaluate an optimization under constraints the three different algorithms are executed with a minimum target hashrate of 80% (95%) of the maximum hashrate on the Titan X (Quadro P4000). The starting frequencies must always be at the maximum for optimization with minimum hashrates, as the absolute value of applicable hashrates is calculated using the measurement values at maximum frequencies.

The results and the required number of function evaluations for the different algorithms are shown in Table 8. The table structure is similar to that described in Sect. 5.4.1. However, instead of different starting points, the different algorithms are displayed. Figure 10 shows the optimization procedures corresponding to the table. The purple grid (Titan X) or the green line (Quadro P4000) marks the function area where the constraint of the applicable hashrate is satisfied. The

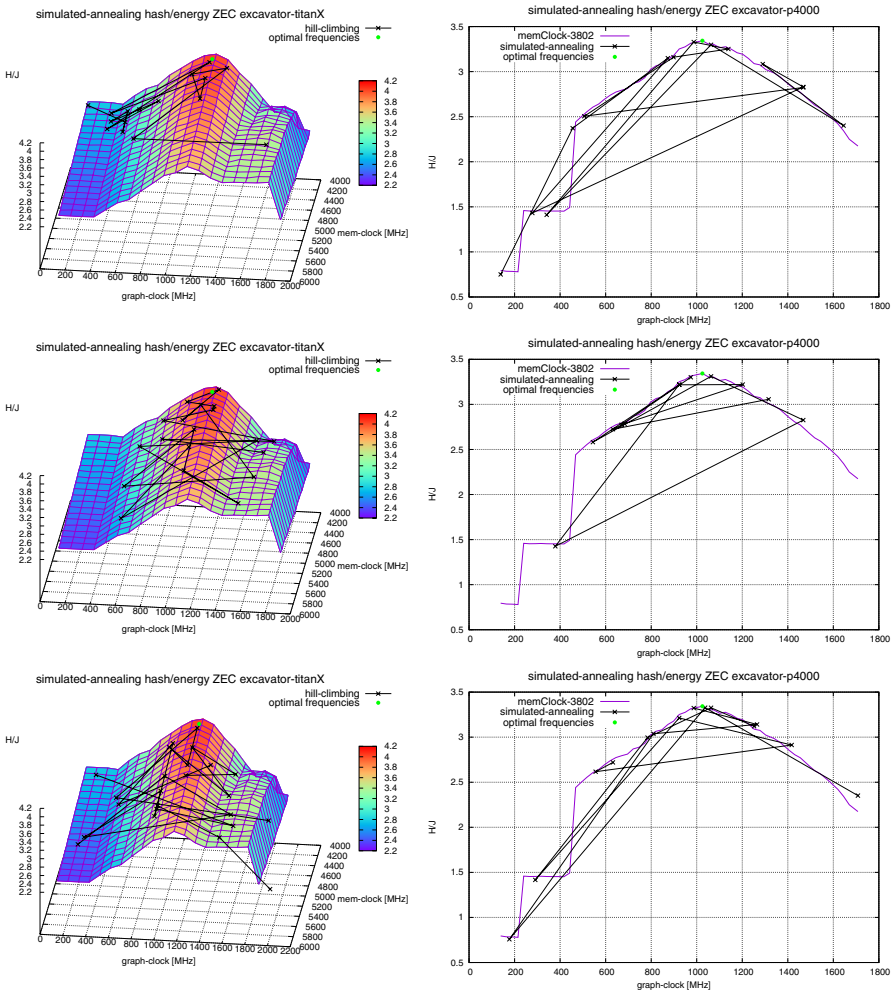

Fig. 8 Simulated Annealing: frequency optimization procedure of ZEC with starting points of minimal (above), medial (middle) and maximal (below) frequencies on the Titan X (left column) and the Quadro P4000 (right column)

best value of energy efficiency on this grid or line is the result. The optimum is subject to measurement fluctuations and is around 3.6 H/J on the Titan X and 3.05 H/J on the Quadro P4000.

The results of the different algorithms are relatively similar. Nonetheless, the Nelder-Mead procedure provides slightly worse values. The number of function evaluations needed to find the best value behaves similar to an optimization without a constraint. The Nelder-Mead procedure needs less function evaluations than Hill Climbing and Simulating Annealing during 2D optimization on the Titan X, but more function evaluations than these procedures during 1D optimization on the Quadro P4000.

Table 7 Nelder-Mead: result and number of function evaluations from optimizing the ZEC, starting with minimal, medial and maximal frequencies

Nelder-Mead ZEC	TITAN X	Quadro P4000
Optimum	4.13566 (4155, 1151)	3.34324 (3802, 1025)
Result		
Start-Min	4.01787 (4050, 1177)	3.33445 (3802, 1025)
Start-Mid	3.90598 (4732, 1151)	3.33988 (3802, 1037)
Start-Max	3.72097 (5549, 1164)	3.34415 (3802, 1037)
Average	3.8816 (4777, 1164)	3.3395 (3802, 1029)
Evaluations		
Start-Min	4 (12)	5 (12)
Start-Mid	16 (18)	15 (16)
Start-Max	11 (15)	11 (14)
Average	10.33 (15)	10.33 (10.66)

5.5 Monitoring

To evaluate the monitoring phase, the framework has been executed on a computer with all four GPUs from Table 1 attached in a time frame from 04.10.2018 20:00h till 07.10.2018 20:00h. The currencies used were ETH, XMR and ZEC, as well as some new, less popular currencies Lux-Coin, Raven, Bitcore and Vertcoin (LUX, RVN, BTX and VTC), so-called Altcoins. Those were added for this evaluation due to their popularity at the time. For Altcoins the *ccminer* [25] in version 2.3 has been used. The energy costs were set to 0.1 Euro/kWh.

Figure 11 shows the calculated mining profit at energy optimal frequencies for every currency on the individual GPUs. In each case only the most profitable currency is mined. Figure 12 shows the earnings obtained and the energy costs, as well as the resulting profits for all GPUs individually and overall.

The figure indicates that the Titan V draws the highest profits, followed by the Titan X. Here, the GTX 1080 and Quadro P4000 lie on average. The energy costs are quite similar for the Titan V and the Titan X, followed by the GTX 1080 and the Quadro P4000, the latter being the most economical one. The currency ETH is predominantly mined on the Titan V, Titan X and Quadro P4000, while LUX is mostly mined on the GTX 1080.

6 Related work

The effect of DVFS on the energy consumption of CPUs and GPUs has been explored by several research papers, see [1, 26] for an overview. In [27] different technologies for DVFS on GPUs are introduced and compared. In [28] the effect of DVFS is studied on a NVIDIA Geforce GTX 560 Ti using various sample programs. Both core and VRAM frequencies as well as core and VRAM voltages are manually

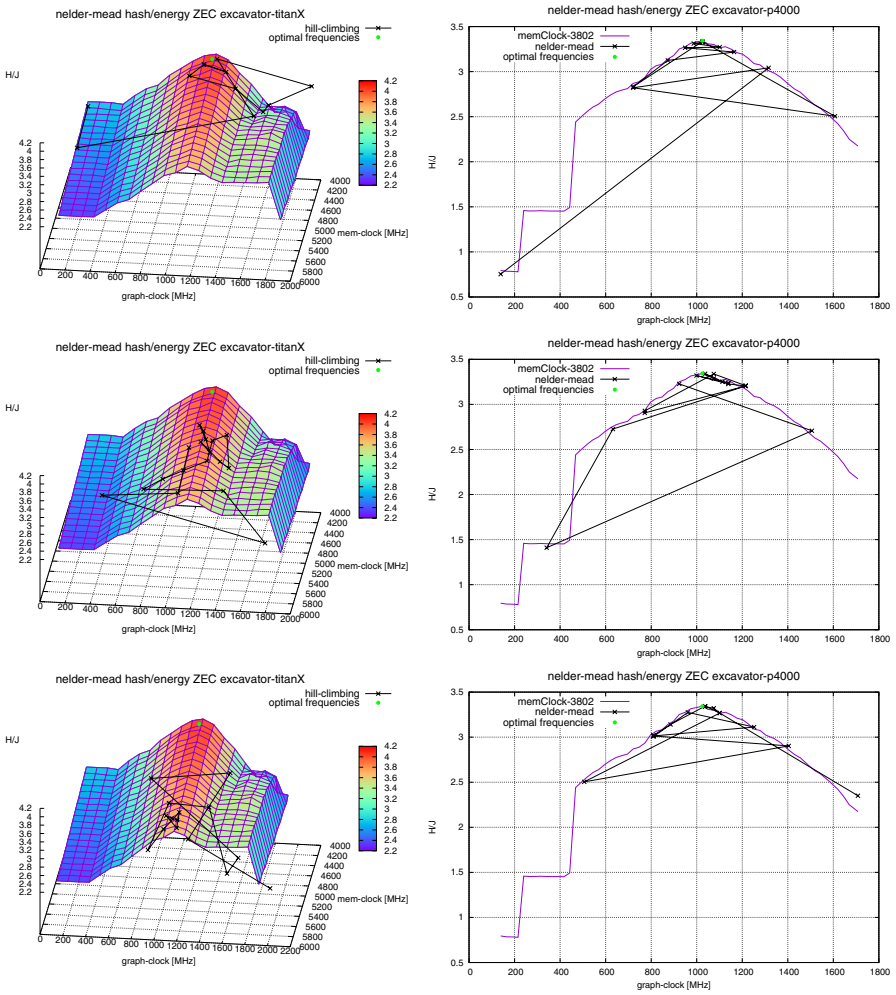

Fig. 9 Nelder-Mead: frequency optimization procedure of ZEC with starting points of minimal (above), medial (middle) and maximal (below) frequencies on the Titan X (left column) and the Quadro P4000 (right column)

adjusted using the tools *NVIDIA Inspector* and *MSI Afterburner*. This paper was the inspiration for our work. Our contribution is the dynamic setting of the frequencies and the determination of the optimal frequencies. In [29] the effect of DVFS on GPU and CPU is compared using matrix calculation as example. Cameirinha Diogo Mineiro [30] introduces a technology to reduce the energy consumption during the run time of GPU programs using DVFS and monitoring of the memory bandwidth currently observed. Bishwajit et al. [31] presents models to forecast the energy consumption of a GPU when using different core and VRAM frequencies. The models are trained using machine learning techniques and measurement data from different

Table 8 Result and number of function evaluations of ZEC optimization with different algorithms under the constraint of a minimally applicable hashrate

Constrained optimization ZEC	TITAN X	Quadro P4000
Constraint	80% hashrate	95% hashrate
Optimum	ca. 3.6 (4755, 1379)	ca. 3.05 (3802, 1303)
Result		
Hill Climbing	3.44178 (5421, 1430)	3.05715 (3802, 1341)
Simulated Annealing	3.48682 (4749, 1430)	3.0806 (3802, 1316)
Nelder-Mead	3.3656 (4836, 1468)	3.00388 (3802, 1328)
Evaluations		
Hill Climbing	12 (22)	3 (6)
Simulated Annealing	18 (21)	2 (9)
Nelder-Mead	6 (16)	8 (12)

applications. In [32], these kinds of models are used to increase the energy efficiency of mobile video games. The trained models are then used in a power management system which adjusts CPU and GPU frequencies at run time. In [33] a general overview and a categorization of various autotuning techniques are given.

There exists commercial software in the field of mining, named *Awesome Miner* [34]. This software is able to manually issue a profile with GPU frequencies, hashrates and energy consumption for every GPU and currency. The mining profit is calculated based on these profiles and their equivalent coin statistics. The most profitable currency is then mined.

7 Summary and conclusion

In this article, we have presented an autotuning framework to augment the energy efficiency on NVIDIA GPUs. Special attention has been given to the application of the framework for the mining of crypto-currencies.

The framework has been applied in a manner so that several GPUs of a computer and as many parameters as possible can be adapted using configuration data. The program procedure is divided into a frequency optimization and a profit monitoring phase.

During the frequency optimization phase, the frequency optimization occurs simultaneously on all specified GPUs. Yet, for each GPU every available currency must be optimized individually. The concept of GPU groups for identical GPUs solves this issue and allows for a division of currencies onto the different GPUs of a group. In order to permit frequency adjustments in a large enough area and allow for energy demand measurements on Windows and Linux, three NVIDIA-specific libraries (*NVML*, *NVAPI*, *NV-Control X*) were necessary. The optimization itself is based on three different optimization algorithms (Hill Climbing, Simulated

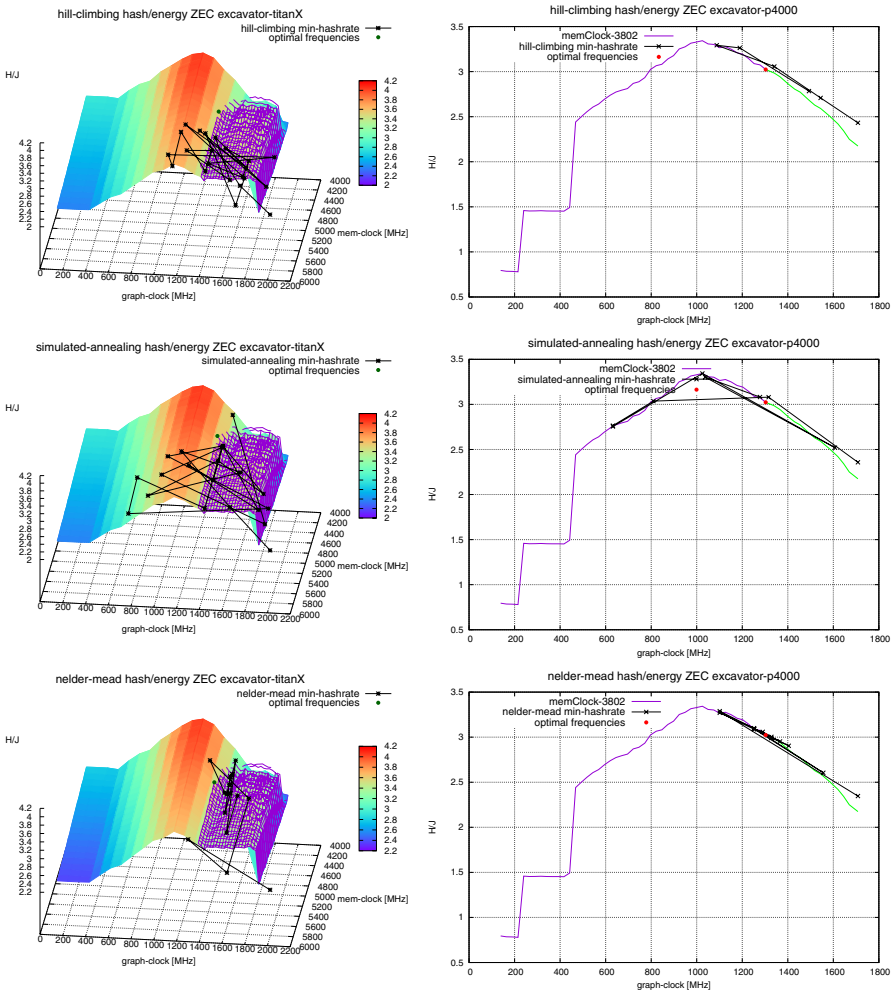

Fig. 10 Frequency optimization procedure of ZEC with Hill Climbing (above), Simulated Annealing (middle) and Nelder-Mead (below) with the constraint of a minimally applicable hashrate on the Titan X (left column) and the Quadro P4000 (right column)

Annealing, Nelder-Mead). These are employed in an offline phase based on short benchmarks and an online phase during which the mining with mining pools is already running. At the end of the frequency optimization phase, energy optimal frequencies are made known on all GPUs for all available currencies. In the following profit monitoring phase, energy costs and mining revenues at optimal frequencies are calculated and mining of the most profitable currency is initiated. E The energy consumption and mining revenues are periodically updated while taking into account current stock prices and hashrates. If the presently mined currency

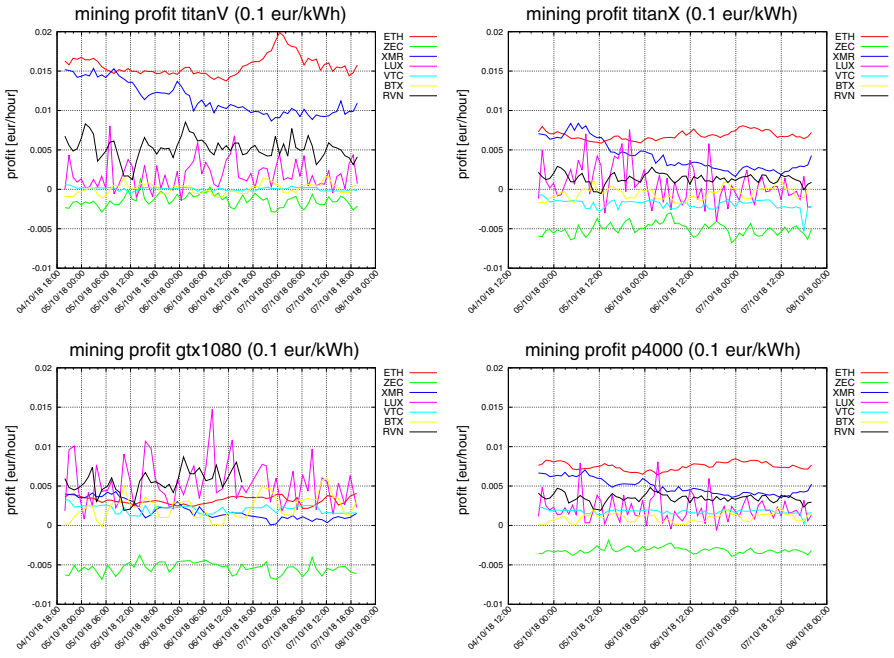

Fig. 11 Calculated mining profits of the available currencies on the individual GPUs of a computer during the monitoring phase

is not the most profitable one anymore, it is substituted followed by a frequency re-optimization.

The framework has been evaluated using different GPUs and currencies. First, the energy optimal frequencies were determined and the energy efficiency was compared with the optimum frequencies and the frequencies used by the NVIDIA driver. Depending on GPU used and the currency, an efficiency increase of up to 84% could be obtained. In the following, the different optimization algorithms were evaluated using the optimum found and the required number of function evaluations. Finally, the mining revenues calculated in the profit monitoring phase for available currencies were researched on a computer with four GPUs for a longer time period.

Our new contribution is the development of an easy-to-use open-source framework which allows to start program binaries which then are automatically adjusted to the best energy-efficient GPU setting. Until our publication there was no open-source project which was able to adjust the frequencies automatically. We used mining algorithms for evaluation because of the high energy consumption. But, it is also possible to run our framework with other GPU implementations like weather simulations. The criterium of the hashrate would be replaced with some other application specific or general criterium like the inverse run time.

For future work the voltages of the GPUs should also be considered. But, since our free available APIs **NVAPI** and **NV-Control X** do not provide this functionality, we were not able to change the voltages.

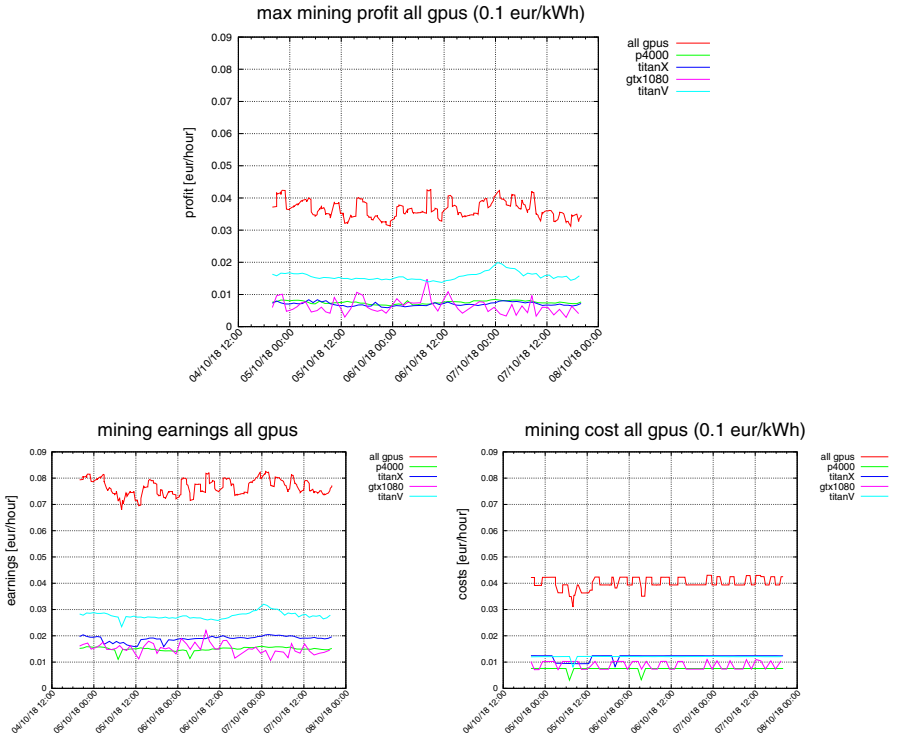

Fig. 12 Calculated mining profits (above), mining earnings (below left) and energy costs (below right) of the most profitable currency, respectively, on all GPUs of a computer during profit monitoring

Acknowledgements Open Access funding provided by Projekt DEAL.

References

1. TOP500.org (2018) TOP500 List November 2018. https://www.top500.org/lists/2018/11/. Accessed 18 Mar 2020
2. Oak Ridge National Laboratory (2018) SUMMIT. https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/. Accessed 18 Mar 2020
3. Konstantopoulos G (2017) Understanding blockchain fundamentals. https://medium.com/loom-network/search?q=Understanding%20Blockchain%20Fundamentals. Accessed 18 Mar 2020
4. Bashir I (2017) Mastering blockchain. Packt Publishing Ltd., ISBN: 978-1-78712-544-5

5. Unknown (2018) ETHASH. https://miningbitcoinguide.com/mining/sposoby/ethash. Accessed 18 Mar 2020
6. Constantin V (2018) ETHASH. https://cryptomonday.de/wie-funktioniert-mining-in-ethereum/. Accessed 18 Mar 2020
7. Cavicchioli M (2018) CryptoNight. https://monerodocs.org/proof-of-work/cryptonight/. Accessed 18 Mar 2020
8. Dölle M (2018) Ende der Grafikkarten- Ära: 8000 ASIC-Miner für Zcash, Bitcoin Gold & Co. https://www.heise.de/newsticker/meldung/Ende-der-Grafikkarten-Aera-8000-ASIC-Miner-fuer-Zcash-Bitcoin-Gold-Co-4091821.html. Accessed 18 Mar 2020
9. Vorick D (2018) The state of cryptocurrency mining. https://blog.sia.tech/the-state-of-cryptocurrency-mining-538004a37f9b. Accessed 18 Mar 2020
10. Biryukov A, Khovratovich D (2018) Equihash: asymmetric proof-of- work based on the generalized birthday problem (full version). https://orbilu.uni.lu/bitstream/10993/22277/2/946.pdf. Accessed 18 Mar 2020
11. Cavicchioli M (2018) Proof of work algorithms: Blake2b, Equihash, Tensority and X16R & S. https://en.cryptonomist.ch/2019/07/28/mining-algorithms-proof-of-work-2/. Accessed 18 Mar 2020
12. Oberhumer S (2018) ethminer. https://github.com/ethereum-mining/ethminer. Accessed 18 Mar 2020
13. fireice-uk. xmr-stak (2018) https://github.com/fireice-uk/xmr-stak. Accessed 18 Mar 2020
14. Nicehash (2018) excavator. https://github.com/nicehash/excavator. Accessed 18 Mar 2020
15. NVIDIA (2018) NVIDIA Visual Profiler. https://developer.nvidia.com/nvidia-visual-profiler. Accessed 18 Mar 2020
16. NVIDIA (2019) CUDA C++ programming guide. https://docs.nvidia.com/cuda/cuda-c-programming-guide. Accessed 18 Mar 2020
17. NVIDIA (2018) NVIDIA Management Library (NVML). https://developer.nvidia.com/nvidia-management-library-nvml. Accessed 18 Mar 2020
18. NVIDIA (2018) NVAPI. https://developer.nvidia.com/nvapi. Accessed 18 Mar 2020
19. NVIDIA (2018) NV-CONTROL X Extension-API specification v 1.6. https://github.com/NVIDIA/nvidia-settings/blob/master/doc/NV-CONTROL-API.txt. Accessed 18 Mar 2020
20. ASL ETHZ (2018) Numerical methods. https://github.com/ethz-asl/numerical_methods. Accessed 18 Mar 2020
21. Forum Bitcoin (2017) Ethereum (ETH) mining profit formula. https://bitcointalk.org/index.php?topic=2262328.0. Accessed 18 Mar 2020
22. CryptoCompare.com (2018) CryptoCompare API. https://www.cryptocompare.com/api/#-api-data-price-. Accessed 18 Mar 2020
23. whattomine.com (2018) Coin calculators. https://whattomine.com/calculators. Accessed 18 Mar 2020
24. Cheng J (2018) Numerical optimization.http://www.jade-cheng.com/au/coalhmm/optimization/. Accessed 18 Mar 2020
25. Pruvot T (2018) ccminer. https://github.com/tpruvot/ccminer. Accessed 18 Mar 2020
26. Rauber T et al (2014) Energy measurement, modeling, and prediction for processors with frequency scaling. J Supercomput 70(3):1451–1476. https://doi.org/10.1007/s11227-014-1236-4
27. Mishra A, Khare N (2015) Analysis of DVFS techniques for improving the GPU energy efficiency. Open J Energy Effic 4:77–86
28. Mei X, Yung LS, Zhao K, Chu X (2013) A measurement study of GPU DVFS on energy conservation. https://www.researchgate.net/publication/262365062. Accessed 18 Mar 2020
29. Ge R, Vogt R, Majumder J, Alam A, Burtscher M, Zong Z (2013) Effects of dynamic voltage and frequency scaling on a K20 GPU. https://ieeexplore.ieee.org/document/6687422. Accessed 18 Mar 2020
30. Cameirinha DM (2015) Exploiting DVFS for GPU energy management. https://fenix.tecnico.ulisboa.pt/downloadFile/563345090414604/Dissertacao.pdf. Accessed 18 Mar 2020
31. Dutta B, Adhinarayanan V, Feng W (2018) GPU power prediction via ensemble machine learning for DVFS space exploration. https://www.researchgate.net/publication/326637320. Accessed 18 Mar 2020
32. Park J-G, Dutt N, Lim S-S (2017) ML-Gov: a machine learning enhanced integrated CPU-GPU DVFS governor for mobile gaming. https://www.researchgate.net/publication/320850321. Accessed 18 Mar 2020

33. Durillo JJ, Fahringer T (2014) From single- to multi-objective auto-tuning of programs: Advantages and implications. https://www.researchgate.net/publication/271724916. Accessed 18 Mar 2020
34. IntelliBreeze Software AB (2018) Awesome Miner.http://www.awesomeminer.com/home. Accessed 18 Mar 2020