Parallel Computing 94–95 (2020) 102621

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Analysis of energy efficiency of a parallel AES algorithm for CPU-GPU heterogeneous platforms

Xiongwei Fei[a,b,*], Kenli Li[a], Wangdong Yang[a], Keqin Li[a,c]

[a] College of Computer Science and Electronic Engineering, Hunan University, Changsha, China
[b] College of Information and Electronic Engineering, Hunan City University, Yiyang, China
[c] Department of Computer Science, State University of New York, NY, USA

ARTICLE INFO

Article history:
Received 5 July 2019
Revised 16 January 2020
Accepted 10 March 2020
Available online 20 March 2020

Keywords:
Advanced encryption standard
CPU parallelism
Energy efficiency
Frequency adjustment
GPU parallelism
Heterogeneous platform
Occupancy
Overlap
Workload balancing

ABSTRACT

Encryption plays an important role in protecting data, especially data transferred on the Internet. However, encryption is computationally expensive and this leads to high energy costs. Parallel encryption solutions using more CPU/GPU cores can achieve high performance. If we consider energy efficiency to be cost effective using parallel encryption solutions at the same time, this problem can be alleviated effectively. Because many CPU/GPU cores and encryption are pervasive currently, saving energy cost by parallel encrypting has become an unavoidable problem. In this paper, we propose an energy-efficient parallel Advance Encryption Standard (AES) algorithm for CPU-GPU heterogeneous platforms. These platforms, such as the Green 500 computers, are popular in both high performance and general computing. Parallelizing AES algorithm, using both GPUs and CPUs, balances the workload between CPUs and GPUs based on their computing capacities. This approach also uses the Nvidia Management Library (NVML) to adjust GPU frequencies, overlaps data transfers and computation, and fully utilizes GPU computing resources to reduce energy consumption as much as possible. Experiments conducted on a platform with one K20M GPU and two Xeon E5-2640 v2 CPUs show that this approach can reduce energy consumption by 74% compared to CPU-only parallel AES algorithm and 21% compared to GPU-only parallel AES algorithm on the same platform. Its energy efficiency is 4.66 MB/Joule on average higher than both CPU-only parallel AES algorithm (1.15 MB/Joule) and GPU-only parallel AES algorithm (3.65 MB/Joule). As an energy-efficient parallel AES algorithm solution, it can be used to encrypt data on heterogeneous platforms to save energy, especially for the computers with thousands of heterogeneous nodes.

1. Introduction

1.1. Motivation

For large-scale compute-intensive applications, users want faster computers. In the past, computer manufacturers improved performance by integrating more transistors and enhancing the frequency of cores, but this often led to higher energy consumption or higher operating temperatures. The former means spending more money, the latter may affect hardware functionality, even damage it. Thus, users need to seek highly energy-efficient computers with more cores but cores of relatively low-frequency. This can be seen by reviewing *The green 500 list-No. 2018* [1] that primarily evaluates computers based on their energy efficiency. The mainstream computers on the *Green 500* list are built on CPU + Nvidia GPU heterogeneous architecture.

CPU-GPU heterogeneous platforms are popular not only for supercomputers but also for general computers because of the high cost-to-performance ratio of GPUs. GPUs support general purpose computing and are easy to program using frameworks offered by their manufacturers. Nvidia provides the Computing Unified Device Architecture (CUDA) to facilitate programming for general purpose applications. The most important point is that GPUs show high energy efficiency due to simple control and high throughput. For this reason, we choose this type of platform to research and develop Energy-Efficient Parallel (EEP) AES algorithm as a popular and marketable approach.

1.2. Our contributions

In this work, we first analyze the opportunities for improving the energy efficiency of AES algorithm on CPU-GPU heterogeneous platforms. The energy consumed by GPUs is measured by

* Corresponding author at: College of Computer Science and Electronic Engineering, Hunan University, Changsha, China.
E-mail addresses: feixiongwei@hnu.edu.cn (X. Fei), lkl@hnu.edu.cn (K. Li), yangwangdong@hnu.edu.cn (W. Yang), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.parco.2020.102621
0167-8191/

Table 1
Some previous works on AES parallelizing.

Contributors	Language	Platform	Performance
Duta et al. [4]	CUDA, OpenCL, OpenMP	Unknown	CUDA: 25.51x Speedup; OpenCL: 18x Speedup; OpenMP: 11.29x Speedup
Pousa et al. [11]	OpenMP, MPI, CUDA	2 Intel Xeon E5405 (1 node); Cluster with 4 nodes; Nvidia Geforce GTX 560Ti	CUDA: 507.29x Speedup; MPI (32 cores): 31.67x Speedup; OpenMP (8 cores): 7.82x Speedup
Ortega et al. [12]	CUDA, OpenMP	Quadro FX 1800; Intel Core i7 960	CUDA: 18.31x Speedup; OpenMP: 3.05x Speedup
EEP	*CUDA + OpenMP*	*NVIDIA Tesla K20M; Intel Xeon E5-2640 v2*	*108.06x Speedup*

the NVML library [2], which is able to acquire the power and adjust the frequencies of Nvidia GPUs. The energy consumed by CPUs can be measured by the Intel Power Gadget Application Program Interfaces (APIs) [3], which can acquire the energy of Intel CPUs by energy Model-Specific Registers (MSRs). Then, we propose the corresponding methods to achieve an energy-efficient parallel AES algorithm. We evaluate the energy efficiency of EEP by comparing it with CPU-only parallel (CP) AES algorithm proposed in [4] and GPU-only parallel (GP) AES algorithm presented in [5]. Note that GP refers to the main encryption performed on GPUs in parallel, which still need the assistance of CPUs for such tasks as transferring data, executing key extensions, etc.

In summary, our contributions are listed as follows.

- We systematically analyze five opportunities for achieving energy efficiency of parallel AES algorithm.
- We propose an energy-efficient parallel AES algorithm called EEP, which synthetically adopts the methods of hybrid parallelizing, workload balancing, frequency adjusting, communication-computation overlapping, and best occupancy.
- We evaluate EEP in several aspects, including energy ratio, energy saving, energy efficiency, and so on, and compare EEP with CPU-only parallel AES algorithm, GPU-only parallel AES algorithm, and some previous works.

1.3. Organization

The remainder of this paper is organized as follows: Section 2 reviews some related works. Section 3, describes two opportunities to save energy. Section 4 provides the corresponding methodologies. Section 5 presents the algorithm for EEP that adopts the aforementioned methods. Section 6 describes the experiments conducted, then provides and evaluates their results. Section 7 provides conclusions and a look to the future.

2. Related works

2.1. Parallel AES algorithm

AES algorithm [6] is a symmetric block cipher and suitable for improving performance through parallelism when it works in ECB (Electronic Code Book) or CTR (Counter) mode. Some works that have enhanced the performance of AES algorithm relying on GPUs or CPUs are listed in Table 1. For comparison, this table also lists the performance of the EEP, which is discussed in Section 6.2.1 later.

Manavski first [7] used CUDA to parallelize AES algorithm on a GPU and achieved about 20 times speedup. Maistri et al. [8] also implemented parallel AES algorithm using CUDA and got good performance with excellent price/performance ratio. Iwai et al. [9] implemented parallel AES algorithm by use of CUDA on GPU and analyzed the effectiveness of AES algorithm implementation from the conditions of parallel processing granularity, memory allocation, etc. They found that the granularity at 16 Bytes/thread, and the shared memory of allocating T table and round keys are effective. Nishikawa et al. [10] got 48:6 Gbps throughput of parallel AES algorithm on Nvidia Telsa C2050. And then, Li et al. [5] further achieved around 60 Gbps throughput as the maximum performance on Nvidia Tesla C2050 GPU by using CUDA.

As a CPU has multiple cores, another parallelism is to use the CPU to parallelize AES algorithm. This attracts the attention of researchers as well. Some recent research is listed in Table 1 too. Duta et al. [4] parallelized AES algorithm using CUDA, OpenMP (Open Multi-Processing), and OpenCL (Open Computing Language) respectively and found that their performance descend by CUDA, OpenCL, and OpenMP. We choose the parallelized version of OpenMP to compare with our EEP and denote it as CP. POUSA et al. [11] implemented parallel AES algorithm by using OpenMP, MPI (Message Passing Interface), and CUDA respectively, and the last one also outputs the highest efficiency. Besides, Ortega et al. [12] parallelized AES algorithm using OpenMP and CUDA on multicore CPUs and many-core GPUs respectively, and observed that the latter outperforms the former. These results prove that parallelizing AES algorithm by CUDA is a strong and effective method.

Based on the above research for CPU-GPU heterogeneous computers, parallelizing AES algorithm by GPUs and CPUs simultaneously can be done with GPUs taking part of workload in CUDA manner and CPUs taking the remaining workload in OpenMP manner. This combination improves performance. Such heterogeneous platforms can also achieve energy efficiency due to the high energy efficiency of GPUs. We will analyze the reason in Section 3.2 and catch it as an opportunity to develop EEP.

2.2. Energy efficiency

As saving energy is as important as improving performance, it is hotly studied by many researches naturally. Some previous research efforts on energy efficiency are listed in Table 2 with the comparison of EEP.

Leng et al. [20] proposed a GPGPU power model and integrated it into GPGPU-sim, which is a cycle-level simulator modeling contemporary graphics processing units. It shows energy saving by utilizing dynamic voltage scaling, frequency scaling, and clock gating. Price et al. [21] investigated how temperature, frequency, and voltage impact on the energy efficiency of GPUs. They showed the effect of improving energy efficiency of GPUs for xGPU, a compute-bound application, by lowering voltage, increasing frequency, and maintaining a low die temperature.

In summary, scaling frequency is an important and basic technology in saving energy. Of the 16 works cited in Table 2, scaling frequency was used in works from 2 to 10 and from 12 to 14. For greater improvement in energy efficiency, scaling frequency should be combined with other technologies. This can also be seen in Table 2. Work 3 combined CPU frequency scaling. Work 4 combined voltage adjustment. Work 6 combined workload distribution. Work 8 combined clock gating. Work 9 combined lowering both voltage and temperature. Work 10 combined task migration. Work 12 combined dynamically scaling the number of cores. Work 13 combined the parallelism degrees of applications. Work 14 combined distribution of workload.

Table 2
Previous works on energy efficiency.

No.	Contributors	Methods	Platform	Energy Measure	Energy Efficiency
1	Hong and Kim [13]	Selecting the number of active cores	Nvidia GTX 280	Extech 380,801 power analyzer	Energy saving: 10.99%
2	Ge et al. [14]	DVFS	Dual Intel Sandy Bridge E5-2670, Nvidia K20c	RAPL, SMI	Energy efficiency: GPU 4.6x than CPU for Matrix Multiplication, 8.7x for TSP and FSM
3	Paul et al. [15]	Coordinated and dynamic energy management	AMD A10-5800	Power management firmware	Energy-delay squared product: up to 30%
4	Mei et al. [16]	Scaling down GPU core voltage and frequency	Intel Core i5-750, Nvidia GeForce GTX 560 Ti	Power meter	Energy saving: 19.28%, Performance decrease: <= 4%
5	Huang et al. [17]	GPU parallelism	Intel Core2 Duo, Nvidia Geforce GTX 280	Power meter	Energy efficiency of GEM: GPU version better than serial 763x and multi-threaded 237x
6	Abe et al. [18]	Scaling core and memory frequencies of GPU appropriate to the workload characteristics	Core i5 2400, Nvidia GeForce GTX 480	Power meter	Energy saving: 28%, Performance decrease: 1%
7	Jiao et al. [19]	Scaling frequencies by application intensity	Intel Core 2 Quad Q6600, Nvidia GTX 280	power meter	Energy efficient improvement: 4% for matrix multiplication, 8 ~ 9% for Matrix Transpose
8	Leng et al. [20]	DVFS and clock gating	Nvidia Geforce GTX 480, Nvidia Quadro FX 5600	GPGPU-Sim	Energy saving: 6.6% ~ 13.6%
9	Price et al. [21]	Lowering voltage, increasing frequency, and maintaining a low die temperature	Intel i7-2600, Nvidia K20	N/A	Energy efficiency improvement of xGPU: 37% ~ 48%
10	Sarood et al. [22]	DVFS and task migration	Cluster of 32 nodes (128 cores), each of which has a Intel Xeon X3430	Liebert power distribution unit	Cooling energy saving: up to 63%
11	Chiesi et al. [23]	Scheduling jobs by minimum power-slot policy on heterogeneous nodes	2 Intel Xeon E5520 and 2 Nvidia GeForce GTX 590 in a node, 4 nodes totally	Current sensing board, acquisition and elaboration PC	Peak-power reduce: 10% ~ 24%, Time increase: 2%
12	Lee et al. [24]	DVFS and dynamically scaling the number of cores	Modeled GPU by GPUGPU-Sim	Formula computing	Throughput improvement: up to 38%, nearly 20% on average
13	Lee et al. [25]	Trading off the frequency and the parallelism degrees of applications	Intel i7 720QM	N/A	Energy saving of JPEG: up to 60%
14	Ma et al. [26]	DVFS and distributing workload	AMD Phenom II X2, Nvidia GeForce 8800 GTX	Power meter	Energy saving: 21.04%
15	Wang and Ren [27]	Distributing workload	Intel Core i7, AMD 4870	Power meter	Energy saving: 14%
16	Arora et al. [28]	Power gating	AMD A8-5600K APU	Hardware	Energy saving: 8% (avg), up to 36%
17	*EEP*	*Hybrid parallelism, workload Balancing, frequency adjustment, and communication-computation overlapping*	*NVIDIA Tesla K20M; Intel Xeon E5-2640 v2*	*RAPL and NVML*	*Saving 74% energy of CPU-only parallel AES algorithm and 21% energy of GPU-only parallel AES algorithm*

Note: DVFS (Dynamic Voltage and Frequency Scaling), RAPL (Running Average Power Limit), SMI (Nvidia System Management Interface).

Works 1, 11, 15, and 16 also employed other technologies. Work 1 selected active cores. Work 11 scheduled jobs on heterogeneous nodes. Work 15 utilized workload distribution. Work 16 exploited power gating. The cited research examples illustrate that achieving an energy-efficient application needs to combine the properties of both the application and its execution platform. When popular CPU-GPU heterogeneous platforms and AES algorithm are combined to develop an EEP, the properties of both should be deeply and systematically studied. Section 3 provides analysis of the opportunities of achieving EEP on CPU-GPU heterogeneous platforms.

Works 17 is our proposal EEP, whose energy efficiency is listed in Table 2 also for comparison. Later, Section 6.2.5 will discuss the energy efficiency of EEP in details. We can find that EEP is energy more efficient than other works in Table 2.

3. Opportunities

3.1. AES algorithm parallelism opportunity

AES algorithm is a block cipher and each block can be encrypted or decrypted independently in ECB (Electronic Code Book) or CTR (Counter) mode. Thus, it can be accelerated in parallel. Its encryption process for a block (16 bytes data called *state*). The main encryption process for a block (16 bytes data called *state*) involves *Nr* rounds operations and a key extension. *Nr* is determined by AES algorithm types *AES-N: AES*-128, *AES*-192, and *AES*-256. They differ in key length of 128 bits, 192 bits, and 256 bits respectively. Their *Nrs* are 10, 12, and 14 respectively. Obviously, AES algorithm becomes more complicated and more secure as *Nr* increases.

Each round includes four procedures: *SubBytes, ShiftRows, MixColumns*, and *AddRoundKey*, except for the last round missing *MixColumns*. The $\frac{N}{8}$ bytes key is used by all the blocks and needs to be extended by *Extend* once to meet the requirement of *AddRoundKey*. The first *AddRoundKey* before the first round uses the first four words of the extended keys, and then each of the *Nr* rounds uses every four words of the left extended keys. The *Extend* only needs to be executed once in the whole process of encryption, so it has no large impact on performance. Whereas the four procedures need to be run multiple times for each *state*. In details, for a *state, SubBytes* and *ShiftRows* need to be executed *Nr* times; *MixColumns* and *AddRoundKey* need to be executed $Nr-1$ and $Nr+1$ times respectively.

In a round, the four procedures involve some operations as listed below (Note: We use the following abbreviations. X: XOR; L: lookup; R: rotation; M: multiplication.).

- *AddRoundKey* executes 16 XOR operations (16X) on a *state* and a round *key*. Its target is to blind data.
- *SubBytes* aims to nonlinearly substitute data using S-Boxes and can be accelerated by 16 *T* table lookups (16L).
- *ShiftRows* rotates rows by 0, 1, 2, 3 respectively to confuse data, thus it has 6 byte rotations (6R).
- *MixColumns* transforms and mixes data in columns which has $4\times4=16$ multiplications (16M) and $3\times4=12$ XORs (12X).

For example, when encrypting a *state, AES*-128 has the number of operations:

$$\begin{aligned} OPs &= Nr\times(OP_{sb}+OP_{sr})+(Nr+1)\times OP_{ak}+(Nr-1)\times OP_{mc}\\ &= 10\times(28X+16L+6R+16M)+16X-16M-12X\\ &= 284X+160L+60R+144M, \end{aligned}$$

where OP_{ak}, OP_{sb}, OP_{sr}, and OP_{mc} represent the number of operations of *AddRoundKey, SubBytes, ShiftRows*, and *MixColumns* respectively.

If not considering the differences of operations, there are 648 operations for encrypting 16 bytes data. In other words, *AES*-128 has the computing complexity of $O(M)=M^{2.33}$, where M represents the number of bytes of plain text, if not considering the differences of operations. *AES*-192 and *AES*-256 can calculate similarly and their computing complexities are $M^{2.4}$ and $M^{2.46}$ respectively. Therefore, AES algorithm is a computing-intensive problem. It is suitable to be executed on GPUs, because GPUs can support these integer operations through CUDA and efficiently hide the overhead of transferring data by large-scale parallelism.

3.2. AES energy-saving opportunities

For CPU-GPU heterogeneous platforms, power is consumed on both CPUs and GPUs with two parts: dynamic power and static power [21] [27]. The amount of dynamic power is determined by runtime system, whereas the amount of static power is decided by circuit technology, chip layout, and so on [21]. For example, dynamic power depends on the instructions of the kernel running on the GPU. Compute instructions cosume less energy than memory accesses on GPUs. A multiply-add instruction consumes 7–15 times less energy than accessing L1 memory on an Nvidia G80 [29]. Static power consumption is due to current leakage and comes from four main sources : reverse-biased junction leakage current, gate-induced drain leakage, gate direct-tunneling leakage, and subthreshold leakage [30]. Therefore, static power is constantly consumed regardless of transistor switching.

Power consumption *Power* can be calculated as:

$$Power = P_{cs}+P_{cd}+P_{gs}+P_{gd}, \tag{1}$$

where P_{cs}, P_{cd}, P_{gs}, and P_{gd} denote CPU static, CPU dynamic, GPU static, and GPU dynamic power, respectively. While energy consumption Energy$E$ is equal to:

$$\begin{aligned} E &= P\times T\\ &= (P_{cs}+P_{gs})\times T+(P_{cd}+P_{gd})\times T, \end{aligned} \tag{2}$$

where T represents the runtime of a specific application, such as AES. Because P_{gs} and P_{gd} do not change, reducing T, $T\times P_{cd}$, and/or $T\times P_{gd}$ provides opportunities to lower energy consumption.

In [31], P_gd can be approximately expressed as :

$$P\_gd = ACkf^{\alpha},\ \alpha=\frac{\gamma+1}{\gamma-1}, k=K^{\frac{2}{1-\gamma}}, \tag{3}$$

where A is the switching activity factor, C is the capacitance, f is the clock frequency, K and γ are the parameters related to manufacturing technique. γ is usually less than 2 and larger than 1, and P_gd has the exponential relationship with f. T is inversely proportional to f. This means that there is a suitable f to balance the P_gd and T and achieve good energy efficiency.

There are five main opportunities to save energy for parallel AES algorithm on CPU-GPU heterogeneous platforms in this work. They are hybrid parallelism, workload balancing, frequency adjustment, communication-computation overlapping, and full occupancy. Their rationales will be revealed and discussed in Sections 3.2.1–3.2.5.

3.2.1. Hybrid parallelism

Since GPUs have strong computing capacity, GPU parallel applications will spend less time than CPU parallel applications generally. If a hybrid parallel application distribute workload balance to CPUs and GPUs, it will spend less time compared with corresponding GPU parallel or CPU parallel algorithms.

Let us consider the application AES. Fig. 1 illustrates the power and time of CP (CPU Parallel), GP (GPU Parallel), and HP (Hybrid-Parallel) AES algorithms on CPU-GPU heterogeneous platforms. Because $E=P\times T$, the area of rectangles represents the energy consumed. Obviously, HP consumes less energy compared with GP and CP.

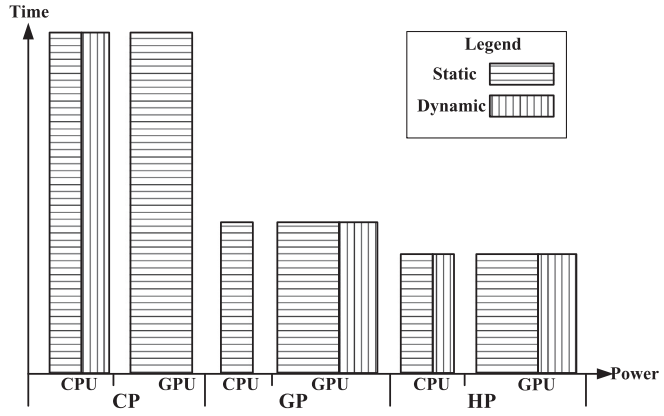

Fig. 1. Energy-saving model via hybrid parallelism. CP, GP, and HP represent CPU parallel AES, GPU parallel AES, and hybrid parallel AES respectively.

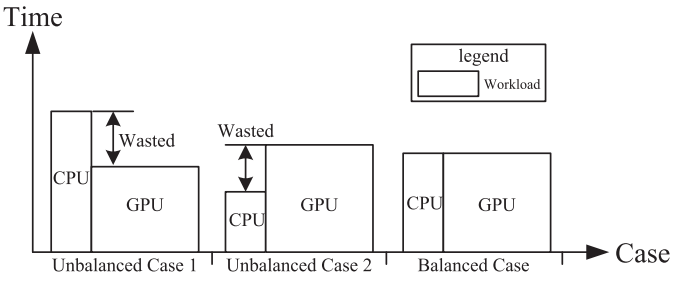

Fig. 2. Workload balancing impact.

CP only uses CPUs to encrypt data, but GPUs will consume static energy too. In other words, GPUs have no energy efficiency because they consume energy without product. Because of the relatively weak computing capacity of the CPU, CP will spend more time on encryption. This gives us an opportunity to reduce time by migrating the encryption task to GPUs in the GP manner. GP encrypts data on GPUs in parallel and will reduce the runtime due to high computing capacity of GPUs. However, GP will waste the energy cost of CPUs because CPUs do not perform any actual encryption. Obviously, using both GPUs and CPUs and letting them encrypt data in parallel, will reduce runtime and save energy.

In this hybrid algorithm, the GPU and in the CPU both spend static energy and dynamic energy. Although both the GPU and the CPU spend more energy than individual single parallel algorithm, the energy spent totally will be reduced with the cost time decreasing. Therefore, hybrid algorithm actually has higher energy efficient in other words.

Thus, adopting hybrid parallelism is an opportunity to achieve an energy-efficient parallel AES.

3.2.2. Workload balancing

The runtime can be reduced by using GPUs and CPUs harmonically. The workload should be balanced between GPUs and CPUs to avoid the unnecessary synchronization time.

Fig. 2 illustrates the impact of workload balancing. “Unbalanced Case 1” assigns too much workload to CPUs and leads to spending more time on CPUs. Similarly, “Unbalanced Case 2” assigns too much workload to GPUs and results in spending more time on GPUs. Because of greater computing capacity of GPUs, “Unbalanced Case 2” will spend less time than “Unbalanced Case 1”. Obviously in both of them, CPUs/GPUs have to wait GPUs/CPUs. Whereas “Balanced Case” avoids the wait by assigning the suitable workload to CPUs and GPUs.

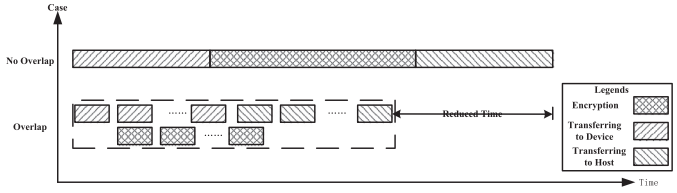

Fig. 3. Comparison of "overlapping" and "no overlapping".

3.2.3. Frequency adjustment

The NVML library is able to adjust the frequencies of GPU cores and memory, which will impact on both P_{gd} and T. To obtain the impact, we run GPU-parallel AES on a heterogeneous platform, where other experiments will be conducted as well. Table 3 depicts the configurations of the platform.

In Table 4, we have gathered the data of speed at encrypting plaintexts in different size. We normalized the data of speed at F5 to 1, and the other data of speed at different frequencies were compared with it and get the relative values. As plain text varies size from 100 MB to 800 MB, the relative values of performance of GP shows a proportional relationship with frequencies.

Comparing the last row with the other rows, the relative values of performance are very close to the relative values of the core frequencies. This enlightens us about two important facts. One is that GP is compute-bound because its performance is proportional to core frequencies. This consolidates our analysis in Section 3.1. The other is that improving core frequencies can reduce the runtime of encryption.

Undoubtedly, this is an opportunity to save energy for parallel AES. The method of how to adjust frequencies of GPU will be described in Section 4.3.

3.2.4. Communication-computation overlapping

Furthermore, there is another opportunity, i.e., decreasing the transferring overhead by overlapping communication and computation. The rationale is illustrated in Fig. 3, where “no overlapping” first transfers data to device, then encrypts the data, and then transfers the data back to host; while “overlapping” divides data into parts and transfers them to device in streams. A GPU can concurrently executes asynchronous commands in streams,. Different streams may execute their commands concurrently or out of order with respect to each other. The encryption can overlap with the data transfers, so the total runtime decreases.

For GPU parallel AES, the transfer time T_f is relatively less than the compute time T_c because of the compute-intensity of AES as discussed above. If employing *nStreams* streams, it will spend time of $\frac{T_f}{nStreams} + T_c$ less than the time of $T_f + T_c$ spent on the case without streams in theory. However, more streams do not definitely mean better performance due to streams’ extra overhead and resource contention. Through some experiments, the relatively better 16 streams are used in our experimental platform, i.e., $nStreams = 16$.

3.2.5. Full occupancy

The last method of saving energy is to reduce runtime by fully using GPU resources. The GPU thread resources can be sufficiently occupied at 100% percent by customizing thread organization. If the full occupancy is reached, a GPU has enough warps to be scheduled and executed, thus reduces possible waits for warps which lack resources. This will save runtime. In other words, this will save energy. How to customize thread organization for the full occupancy will be described in Section 4.5.

Table 3
Configurations of the experimental platform .

GPU	CPU
NVIDIA Tesla K20M (13 multiprocessors)	2 Intel Xeon E5-2640 v2 (support hyper-threading)
6 clock frequencies of core and memory in MHz: (324, 324), (614, 2600), (640, 2600), (666, 2600), (705, 2600), (758, 2600)	clock rate: 2.0 GHz
Total: 2496 cores	Total: 16 cores

4. Methodology

In this section, five methods corresponding to the opportunities in the Section 3.2 will be given. They are hybrid parallelizing, adjusting frequencies, balancing workload, overlapping communication-computation, and full occupying, which will be depicted from Subsections 4.1 to 4.5 orderly.

4.1. Hybrid parallelizing

Hybrid parallelizing AES can reduce the total runtime and improve energy efficiency because resources are not wasted. In order to achieve hybrid parallelism, a master-slave model is adopted. CPUs spawn multiple threads working in parallel, one of which, called service thread, is used to transfer data and call kernel functions and the others of which are used to execute some parts of a computation task.

4.2. Balancing workload

A key issue in reducing runtime in hybrid-parallel AES is how to maintain workload balance on CPUs and GPUs. Because AES is a block cipher and deals with data in groups of 16 bytes, the workload of each group is the same size. This provides a chance of assigning workload balanced based on the computing capabilities of CPUs and GPUs.

The assigning method involves two steps, as follows. First, adopt

$$\mathcal{R} = \frac{Gf \times N\_Mp \times Ws}{Cf \times (\alpha \times N\_C - 1) \times \beta} \tag{4}$$

to calculate the $\mathcal{R}$ of the computing capacity of GPUs over CPUs, and then assign $\frac{1}{\mathcal{R}+1}$ and $\frac{\mathcal{R}}{\mathcal{R}+1}$ of the workload to CPUs and GPUs respectively.

In the right-hand side of Eq. (4), the numerator and the denominator represent the computing capacities of GPUs and CPUs respectively.

In the numerator, *Gf*, *N_Mp*, and *Ws* denote the frequency of the cores, the number of multiprocessors, and the size of warp on GPUs respectively. The above parameters can be retrieved by the function of *cudaGetDeviceProperties*. Because a multiprocessor schedules and issues threads in units of warp, $N\_Mp \times Ws$ means the number of threads running on GPUs simultaneously and $Gf \times N\_Mp \times Ws$ means all the work ability of these running threads.

In the denominator, *Cf* and *N_C* denote the frequency of the cores and the number of cores on CPUs respectively, which can be retrieved by the library of powerGadget. The coefficients α and β are used to differentiate hyper-threading. If CPUs support hyper-threading, they are 2 and 0.7 respectively; otherwise they both are 1.

If CPUs support hyper-threading, adopting hyper-threading can yield more threads and improve the performance of the CPUs.This means CPUs can be assigned more work. Hyper-threading can improve performance by 40–50% according to Intel's specifications and yield as twice as many threads as nonhyper-threading, i.e., $2 \times N\_C$ threads. However, one of them will be used to transfer data and call the kernel. Therefore, only $2 \times N\_C$ threads will actually undertake computation. But two hyper-threads have 1.4–1.5 times the computing ability of one normal thread. We employed 1.4 times to calculate conservatively. Thus, $Cf \times (2 \times N\_C - 1) \times 0.7$ represents the computing capacity of hyper-threads except the service thread.

If CPUs do not support hyper-threading, they will derive *N_C* threads and one of them will serve GPU parallelism, thus $Cf \times (1 \times N\_C - 1) \times 1$ expresses the computing capacity of CPUs.

In this paper, we considered only static allocation, but dynamic allocation will be considered to suit the change of background processes in our future work.

4.3. Adjusting frequencies

Because adjusting frequencies will impact energy consumption, it can be used to save energy for parallel AES. It can be accomplished using the NVML library. In the library, the *nvmlDeviceSetApplicationsClocks* can be used to set the clocks at which applications will run. The clocks include the memory clock and the core clock and must be specified as the clocks supported by the device. The *nvmlDeviceGetSupportedGraphicsClocks* and the *nvmlDeviceGetSupportedMemoryClocks* can query all supported core and memory clocks respectively.

4.4. Overlapping communication and computation

Communication-computation overlapping can save time and hide overhead efficiently. The methodology to achieve this is to adopt streams and asynchronized transfers. First, allocate host memory for the data that will be transferred to device memory. Next, copy data to host memory. Then, create streams. Each stream executes its own data transfers and kernel calls concurrently. The below pseudo code in Algorithm 2 illustrates the process of overlapping communication and computation.

4.5. Full occupancy

For any given application, the number of activated warps in a multiprocessor is limited by the resource usage of the application, including registers and shared memory, and by the physical limits of GPUs. Achieving the full occupancy requires the following steps:

1. Obtain the GPU properties by utilizing *cudaGetDeviceProperties*;
2. Get the resource usage of the application using different block sizes in compiling the application with option $--ptxas-options=-v$;
3. Calculate the occupancies of different block size using the above information, the detailed information can be found in [32];
4. Choose the block size with the full occupancy to organize threads.

Based on calculation by this method, all the GPU occupancy ratios of parallel AES with block sizes of multiple of warp size are listed in Fig. 4. Block sizes of 256, 512, and 1024 with 100% occupancy can be chosen. The block size of 512 was employed in this work.

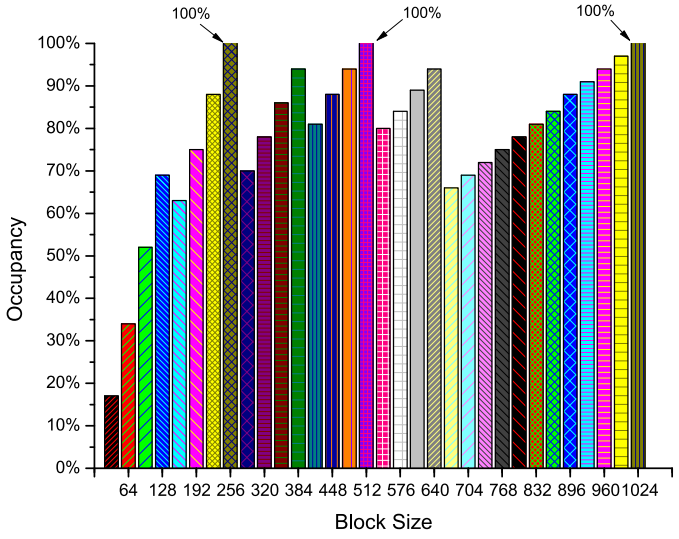

Fig. 4. GPU Occupancy Ratio.The GPU occupancy ratios with all block sizes of the multiple of warp size (32) are portrayed. The full occupancy with ratio of 100% exists in the block sizes of 256, 512, and 1024.

5. Algorithms

Based on the previous two sections, we propose an Energy-Efficient Parallel (EEP) AES Algorithm that employs hybrid parallelizing, balancing workload, adjusting frequencies, overlapping communication and computation, and full occupancy. EEP can be seen in Algorithm 1. The details of the kernel in Line 14 of Algorithm 1 are shown in Algorithm 3.

Algorithm 1 Energy-Efficient Parallel AES Algorithm.

Require:
The data to be protected with Length of *Len*, *plaintext*;
The key of AES, *key*;
Ensure:
1: Set optimal GPU core frequency and memory frequency by NVML;
2: Get the parameters used in Eq. (4);
3: Get *Len*, i.e., the length of *plaintext*;
4: Calculate *ratio* by Eq.(4);
5: Extend *key* to *exKey*;
6: Derive $\alpha \times Ccores$ threads on CPU;
7: Assign a portion with $LenC = Len \times \frac{1}{\mathcal{R}+1}$ of *plaintext* to threads $1 \sim \alpha \times N\_C - 1$ evenly;
8: Assign the left portion of *plaintext*, i.e. $LenG = Len - LenC$, to thread 0;
9: Encrypt $LenT = \frac{LenC}{\alpha \times N\_C - 1}$ of *plaintext* by each of threads $1 \sim \alpha \times N\_C - 1$ in parallel;
10: Allocate *LenG* space on device memory by thread 0;
11: Create *nStreams* streams by thread 0;
12: **for** each stream *Si* in *S* **do**
13: Transfer $LenS = \frac{LenG}{S}$ data to device memory asynchronously;
14: Call $encKernel <<< gDim, bDim, 0, Si >>> (plaintext + LenS * Si, LenS, exKey, T)$ which will encrypt *LenS* data from $plaintext + LenS * Si$ with *key* on GPU and is described in Alg. 3;
15: Transfer $LenS = \frac{lenG}{S}$ data to main memory asynchronously;
16: **end for**
17: Receive the encrypted data from GPU by thread 0;
18: Finish the encryption by threads $1 \sim \alpha \times N\_C - 1$;
19: Return *plaintext* which has been encrypted.

Algorithm 2 Overlapping communication-computation.

Require:
The data to be dealt, *data*;
The number of streams, *nStreams*;
Ensure:
1: Allocate host memory for *data* by *cudaMallocHost*();
2: Copy *data* to host memory by *cudaMemcpy*();
3: Create *nStreams* streams *streams* by *cudaStreamCreate*();
4: **for** each *stream* in *streams* **do**
5: Use *cudaMemcpyAsync*() to asynchrously tranfer the data used by the *stream*: *data*[*stream*];
6: Call kernel by $kernel <<< dg, gb, sm, stream >>> (data[stream])$;
7: Use *cudaMemcpyAsync*() to asynchrously transfer the results produced by the *stream* back to host;
8: **end for**
9: Destroy *streams* by *cudaStreamDestroy*().

Algorithm 3 Encrypt Kernel.

Require:
The data to be encrypted, *data*;
The extension key of AES, *exKey*;
The temporary state data, *state*;
Ensure:
1: Allocate shared memory for *T* table and *exKey*;
2: Load *T* tables and *exKey* to the shared memory;
3: Synchronize threads;
4: **if** thread id $tid < \frac{Len(data)}{16}$ **then**
5: $state \leftarrow data[tid \times 16]...data[(tid+1) \times 16 - 1]$;
6: AddRoundKey(*state*, *exKey*[0]);
7: **for** each round $i = 1$ to the number of rounds *Nr* **do**
8: SubBytes(*state*, *T*);
9: ShiftRows(*state*, *T*);
10: **if** $i \neq Nr$ **then**
11: mixColumns(*state*, *T*);
12: **end if**
13: AddRoundKey(*state*, *exKey*[i]);
14: **end for**
15: **end if**

EEP requires the data to be protected *plaintext* and produces the corresponding cipher text placed at the same location. It employs hybrid-parallel AES on CPUs and GPUs, thus needs to distribute *plaintext* balanced to them.

The distribution needs the parameters of CPUs and GPUs, so the algorithm first sets the optimal GPU core and memory frequencies in Line 1. The details are using *nvmlDeviceGetSupportedMemoryClocks* and *nvmlDeviceGetSupportedGraphicsClocks* to retrieve the supported frequencies of the GPU, and then using *nvmlDeviceSetApplicationsClocks* to set the highest frequencies. Then get these parameters in Line 2. Line 3 gets the length of the *plaintext* required by calculating the distribution. Next, Line 4 calculates the ratio of computing capacity by Eq. (4).

After extending *key* to *exKeys* in Line 5 by a single thread, $\alpha \times N\_C$ threads are derived in Line 6, because the extension of key needs to run once serially. Subsequently, the *plaintext* can be assigned to CPUs and GPUs in Lines 7 and 8 respectively. In details, for CPUs, $\alpha \times N\_C - 1$ threads will evenly undertake the encryption of the portion of $\frac{1}{\mathcal{R}+1}$ of the *plaintext* and the service thread (thread 0) will send the left portion to GPUs for encryption.

In Line 9, $\alpha \times N\_C - 1$ threads encrypt the data in parallel. In Lines 10–17, thread 0 executes its service at the same time. In details, thread 0 first allocates memory for the *plaintext* in Line 10, and then creates *nStreams* streams in Line 11. All the streams will asynchronously and evenly transfer the *plaintext* to device and back host in Lines 13 and 15 respectively. Of course, the real encryption work is done by these streams of GPU in parallel as shown in Line 14. Every stream is in charge of encrypting data

evenly and organizes the threads in the full occupancy. Once all the threads on CPU have gotten the encrypted *plaintext* in Lines 17 and 18, the *plaintext* can be returned and all the encryption work has been done.

Algorithm 3 depicts the details of the kernel in Line 14 of Algorithm 1. The kernel is called by thread 0 on CPUs and executed on GPUs in many threads manner. The threads are organized in the full occupancy. They encrypt the data transferred to GPUs and produce the corresponding ciphertext. In Line 1, the algorithm first allocates shared memory for *T* table (4KB) and *exKey* (176 Bytes), because they will be reused multiple times and can be placed in shared memory. After this, it loads *T* table and *exKey* to the shared memory in Line 2 and synchronizes the threads to ensure that the loads finished. If a thread has data to be encrypted that is decided by the condition in Line 4, this thread will perform AES encryption as shown from Lines 5–14.

6. Experiments

6.1. Experimental setup

Experiments are conducted on a hybrid CPU-GPU platform with the configurations in Table 3. We employ three different algorithms, i.e., CP (CPU-only Parallel), GP(GPU-only Parallel), and EEP, to encrypt plaintexts with length of 100, 200, 400, 600, 800, and 1000 MB. Each plain text is encrypted 20 times to get the average value. In the process of encryption, on one side we sample the power of the CPUs and the GPU periodically and write them to log files; on the other side we time the process. After the completion of encryption, we calculate the average power of the CPUs and the GPU and then multiply the spent time *T* to get the energy consumption *E* as Eq. (2).

For EEP, we use NVML to get the power by *nvmlDeviceGetPowerUsage* every 50*ns* and *Event* to record the spent time, thereby calculate the cost energy of the GPU by Eq. (2). In CUDA, event API (Application program interface) can be used to accurately time events by starting events at different times, and comparing their start times. The *nvmlDeviceGetPowerUsage* is called by a single thread on the CPUs during the encryption.

We use *PowerGadget* provided by Intel to get the data of power of the CPUs. *PowerGadget* uses RAPL (Running Average Power Limit) library to get power data periodically. A single thread on the CPUs is used to gather the power data by calling *get_pkg_total_energy_consumed* in the experiments.

According to Algorithm 1, EEP needs to derive $\alpha \times N\_C = 2 \times 16 = 32$ threads on the CPUs, but two of which will be used to acquire the power of the GPU and the CPUs respectively in our experiments. Additionally, one of these threads will act as the service thread. Therefore, only 29 threads will encrypt the plain text on the CPUs. To adapt this change, Algorithm 1 will change the number of working CPU threads to 29 also, and then calculate the *ratio* by this new value.

For CP, it also derives $\alpha \times N\_C = 2 \times 16 = 32$ threads on the CPUs and two of them will be used to get the power of the CPUs and the GPU respectively. That is to say only 30 threads will encrypt plain text on the CPUs.

For GP, it will derive 3 threads on the CPUs, two of which will be used to get the power of the CPUs and the GPU respectively and one of which will be in charge of serving the GPU. The GPU will be in charge of all the workload of encryption.

6.2. Experimental results and discussion

In this subsection, we will report and discuss the results of the experiments from five aspects. In details, performance comparison, energy comparison, energy ratio, effect of adjusting frequency, and energy efficiency will be reported and discussed in Sections 6.2.1–6.2.5 respectively.

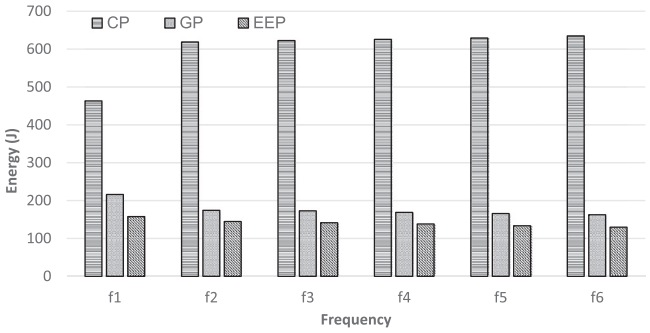

Fig. 5. Energy Comparison. The energy includes all the energy of the CPUs and the GPU cost by CP, GP, and EEP in encrypting plaintext at different frequencies. The different frequencies of the GPU, i.e., *Fi*, $i = 1 \ldots 6$, correspond to six frequencies of Nvidia K20M listed in Table 4 respectively.

Table 4
Relative performance of GPU-parallel AES.

Size(MB)	F1	F2	F3	F4	F5	F6
100	0.47	0.89	0.93	0.96	1.00	1.06
200	0.47	0.89	0.87	0.96	1.00	1.07
400	0.42	0.92	0.90	0.99	1.00	1.11
800	0.47	0.89	0.90	0.93	1.00	1.06
RV of core freq.	*0.46*	*0.87*	*0.91*	*0.94*	*1.00*	*1.08*

Note: F1 (324&324 MHz), F2 (614&2600 MHz), F3 (640&2600 MHz), F4 (666&2600 MHz), F5 (705&2600 MHz), F6 (758&2600 MHz), RV (Relative Value)

Table 5
Performance comparison.

Plaintext Size (MB)	EEP Time (ms)	CS Time (ms)	Speedup
100	170.7973	18095.2602	105.9459
200	335.5173	36231.9162	107.9882
400	666.97885	72456.1788	108.6334
600	1025.2082	108590.0984	105.9200
800	1320.2014	144789.5642	109.6723
1000	1642.4845	181009.2286	110.2045

6.2.1. Performance comparison

Table 5 lists the time of EEP and CS (CPU Serial AES algorithm) is spent in encrypting plaintext of different sizes. This table also lists the speedup calculated by $Speedup = EEP\_Time/CS\_Time$, where *CS_Time* means the encryption time of EEP and CS. The average *Speedup* is 108.06. In Section 2.1, Table 5 lists the performance some previous work, we can see that EEP has relatively high performance.

6.2.2. Energy comparison

Fig. 5 compares the total energy cost in encryption of six different amounts of plain text at different GPU frequencies. In this figure, the energy costs are increased with increased amounts of plain text, because larger amounts of plain text will lead to more operations and require more time. EEP costs less energy than GP, while GP costs less energy than CP. That is to say, EEP sufficiently utilizes the energy efficiency of both GPU and CPUs, however GP utilizes only the energy efficiency of GPUs and CP utilizes only the energy efficiency of the CPUs.

The figure also demonstrates the impact of different GPU frequencies. CP costs more energy at higher GPU frequencies, because the GPU is not involved in the encryption production, but costs more energy at higher frequencies. GP costs less energy than CP because the GPU has greater computing capacity and provides

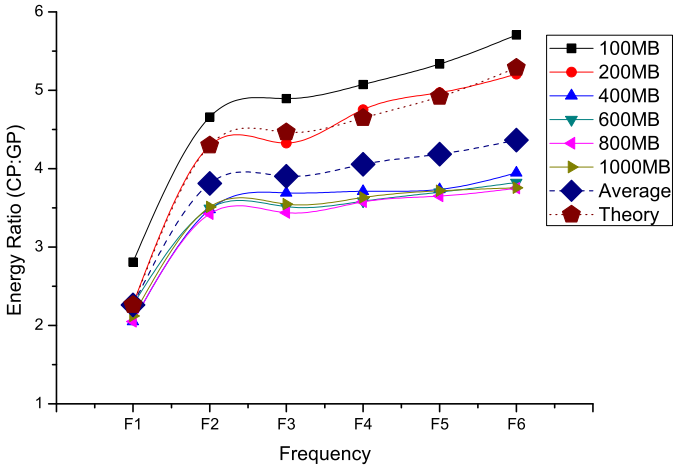

Fig. 6. Energy ratio of CP to GP. The ratio is calculated by $\frac{E_{CP}}{E_{GP}}$, where E_{CP} and E_{GP} represent the energy cost by CP and GP respectively in encrypting plain text.

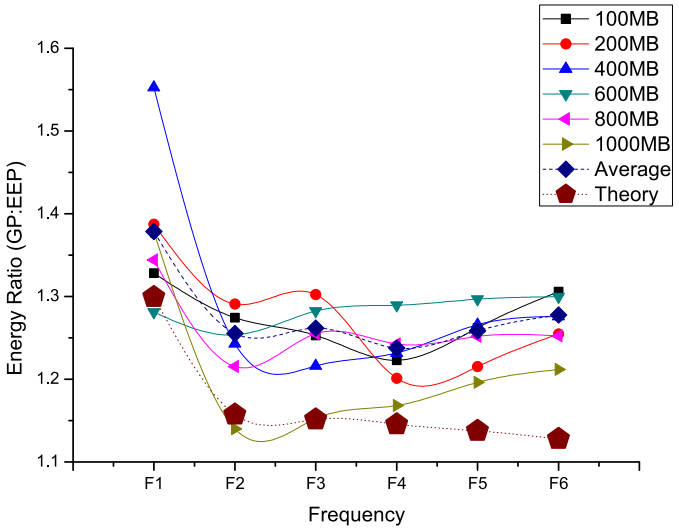

Fig. 7. Energy ratio of GP to EEP. The ratio is calculated by $\frac{E_{GP}}{E_{EEP}}$, where E_{GP} and E_{EEP} represent the energy cost by GP and EEP respectively in encrypting plaintext.

higher energy efficiency. GP reduces the energy cost at higher GPU frequencies, because it decreases the runtime. EEP also costs less energy at higher GPU frequencies for the same reason. In addition, CPUs contribute energy efficiency in the process of encryption as well, so EEP costs less energy than GP.

6.2.3. Energy ratio

To better show the effect of saving energy by EEP, Fig. 6 shows the energy ratios of CP to GP, while Fig. 7 exhibits the energy ratios of GP to EEP.

In Fig. 6, The average energy ratio of CP to GP $AR = \frac{A_{EC}}{A_{EG}}$, where A_{EC} and A_{EG} represent the average energy cost by CP and GP respectively. The theoretical energy ratio $TR = \frac{(TR_{base} \times Fi_{core})}{F1_{core}}$, $i = 1 \ldots 6$, where Fi_{core} represents the *ith* core frequency of GPU and TR_{base} is set to the average energy ratio at F1. In the figure, regardless of the amount of plain text, the ratios of CP to GP become greater with the increase of frequencies. The ratio at frequency *F*1 has a relatively lower value, 2.26 on average, while these ratios at frequencies *F*2 to *F*6 have higher values, such as 3.81, 3.90, 4.06, 4.18, and 4.36 respectively on average. The reason is that GP costs less energy at higher GPU frequencies; whileas, CP is the opposite. For plain text of 400 MB to 1000 MB, the energy ratios are very similar to each other and sometimes cross. This is because the GPU fully uses resources and consumes the maximal energy for large amounts of plain text. Note the cross-over lines are caused by measurement deviation.

To explain these ratios and verify them further, we analyze and discuss them as follows: For GP, the theoretical ratios and the average ratios are shown in Fig. 6. From this figure, we find that the theoretical energy ratios of CP to GP are close to and little greater than the average energy ratios. This is because the theoretical energy ratios do not consider the static energy consumption that always holds a fixed portion of the consumed energy. As a meaningful conclusion, we roughly get the linear relationship between the energy ratio and the core frequency of the GPU.

In Fig. 7, the average energy ratio of GP to EEP $AR = \frac{A_{EG}}{A_{EE}}$, where A_{EG} and A_{EE} represent the average energy cost by GP and EEP respectively in experiments. The theoretical energy ratio of GP to EEP is $TR = 1 + \frac{1}{\mathcal{R}}$, where the $\mathcal{R}$ is calculated by Eq. (4). This is because $TR = \frac{T_{EG}}{T_{EE}}$, where T_{EG} and T_{EE} represent the energy cost by GP and EEP in theory respectively. Omitting the impact of the workload distributed to CPUs [18], $TR \approx \frac{\mathcal{R}+1}{\mathcal{R}} = 1 + \frac{1}{\mathcal{R}}$, because in EEP the part of $\frac{\mathcal{R}}{\mathcal{R}+1}$ of workload is distributed to the GPU, while in GP the whole workload is distributed. In the figure, we find that the energy ratios of GP to EEP have lower values than those of CP to GP. This is because GPUs have greater computing capacity than CPUs and CP will take longer than GP to finish encryption, thus consuming much more energy. Whereas EEP does not have a large computing difference with GP. Currently, GPUs have much more cores and consume more energy than CPU with relative less cores. Because the GPU consumes more energy, these ratios are close. Similarly, the cross-over lines are caused by measurement deviation.

Another finding is that the energy ratios decrease with the increase of core frequencies of the GPU. Fig. 7 shows the average energy ratios in the experiments and the energy ratios in theory. As described in the caption, because $\mathcal{R}$ is determined by and proportional to the current core frequency of the GPU as shown in Eq. (4), the theoretical energy ratios, i.e., $1 + \frac{1}{\mathcal{R}}$, decrease with the increase of core frequencies of the GPU.

The last finding is that the theoretical ratios are close to but less than the average energy ratios. This is because the impact of streams is omitted. EEP employs 16 streams and reduces the total time roughly to $\frac{T_f}{16} + T_c$, where T_f and T_c denote the transfer time and compute time respectively. Whereas in the calculation, it is just considered as $T_f + T_c$ for the convenience of comparison. The time spent by EEP in encryption is overestimated, thus the energy is overestimated also. So the theoretical energy ratios of GP to EEP calculated by $1 + \frac{1}{\mathcal{R}}$ are less than those calculated by $\frac{T_{EG}}{T_{EE}}$.

6.2.4. Effect of adjusting frequency

To further study the effect of adjusting frequency, we focus on the energy cost by EEP at different frequencies. In Fig. 8, energy has the same trend for all the different amounts of plain text, i.e., the energy decreases with the increase of core frequencies of the GPU. This is because higher core frequencies help reduce the total runtime and the static energy consumption of the GPU and the CPUs. Of course, for larger amounts of plain text, the total runtime will increase and the energy will increase correspondingly.

As discussed in Section 6.2.2, it is beneficial to set the frequency of GPUs to the highest for GP and EEP, while to the lowest for CP. Fig. 9 demonstrates the energy cost by CP, GP and EEP at the frequency of *F*1, *F*6, and *F*6 respectively and their energy-saving percentages. The energy includes the total energy cost by CP at F1, GP at F6, and EEP at F6. *GC*, *EC*, and *EG* represent the energy-saving percentage of GP over CP, EEP over CP, and EEP over GP, respectively. $GC = (1 - \frac{Egp}{Ecp}) \times 100\%$, $EC = (1 - \frac{Eeep}{Ecp}) \times 100\%$, and $EG = (1 - \frac{Eeep}{Egp}) \times 100\%$, where *Egp*, *Ecp* and *Eeep* represent the energy cost by GP, CP, and EEP, respectively. Obviously, EEP

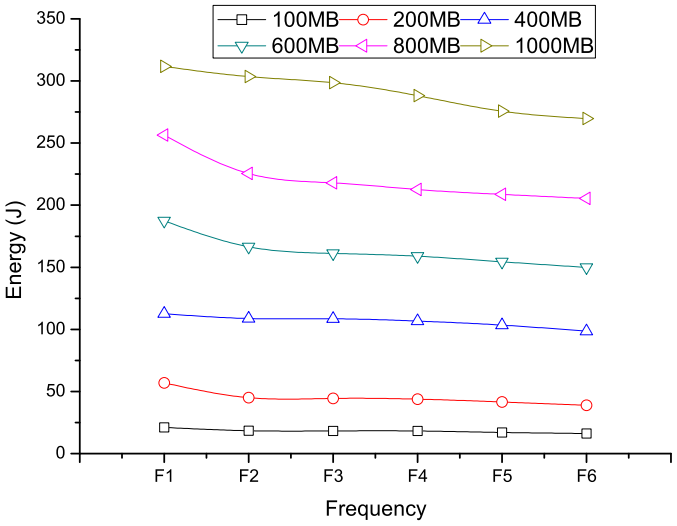

Fig. 8. Energy cost by EEP in encrypting different amounts of plain text at different frequencies.

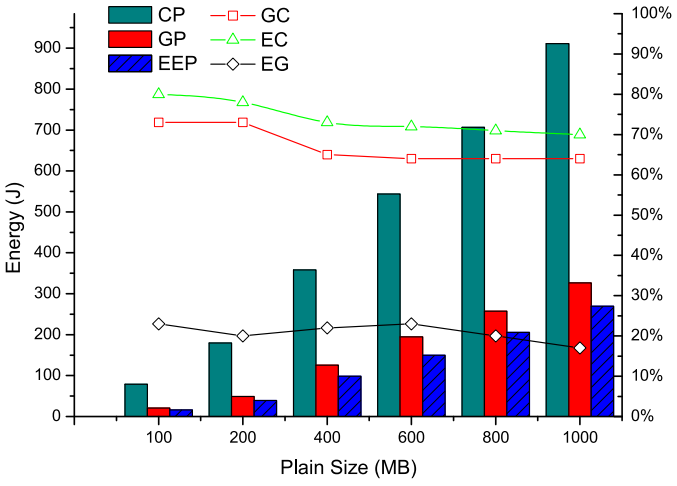

Fig. 9. Energy and energy-saving percentage.

costs the least energy compared with GP and CP, while GP costs less energy than CP. The energy-saving percentages of GP over CP, EEP over CP, and EEP over GP are 67%, 74%, and 21% on average, respectively. This is because both EEP and GP utilize GPU parallelism and occupy the resources of GPU as much as possible.

6.2.5. Energy efficiency

Evaluating energy efficiency *EE* usually adopts a combined metric of performance *Pf* and power *Power*, i.e., *EE* can be calculated by $EE = \frac{Pf}{Power}$.

For AES, the performance *Pf* can be calculated by $Pf = \frac{PS}{Time}$, where *PS* and *T* represent the amount of plain text and the time spent in encryption, respectively.

The *Power* of EEP, GP, and CP can be gotten by sampling with NVML for GPUs and Power Gadget for CPUs. Since the Energy *E* can be calculated by $E = P \times T$, $EE = \frac{PS}{T \times P} = \frac{PS}{E}$. For AES, *EE* has the unit of MB/J. Fig. 10 shows *EE* of CP, GP, and EEP at *F*6 (758 MHz). From this figure, there are three observations:

1. For CP, *EE* is the lowest among GP and EEP and remains stable, because it utilizes only CPU energy efficiency and its overhead is almost the same for different amounts of plain text.

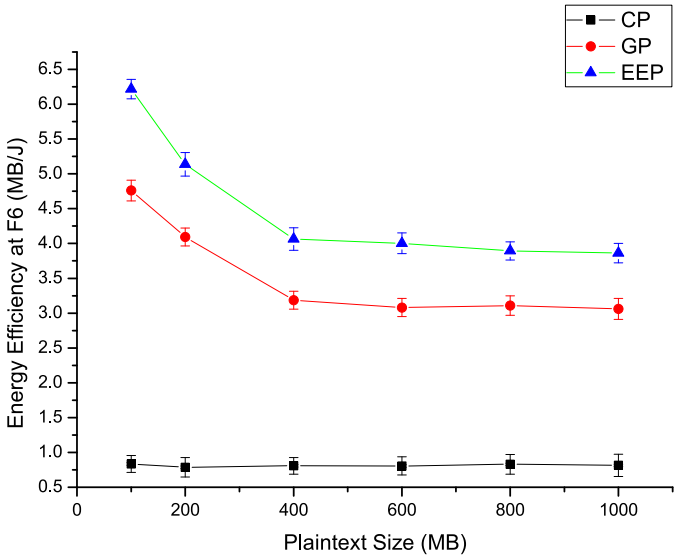

Fig. 10. Energy efficiency at F6 frequency.

2. For GP and EEP, the GPU has higher energy efficiency at low amounts of plain text and has stable energy efficiency at moderate amounts of plain text. This is because the GPU does not fully use all the resources to encrypt low amounts of plain text and then spends less energy, while once the plain text is large enough, the GPU will fully utilize the resources, the energy consumption will remain stable, and the energy efficiency also remains steady. In our experiments, the energy efficiency of GP decreases as the amounts of plain text increase from 100 MB to 400 MB, but remains stable from 400 MB to 1000 MB. This illustrates that 100 MB and 200 MB of plain text are low, but 400 MB to 1000 MB of plain text are moderate.
3. EEP has a higher energy efficiency than GP for every amount of plain text in our experiments. This is because GP wastes the computing resources of CPUs but still consume energy while the GPU is working. Naturally, EEP has higher energy efficiency than GP.

Because F6 is the highest frequency supported by K20M, both EEP and GP have their highest energy-efficiencies correspondingly. For the other frequencies, both EEP and GP have relatively low energy efficiencies, while CP has relatively high energy efficiencies due to wasting less energy on the GPU at low frequencies.

To prove the above paragraph, the average energy efficiencies for CP, GP, and EEP are calculated to exhibit their trends. Fig. 11 shows their average energy efficiencies at frequencies F1–F5.

First, the trend in Fig. 11 is consistent with that in Fig. 10. This states that different frequencies of the GPU can affect the energy efficiencies to a certain degree.

Second, the average energy efficiency of CP is relatively higher than that in Fig. 10. Specifically, the energy efficiency of CP on average is 0.81 MB/J at F6 and 0.88 MB/J at the other frequencies on average. CP has the highest energy efficiency of 1.15 MB/J on average at F1 compared with the other frequencies.

Third, for GP and EEP, their average energy efficiencies are relatively lower than that in Fig. 10. Specifically, the energy efficiencies of GP and EEP on average are 3.60 MB/J and 4.66 MB/J at F6 respectively and are 3.14 MB/J and 4.01 MB/J at the other frequencies, respectively. GP and EEP have their highest energy efficiencies on average at F6 compared with the other frequencies.

Fourth, in Section 2.2 and Table 2, we systematically analyzes previous works on energy efficiency and compare these previous

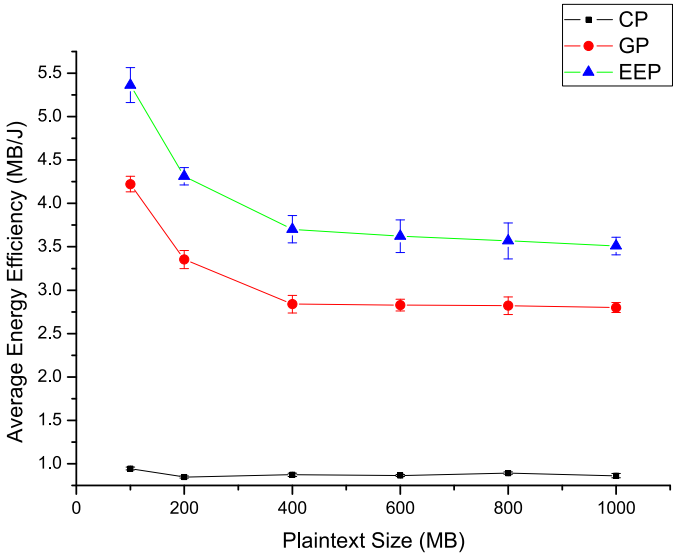

Fig. 11. Energy efficiency on average at GPU frequencies *F*1 to *F*5.

works with EEP on energy efficiency. We can find that EEP in these works has better energy efficiency also.

6.2.6. Summary

We have verified the methods in the experiments and have shown the effects of EEP. All in all, EEP employs hybrid parallelizing, workload balancing based on computing capacities, frequencies adjusting, communication-computation overlapping, full occupancy to achieve energy saving of 74% over CP (F1) and 21% over GP. From the aspect of energy efficiency, EEP has the highest energy efficiency of 4.66 MB/J higher than both 3.65 MB/J of GP and 1.15 MB/J of CP (F1). In Table 2, EEP also shows better energy efficiency than other works.

7. Conclusion

In this paper, we propose an Energy-Efficient Parallel (EEP) AES algorithm for CPU-GPU heterogeneous platforms. We aim to save energy and improve the energy efficiency of parallel AES algorithm on this type of platform based on the properties of AES algorithm and its execution platforms.

This work explores the energy efficiency of parallel AES algorithm for CPU-GPU heterogeneous platforms. In future, we plan to study how to save energy in other applications such as Leukocyte, HotSpot, and so on for CPU-GPU platforms.

Declaration of Competing Interest

We declare that we have no financial and personal relationships with other people or organizations that can inappropriately influence our work, there is no professional or other personal interest of any nature or kind in any product, service and/or company that could be construed as influencing the position presented in, or the review of, the manuscript entitled, An Energy-Efficient Parallel AES Algorithm for CPU-GPU Heterogeneous Platforms.

Acknowledgments

The authors would like to thank the anonymous reviewers for their insightful comments and valuable suggestions, and the support of the Key Program of National Natural Science Foundation of China (Grant nos. 61133005, 61432005), Natural Science Foundation of Hunan Province (Grant no. 2018JJ2025), and Scientific Research Fund of Hunan Provincial Education Department (Grant no. 18B442).

References

[1] green500.org, The green500 list - november 2018, (https://www.top500.org/green500/lists/2018/11/).
[2] N. Corp, Nvml api reference manual (2017).
[3] S.-W. Kim, J.J.-S. Lee, V. Dugar, J. De Vega, Intel® Power Gadget, 7, 2014.
[4] C.-L. Duta, G. Michiu, S. Stoica, L. Gheorghe, Accelerating encryption algorithms using parallelism, in: Control Systems and Computer Science (CSCS), 2013 19th International Conference on, IEEE, 2013, pp. 549–554.
[5] Q. Li, C. Zhong, K. Zhao, X. Mei, X. Chu, Implementation and analysis of AES encryption on GPU, in: High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on, IEEE, 2012, pp. 843–848.
[6] J. Daemen, V. Rijmen, Aes proposal: Rijndael (1999).
[7] S.A. Manavski, CUDA compatible GPU as an efficient hardware accelerator for AES cryptography, in: Signal Processing and Communications, 2007. ICSPC 2007. IEEE International Conference on, IEEE, 2007, pp. 65–68.
[8] P. Maistri, F. Masson, R. Leveugle, Implementation of the advanced encryption standard on GPUs with the Nvidia Cuda framework, in: Industrial Electronics and Applications (ISIEA), 2011 IEEE Symposium on, IEEE, 2011, pp. 213–217.
[9] K. Iwai, N. Nishikawa, T. Kurokawa, Acceleration of AES encryption on CUDA GPU, Int. J. Netw. Comput. 2 (1) (2012) pp–131.
[10] N. Nishikawa, K. Iwai, T. Kurokawa, High-performance symmetric block ciphers on Cuda, in: Networking and Computing (ICNC), 2011 Second International Conference on, IEEE, 2011, pp. 221–227.
[11] A. Pousa, V. Sanz, A. de Giusti, Performance analysis of a symmetric cryptographic algorithm on multicore architectures, Computer Science & Technology Series-XVII Argentine Congress of Computer Science-Selected Papers. Edulp, 2012.
[12] J. Ortega, H. Trefftz, C. Trefftz, Parallelizing AES on multicores and GPUs, in: Proceedings of the IEEE International Conference on Electro/Information Technology (EIT), 2011, pp. 15–17.
[13] S. Hong, H. Kim, An integrated GPU power and performance model, in: ACM SIGARCH Computer Architecture News, 38, ACM, 2010, pp. 280–289.
[14] R. Ge, R. Vogt, J. Majumder, A. Alam, M. Burtscher, Z. Zong, Effects of dynamic voltage and frequency scaling on a k20 GPU, in: Parallel Processing (ICPP), 2013 42nd International Conference on, IEEE, 2013, pp. 826–833.
[15] I. Paul, V. Ravi, S. Manne, M. Arora, S. Yalamanchili, Coordinated energy management in heterogeneous processors, Sci. Program. 22 (2) (2014) 93–108.
[16] X. Mei, L.S. Yung, K. Zhao, X. Chu, A measurement study of GPU DVFS on energy conservation, in: Proceedings of the Workshop on Power-Aware Computing and Systems, ACM, 2013, p. 10.
[17] S. Huang, S. Xiao, W.-c. Feng, On the energy efficiency of graphics processing units for scientific computing, in: Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on, IEEE, 2009, pp. 1–8.
[18] Y. Abe, H. Sasaki, M. Peres, K. Inoue, K. Murakami, S. Kato, Power and performance analysis of gpu-accelerated systems, in: Proceedings of the 2012 USENIX conference on Power-Aware Computing and Systems, ser. HotPower, 12, 2012. 10–10
[19] Y. Jiao, H. Lin, P. Balaji, W.-c. Feng, Power and performance characterization of computational kernels on the GPU, in: Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on & Int'l Conference on Cyber, Physical and Social Computing (CPSCom), IEEE, 2010, pp. 221–228.
[20] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N.S. Kim, T.M. Aamodt, V.J. Reddi, Gpuwattch: enabling energy optimizations in gpgpus, ACM SIGARCH Compu. Archit. News 41 (3) (2013) 487–498.
[21] D. Price, M. Clark, B. Barsdell, R. Babich, L. Greenhill, Optimizing performance per watt on GPUs in high performance computing: temperature, frequency and voltage effects, arXiv:1407.8116(2014).
[22] O. Sarood, P. Miller, E. Totoni, L.V. Kale, 'Cool' load balancing for high performance computing data centers, Comput. IEEE Trans. 61 (12) (2012) 1752–1764.
[23] M. Chiesi, L. Vanzolini, C. Mucci, E.F. Scarselli, R. Guerrieri, Power-aware job scheduling on heterogeneous multicore architectures, Parallel Distrib. Syst. IEEE Trans. 26 (3) (2015) 868–877.
[24] J. Lee, V. Sathisha, M. Schulte, K. Compton, N.S. Kim, Improving throughput of power-constrained GPUs using dynamic voltage/frequency and core scaling, in: Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on, IEEE, 2011, pp. 111–120.
[25] S. Lee, H. Kim, Y. Chung, Power-time tradeoff of parallel execution on multi-core platforms, in: Mobile, Ubiquitous, and Intelligent Computing, Springer, 2014, pp. 157–163.
[26] K. Ma, X. Li, W. Chen, C. Zhang, X. Wang, Greengpu: a holistic approach to energy efficiency in GPU-CPU heterogeneous architectures, in: Parallel Processing (ICPP), 2012 41st International Conference on, IEEE, 2012, pp. 48–57.
[27] G. Wang, X. Ren, Power-efficient work distribution method for CPU-GPU heterogeneous system, in: Parallel and Distributed Processing with Applications (ISPA), 2010 International Symposium on, IEEE, 2010, pp. 122–129.
[28] M. Arora, S. Manne, I. Paul, N. Jayasena, D.M. Tullsen, Understanding idle behavior and power gating mechanisms in the context of modern benchmarks

on CPU-GPU integrated systems, in: High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on, IEEE, 2015, pp. 366–377.
[29] S. Collange, D. Defour, A. Tisserand, Power consumption of gpus from a software perspective, in: Computational Science–ICCS 2009, Springer, 2009, pp. 914–923.
[30] F. Fallah, M. Pedram, Standby and active leakage current control and minimization in CMOS VLSI circuits, IEICE Trans. Electron. 88 (4) (2005) 509–519.
[31] X. Fei, K. Li, W. Yang, K. Li, Velocity-aware parallel encryption algorithm with low energy consumption for streams, Transactions on Big Data, 99, IEEE, 2017, pp. 1–14.
[32] X. Fei, K. Li, W. Yang, K. Li, Practical parallel AES algorithms on cloud for massive users and their performance evaluation, Concurr. Comput. (2015).