Analyzing GPU Energy Consumption in Data Movement and Storage

Paul Delestrac, Jonathan Miquel, Debjyoti Bhattacharjee, Diksha Moolchandani, Francky Catthoor, Lionel Torres, David Novo

Analyzing GPU Energy Consumption in Data Movement and Storage

Paul Delestrac[†] Jonathan Miquel[†] Debjyoti Bhattacharjee[‡] Diksha Moolchandani[‡]
Francky Catthoor[‡*] Lionel Torres[†] David Novo[†]

[†]LIRMM, Univ. Montpellier, CNRS, Montpellier, France [‡]IMEC, Leuven, Belgium [*]KU Leuven, Leuven, Belgium

***Abstract*—GPUs are the prevailing solution to execute high-performance tasks (e.g., machine learning training). As the peak performance of modern GPUs increases with each generation, so does their thermal design power (TDP). Hence, identifying energy bottlenecks in the GPU architecture is crucial to designing more efficient architectures in the future. However, due to the complex proprietary nature of modern GPU architectures, providing a detailed breakdown of the GPU energy consumption is not trivial.**

The goal of this work is to estimate a lower bound for the energy consumed by data movement and storage in modern GPU architectures, leveraging internal power sensors. We establish a basic energy model for modern GPUs, focused on data movement to/from the hardware-managed caches and software-managed memories. We propose a methodology to calibrate the energy model using microbenchmarks, performance counters, and the internal power sensor. We experimentally calibrate the model on an A100 NVIDIA GPU. Then, we challenge the consistency of the results by cross-validating with modified microbenchmarks with additional instructions. Finally, we use the calibrated energy model to evaluate breakdowns for workloads of increasing complexity (e.g., a ResNet-50 training iteration with different software optimizations). Our results show that data movement dominates the dynamic energy consumption of the GPU (up to 84%), with DRAM accesses being the main contributor.

I. Introduction

Recent evolution of the GPU architecture (e.g., the addition of tensor cores in 2017) drastically improved throughput and latency, making GPUs the default solution for high-performance tasks (e.g., machine learning training). Despite these architectural innovations, GPUs still consume a considerable amount of energy (e.g., the NVIDIA A100 GPU features a 400W TDP, while the TDP of the upcoming NVIDIA Blackwell is estimated at 1000W). The design of energy-efficient accelerators relies on a deep understanding of the energy bottlenecks. However, modern GPU architectures are proprietary and complex, and manufacturer-provided profiling tools do not offer a detailed energy consumption breakdown.

Hence, architecture designers have to resort to simulation-based approaches to estimate the energy consumption of the inner GPU components [1]. This solution depends on the precision of the models and the simulation time can be extensively long for heavy workloads. Previous GPU power models were proposed to isolate the power consumption of specific GPU components using microbenchmarks [2]–[6]. However, some of these approaches were either: performed on older architectures and/or relied on external power measurement tools, which is impractical when the GPU is not physically accessible (e.g., cloud server); rely on analyzing PTX assembly code, which is not applicable when using vendors-provided precompiled libraries (e.g., cuDNN [7]); use Machine Learning (ML) models to predict power consumption [8]–[13] which obfuscates the power consumption details of the GPU component behind trained weights; or, ignore crucial kernel parameters that can influence the energy per access [3] (i.e., type of transaction, kernel dimensions, access pattern).

Our **main goal** is to propose a method that uses performance counters and internal power sensors to estimate a tight lower bound of the energy consumed by GPUs in data movement and storage. To this end, we establish a basic energy model of the GPU architecture, focused on data movement and storage across the memory hierarchy. Then, we propose a methodology to calibrate the proposed energy model for modern GPUs. This methodology combines profiling-specific microbenchmarks with tools that grant access to the GPU's performance counters and internal power sensors [14]. We cater our methodology for assessing a *lower bound* on the energy consumption of data movement and storage. We use our methodology to evaluate the energy consumption for accesses to shared memory, L1 and L2 caches, and DRAM of an NVIDIA A100 GPU. We show that ignoring crucial kernel parameters (e.g., kernel dimensions, access pattern) can lead to a $15\times$ overestimation of energy consumption. To cross-validate the methodology, we challenge the energy evaluations by performing the calibration on modified microbenchmarks (i.e., adding compute instructions to the memory accesses). Finally, we use our calibrated model to evaluate a breakdown of the energy consumption of the A100 GPU for increasingly complex ML workloads. Our results show that even a lower bound estimation of the energy consumed by data movement and storage remains a substantial portion of the total GPU energy, with DRAM accesses being the main contributor.

In summary, this work makes the following contributions:

- We propose a basic energy model for modern GPU architectures, focused on data movement and storage.
- We describe a methodology to calibrate the proposed energy model using microbenchmarks to assess a lower bound energy of data movement and storage.
- We calibrate our energy model on the NVIDIA A100 and check the consistency of the energy evaluations.
- We show that our model can provide a detailed breakdown of the energy consumption of complex applications.

II. Related Work

GPU power models were proposed in previous works, some of which use microbenchmarks for calibration [2]–[4], [10], [15], [16] and/or estimate either the total GPU power consumption [6], [8]–[12], [16], [17] or a power breakdown [4], [5], [16], [17]. We identify six main limitations of related works and discuss how our methodology addresses them.

Overevaluation of the energy. Some works ignore crucial kernel parameters [3] (i.e., transaction type, kernel dimensions, access pattern). Others, use microbenchmarks to calibrate power models and estimate the total power usage or provide a power breakdown, with reasonable accuracy [16]. However, these estimations can lead to overevaluation of the energy by memory operations, which can complicate the ranking of different options in the design space exploration.

Require access to PTX code. Some works rely on analyzing the PTX assembly code of GPU benchmarks [6], [11], [12], [17]–[19] or microbenchmarks [10]. These approaches are not applicable when the PTX code of the targeted GPU workloads is not accessible, which is the case in vendors-provided precompiled libraries often used by ML applications (e.g., cuDNN [7] is used by TensorFlow [20] and PyTorch [21]).

DL-based models. Some works rely on using DL-based models [8]–[13] trained to infer power predictions based on performance counter values. While these models can provide accurate power predictions, using DL models obfuscates the power consumption details of the GPU component behind trained weights. Hence, these models cannot provide a detailed power breakdown of the GPU architecture.

Ignore memory operations. Some works evaluate the energy cost for compute instructions but ignore memory instructions altogether [2] (i.e., only evaluating the energy consumption of compute operations). Similarly, some works focus only on the software abstractions of the GPU memory hierarchy [15], without distinguishing between the physical implementations. This can lead to inaccurate evaluations if performed on modern GPUs, which include hardware-managed cache hierarchies.

Require physical access. Some works [2]–[6] use external power measurement tools [22], building testbeds to measure the power consumption of the GPU. These methodologies are not relevant when physical access to the GPU is not available (e.g., cloud server). Other previous works have compared internal power sensor readings to external solutions [2], [23], showing that the error introduced by internal sensors can be made negligible with simple processing of the measurements.

Dated GPU architectures. Some works are based on dated GPU architectures [4]–[6]. These approaches are similar to ours (e.g., microbenchmarks, counters), but do not apply to modern GPUs with more complex memory hierarchies.

In contrast, we propose a methodology that takes into account major kernel parameters to provide a tight lower-bound estimation of the energy consumed by memory operations in the physical memory levels of modern GPU architectures (i.e., making it easier to rank different design options). This methodology does not require access to PTX code and can be used with internal or external power measurement techniques.

III. Background

A. GPU microarchitecture

GPUs execute *kernel* programs that detail the operations to be performed by one *thread*, replicated and executed in parallel in the GPU architecture. Threads are organized in *thread blocks*, and distributed to the different *Streaming Multiprocessors* (SM) cores of the GPU. When a thread block is scheduled for execution in an SM, its threads are distributed in groups of 32 (i.e., *warps*). Warp instructions can be issued in an interleaved manner and executed in parallel.

Modern GPUs' memory hierarchy is composed of several levels of cache to try and reduce the latency (and energy consumption) of memory accesses (i.e., LOADs or STOREs): L1 cache (private to each SM), L2 cache (common to multiple SMs), and DRAM (global memory). Accesses to these hardware caches are tagged as *HIT* or *MISS* depending on whether the targeted data is present in the cache or not. These hardware-managed caches are complemented by software-managed caches (e.g., texture cache, constant cache, shared memory) from (to) which a programmer can explicitly load (store) data. Each access to a memory preloads a complete *cache line* to be accessed by the threads before the next access replaces it. The size of the cache line is known as the *access granularity* of the memory (e.g., 32 bytes for NVIDIA A100). The proposed energy model covers both types of memories.

B. GPU performance counters and internal power sensors

NVIDIA GPUs provide access to a set of performance counters through the NVIDIA Nsight Compute (ncu) profiling tool [24]. Performance counters are exposed to the user through a set of *metrics*. A limited number of performance counters can be observed during a profiling run, above which multiple replays of the profiled kernels will be prompted by the tool, adding profiling overhead. This work selects specific metrics to evaluate the number of memory transactions at each level of the GPU memory hierarchy (see Section V).

Modern NVIDIA GPUs provide access to an internal power sensor using the NVIDIA Management Library (NVML). Limitations of this sensor were detailed in previous works [23] and evaluated against external power sensors [2], [23]. This work builds upon the methodology from Burtscher et al. [23] to collect power measurements. We provide more details on our power measurement methodology in Section V.

IV. Energy model of data movement in GPUs

In this section, we propose an energy model to characterize the energy consumption of data movement in the GPU memory hierarchy, noted E_{MEMORY}. We first provide background on the different high-level components of the GPU energy model. Then, we model data movement and storage based on our analysis of modern GPU architectures. The proposed analytical energy model covers software-addressed memory spaces and hardware-managed caches. Finally, we discuss the influence of parallelized transactions on the energy consumption of the memory hierarchy.

A. Background on GPU energy models

The energy consumption of a GPU running a kernel can be divided between static and dynamic energy components:

$$E_{\text{TOTAL}} = E_{\text{STATIC}} + E_{\text{DYNAMIC}}. \quad (1)$$

The static energy is the part of the total energy consumed by the GPU, which is constant (i.e., independent of the kernel being executed). This static energy component is specific to a given GPU model and influenced by the temperature, the voltage, and the frequency of the GPU. The dynamic energy is the additional energy (i.e., on top of the static energy) consumed by the GPU when it is executing a kernel. This dynamic energy component has been modeled in the past as the sum of the energy consumed by the memory (E_{MEMORY}) and the energy consumed by the SMs (E_{COMPUTE}) [4], [25]:

$$E_{\text{DYNAMIC}} = E_{\text{MEMORY}} + E_{\text{COMPUTE}}. \quad (2)$$

Static energy (E_{STATIC}) and compute energy (E_{COMPUTE}) have been modeled further in the past [2], [4]. Our work focuses on providing a tight lower-bound estimation of the energy consumed in data movement and storage (E_{MEMORY}).

B. Analytical energy model

We consider the energy spent moving data across the GPU cache hierarchy as the sum of energy spent loading (storing) data from (to) the memory levels, including both hardware-managed (i.e., caches) and software-addressed memory spaces (i.e., main memory and shared memory). Hence,

$$E_{\text{MEMORY}} = \sum_{\text{MEM}} E_{\text{MEM}}. \quad (3)$$

We define the energy spent by a given level of memory (E_{MEM}) as the number of accesses (noted #accesses) multiplied by the energy of one access to the memory (noted ε_{MEM}). However, accessing a memory level also requires the activation of the components of the memory level, which we evaluate as an offset energy (noted ΔE_{MEM}). In Section V, we propose a methodology to isolate the energy cost of one HIT access from the offset energy using linear regression:

$$E_{\text{MEM}} = \#\text{accesses} \times \varepsilon_{\text{MEM}} + \Delta E_{\text{MEM}}. \quad (4)$$

In the case of hardware-managed caches (L1 to LLC), a MISS will be matched to either a HIT at a lower cache level or access to the main memory. Hence, for these memories, we restrict the count of the number of accesses only to HIT accesses.

C. Influence of parallel accesses

The GPU architecture is intended to achieve better energy efficiency in two ways: by accessing a given memory using multiple threads at a time (i.e., parallel accesses) and/or by amortizing the energy cost of loading a cache line with subsequent memory accesses (i.e., coalescence). Hence, the energy cost of one memory access (noted ε) to a given memory level is influenced by three parameters: the number of threads accessing the memory at a time (noted N_T), the coalescence of these accesses and the access granularity of the memory (i.e., size of the cache line, noted C) [26]. Ideally, threads memory operations are subsequent and fully coalesced and the number of accesses can be defined as the number of entire cache lines that a given amount of threads will load

$$\#\text{accesses} = \left\lceil \frac{N_T}{C} \right\rceil \times C. \quad (5)$$

In this case, better energy efficiency per access can only be achieved by having enough threads to occupy all memory ports in parallel, capitalizing even more on the energy cost of activation of the memory,

$$\varepsilon = \frac{E_{\text{MEM}} - \Delta E_{\text{MEM}}}{\left\lceil \frac{N_T}{C} \right\rceil \times C}. \quad (6)$$

In Section V, we increase the number of threads when accessing a memory level to find the lower bound of Eq. 6.

V. Methodology

In this section, we describe a methodology to evaluate a lower bound for the energy consumption of data movement and storage in the GPU memory hierarchy (illustrated in Fig. 1). The methodology is divided into 3 different phases: identification of parameters of the device (V-A), calibration of the energy model using microbenchmarks (V-B), and evaluation of the energy breakdown of a new application (VI).

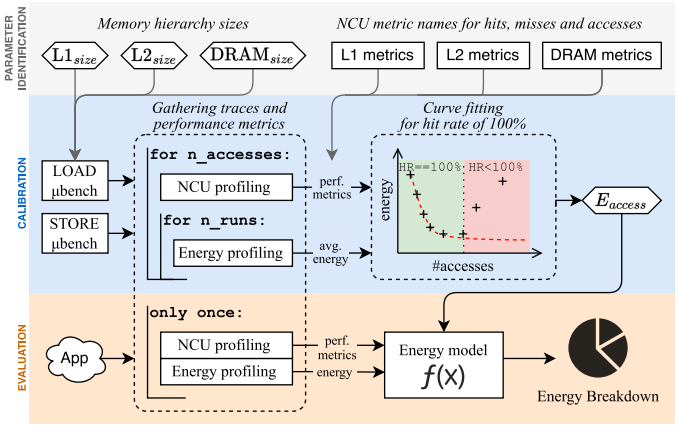

Fig. 1: Different phases of the methodology: (1) parameters identification, (2) calibration, and (3) evaluation

A. Parameter identification phase

Calibrating the proposed energy model relies on two specific sets of parameters, specific to the target GPU model. The first input parameter is the list of performance counters that count the number of accesses at each cache level (along with the proportion of *HIT*s and *MISS*es). This parameter is needed as a safety check to ensure that the microbenchmarks are targeting the right cache level during the calibration phase. The second parameter is the list of sizes of each GPU cache level. The size of the targeted GPU cache is needed as a direct input for the microbenchmarks when calibrating the energy model. The sizes of the GPU caches are typically shared by the manufacturer in the GPU documentation (e.g., NVIDIA's

whitepapers [27], [28]). However, the sizes of the GPU caches can also be identified by evaluating the hit/miss rate of the cache over a range of array sizes [29], [30].

For example, Fig. 2 shows the miss rate of the A100 GPU L1 and L2 caches over a range of array sizes. L1 cache miss rate increases as the array size passes 150kB and misses 100% of the time for an array size of 190kB. L2 cache miss rate increases as the array size reaches 15MB and misses 100% of the time for an array size of 22MB. NVIDIA's whitepaper [28] states that shared memory and L1 cache share a configurable 192kB memory space. L2 cache is 40MB, divided into 2 partitions of 20MB each. The results of this evaluation are consistent with the manufacturer's documentation.

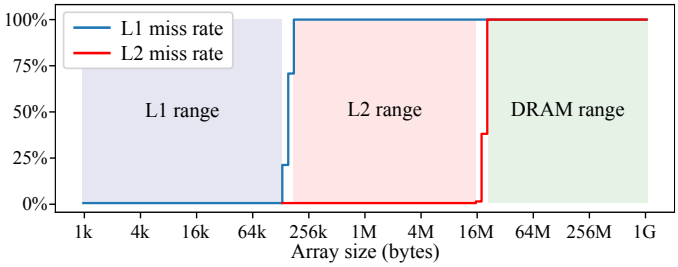

Fig. 2: Miss rate of the L1 and L2 caches of the A100 GPU, over a range of array sizes between 1kB and 1GB

B. Calibration phase

Once the parameters of the device are identified, we calibrate the energy model by running the microbenchmarks on the target GPU, sweeping the number of memory accesses while gathering the performance counters and power measurements, for each memory level. Here, we describe the microbenchmarks and important parameters of the calibration.

Microbenchmarks. We divide the proposed microbenchmarks into two codebases (written in CUDA with inline PTX assembly code): one for *LOAD* operations and one for *STORE* operations. A simplified version of the *LOAD* microbenchmark is shown in Listing 1. Both microbenchmarks are based on the pointer-chasing algorithm used in previous works [3], [29], which consists of a loop that iterates over an array of pointers, each pointing to another element of the array, separated by a given *stride*. The objective of the microbenchmarks is to perform a large number of memory transactions on an array of 64-bit unsigned integers (i.e., `uint64_t`), the size of the array is chosen to fit on the targeted cache level according to the identified parameters (*see Fig. 2*). Using pointer-chasing allows for a higher ratio of memory transactions versus other instructions when executing the microbenchmarks.

Listing 1 shows a typical microbenchmark containing two loops: a *warm-up* loop and a *measurement* loop. The warm-up loop (lines 13 to 17) is used to move the pointer-chasing addresses from the GPU global memory to the targeted cache. During the execution of this loop, all memory accesses to the targeted cache level will be tagged as *MISS*es, as the data is not yet cached. This example microbenchmark is designed to target hardware-managed caches (i.e., L1, L2). However, it can easily be adapted to target other memory spaces (e.g., shared memory) by manually programming the filling of the targeted memory space instead of the warm-up loop. In the case of the DRAM, the warm-up loop is not needed as the data is already in the DRAM memory.

The measurement loop (lines 18 to 27 in Listing 1) follows the same pattern as the warm-up loop, however, we unroll the loop to increase the ratio of memory instructions per iteration, reducing the contribution of other instructions to the total energy consumption. Additionally, we execute multiple iterations of the measurement loop to increase the duration of the microbenchmark, reducing the contribution of possible ramp up and ramp down of the power consumption of the GPU. Thanks to the warm-up loop, all memory accesses to the targeted cache level should be tagged as *HIT*s during the execution of the measurement loop. For each measurement point, we execute the microbenchmarks twice: once with only the warm-up loop and once with both the warm-up and measurement loops. Then, we isolate the energy consumption and performance counter values of the measurement loop by subtracting the corresponding values of the warm-up loop.

```
1  __global__ void load(uint64_t *array) {
2      // Get thread ID and block ID
3      uint64_t tid = threadIdx.x;
4      uint64_t bid = blockIdx.x;
5      // Compute thread start address
6      uint64_t *start = array + tid + (bid *
           SUBTAB_SIZE / sizeof(uint64_t));
7
8      asm volatile(
9      [...] // Init registers
10     "{.reg .pred %p;\n"
11     ".reg .u64 %tmp;\n"
12     "mov.u64 %tmp, %0;\n\n"
13     // Warmup loop
14     "$warmup:\n"
15     "ld.global.u64 %tmp, [%tmp];\n"
16     "setp.ne.u64 %p, %tmp, %0;\n"
17     "@%p bra $warmup;\n"
18     // Measurement loop
19     [...] // Reset counter and address
20     "\n$measurement:\n"
21     "ld.global.u64 %tmp, [%tmp];\n" // 1st LOAD
22     [...] // unrolling X more LOADs
23     "setp.ne.u64 %p, %tmp, %0;\n"
24     "@%p bra $measurement;\n" // for SIZE/STRIDE
25     "add.u32 %k, %k, 1;\n"
26     "setp.lt.u32 %p, %k, %1;\n"
27     "@%p bra $measurement;\n" // for N_ITER
28     "}" : "+l"(start)
29     : "n"(N_ITER));
30 }
```

Listing 1: Simplified LOAD microbenchmark kernel code

Access granularity and access pattern. Ideally, optimized kernels should exploit the cache line locality by accessing the memory in a coalesced and parallel way across multiple threads. Hence, using a single thread at a time to access memory can lead to overestimation of energy consumption per access. The proposed methodology assumes such ideal memory usage and is designed to assess a *lower bound* energy consumption for memory accesses. As mentioned in Sec-

tion IV-C, to evaluate this lower bound, we have to take into account two effects: multiple threads accessing the same cache line in *parallel* and/or each thread accessing the elements of the cache line in a *subsequent* coalesced way.

Different memory levels can have different access granularities. For example, the granularity of the A100's L2 cache can be configured to 32, 64, or 128 bytes. This must be taken into account to prevent the generation of undesired HITs when targeting hardware-managed caches. To have consistently-strided accesses, we set the minimum stride of the pointer-chasing algorithm to the access granularity size of the targeted memory level. For example, the A100 GPU has a LOAD access granularity of 32 bytes (i.e., a *sector*, 4 `uint64_t` elements). Hence, we run the microbenchmarks with a stride of 32 bytes to access the array.

However, with this stride, using only 1 thread per thread block would underuse the sector, loading only a fraction of the accessed elements (*see Fig. 3a*). As each LOAD operation of one thread will trigger a different memory access, this could lead to an unrealistically high energy consumption per access. Typically, multiple GPU threads should access the same sector at the same time (in our case, 4 threads to access a sector of 32 bytes to amortize the energy cost of accessing a sector, *see Fig. 3b*). Past this point, increasing the number of threads per block (*see Fig. 3c*) could still help reduce the energy consumption per memory transaction for two reasons: (1) it provides greater opportunities to parallelize memory transactions, thereby using multiple memory ports and reducing the execution time of the microbenchmark, and (2) it facilitates leveraging the fixed energy cost of using the targeted cache level (i.e., activating the necessary components to access the cache level).

The increase in the number of parallel accesses can also be achieved by increasing the number of thread blocks, leading to a further reduction in energy consumption per memory transaction. From a practical point of view, using more threads also increases the dynamic energy of the GPU, ultimately increasing the signal-to-noise ratio of the power measurements. Hence, in our experiments, we sweep the number of blocks and the number of threads per block until we reach a low saturation point for the energy per access. This evaluates the lower bound of the energy consumption of memory transactions.

Performance counters. The theoretical number of accesses during one iteration of the microbenchmarks is the array size divided by the stride size and the number of threads executed. Practically, we evaluate it using the following formula:

$$\#\text{accesses} = \frac{\text{arraysize} \times \#\text{blocks} \times \#\text{threadsperblock}}{\text{stride} \times \text{access granularity}}. \quad (7)$$

As described in Section IV, the total energy spent increases linearly with the number of accesses but also hides a static offset energy resulting from the activation of auxiliary components (e.g., memory controllers, SMs, etc.). Hence, we evaluate the total energy for multiple amounts of memory accesses and use linear regression to separate this energy offset from the energy cost of each memory transaction. For each measurement point,

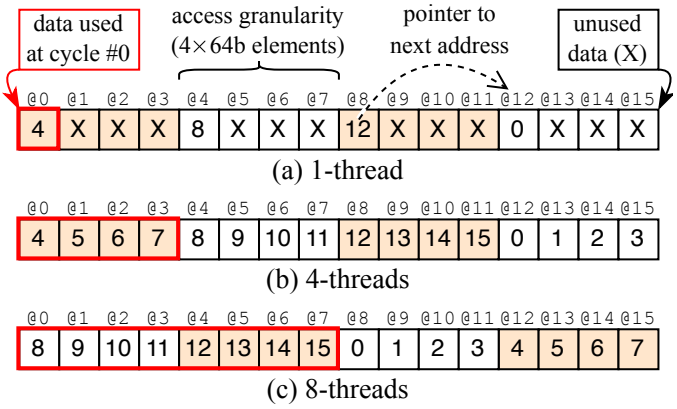

Fig. 3: Thread accesses of the microbenchmarks for different numbers of threads per block, adapting the stride accordingly

we run the profiling once and use the performance counters as a safety check to ensure that two conditions are met. First, we verify that the memory accesses are performed on the targeted memory level, by checking that the hit rate of the target cache level is 100% during the measurement loop. Second, we verify that the counted number of memory accesses is consistent with Eq. 7. Any difference in the evaluated number could indicate changes in the microarchitecture. Thus, parameters would have to be changed accordingly.

Power measurements. We manually add delays in the CPU code before and after the execution of the kernel to ensure we measure the static power consumption of the GPU during a period of inactivity (i.e., idle). While we use the internal power sensor of the GPU in our implementation of the methodology, external power tools could also be used with this alignment technique using delays. In Fig. 4, we show a typical power trace of the A100 GPU, using base clock (1065 MHz), executing a kernel that is representative of the proposed microbenchmarks in its power consumption (i.e., multiply-add operations on a vector with millions of values), during 10 seconds. The power trace is annotated with markers (`t0` to `t5`), separating the different phases of execution. Between `t0` and `t1`, no kernel is executing and no memory is allocated on the GPU, the GPU is in a *sleep* state ($\simeq$ 55 W for the A100). Between `t1` and `t2`, the memory is allocated on the GPU global memory and the GPU enters an *idle* state. During this idle phase, we measure the static power consumption of the GPU. Between `t2` and `t3`, during the 10s of *kernel execution*, the power consumption of the GPU increases depending on the kernel's load. During this phase, we identify the energy consumption above the idle power level as the *dynamic energy* (i.e., green zone in Fig. 4). We identify the rest of the energy consumption as the *static energy* (i.e., red zone in Fig. 4). When executing the microbenchmarks, the duration of this phase can be influenced by increasing or decreasing the number of iterations executed during the measurement loop. Kernel execution ends at `t3`, and the GPU returns to an idle state. Finally, at `t4`, the memory is deallocated and the GPU

goes back to a sleep state until the end of the run at `t5`.

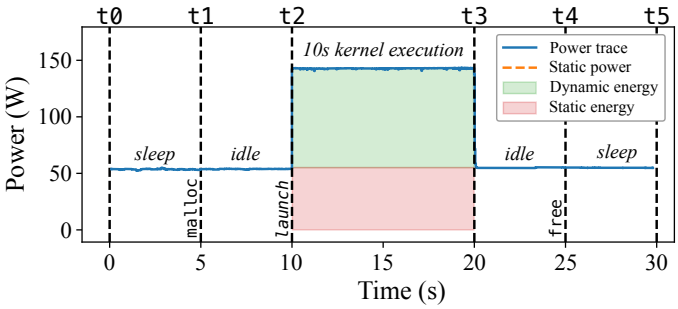

Fig. 4: Power trace of a 10s kernel execution on the A100 GPU with base clock, annotated with markers

Experimentally, we use the NVIDIA Management Library (NVML) to collect power measurements from the GPU and run each measurement point multiple times (at least 3). Using base clock, we observe no difference in power consumption between the sleep and idle phases. However, this is not the case when using boost clock, where we observe a rise in the idle phase to around $80W$ and a *cooldown* phase at `t4`, where the power gradually lowers down to the power value observed in the sleep phase. Hence, we lock the GPU clock at base clock (1065 MHz) to help prevent power fluctuations that can arise from thermal throttling or changes in power modes.

C. Evaluation phase

Once the energy consumption per memory access is isolated for each level of the memory hierarchy, the calibrated energy model can be used to provide energy breakdowns for new applications. To this end, the new application has to be profiled with the same tools used during the calibration phase. However, while the calibration phase needs multiple runs of profiling (i.e., `for n_accesses` and `for n_runs` in Fig. 1), the evaluation phase only needs two runs, one for each profiling tool (i.e., NCU profiling and energy profiling). Our calibration methodology provides a lower-bound evaluation of the energy consumption and covers only the energy consumption of the data movement and storage. Hence, the measured dynamic energy will be higher than the sum of the energy consumption of the different memory levels. This difference includes the energy of the compute instructions (i.e., E_{COMPUTE} in Section IV) and additional energy from memory accesses exceeding the lower bound. In Section VII, we identify this difference as the remaining component of the dynamic energy (i.e., "Dyn. (rest)" in Figs. 9 and 10).

VI. Implementation

Using the methodology described in Section V, we calibrate the proposed energy model for the NVIDIA A100 GPU.

A. Experimental setup

We use the NVIDIA CUDA Toolkit 12.1 to compile our microbenchmarks and run our experiments on a cloud server equipped with A100 paired with AMD EPYC 7343. We use the NVIDIA Management Library (NVML) to read the internal power sensor measurements and lock the clock frequency of the GPU to 1065 MHz (i.e., base clock). We use NVIDIA Nsight Compute CLI (ncu) to read the performance counters. We calibrate the energy model for the L1 cache, L2 cache, DRAM, and shared memory, using only the LOAD microbenchmark. We consider the energy of a STORE access energy consumption as the same as a LOAD access to a given memory. This respects our lower bound assumption, as the energy consumption of a STORE access is usually higher than a LOAD access [31]. For each measurement point, we run at least 3 iterations and report the average energy consumption, standard deviation, and minimum/maximum values.

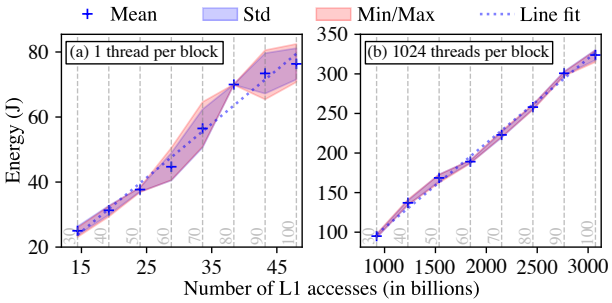

Fig. 5: Calibration traces of the L1 cache LOAD microbenchmark with (a) one thread per block, (b) 1024 threads per block

B. Parameter identification

According to the manufacturer's whitepaper [28], the A100 GPU is equipped with 108 SMs (each with 4 SMSPs), 80 GiB of HBM2 memory, 40 MiB of L2 cache, and 192 kiB of combined shared memory/L1 cache. In Section V, we show that MISSes can occur with smaller array sizes than the cache size. Hence, we choose the array sizes for our microbenchmarks small enough to fit in the targeted cache: 150 kB for the L1 cache, 250 kB for the L2 cache, and 50 MB for the DRAM. For shared memory, we use a 48 kB array size, as it is the maximum static allocation size per block on the A100 GPU [32].

C. Calibration

Fig. 5 shows the total energy consumption of the A100 GPU when running the *LOAD* microbenchmark with a varying number of memory accesses, targeting the L1 cache (i.e., 150 kB array size), using (a) one thread per block and (b) 1024 threads per block. We observe that the energy consumption increases linearly with the number of memory accesses. We use linear regression to calibrate the energy model and evaluate the energy consumption of each memory access. Linear regressions achieved r-squared scores of 0.96 and 0.99 for 1 and 1024 threads per block, respectively. The standard deviation of our measurements shows a maximum value of around 6 J, which is acceptable relative to the total energy consumption. Thus, we observe a higher precision of the calibration when using a higher number of threads per block.

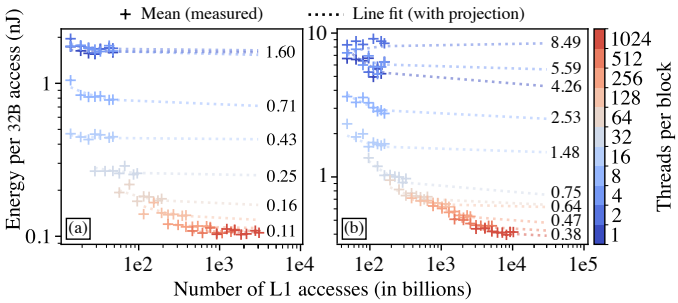

Fig. 6: L1 (a) and L2 (b) cache LOAD calibration results

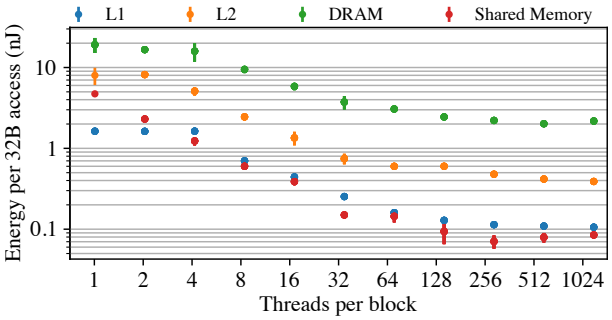

Fig. 7: L1, L2, DRAM and shared memory calibration results

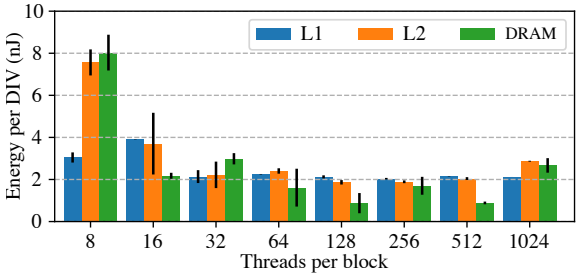

Fig. 8: DIV instruction energy evaluation based on the difference with the LOAD calibration results (L1, L2, and DRAM)

Using the slope of the linear fit, we can estimate the energy consumption of each memory access, which is around 1600 pJ for the L1 cache using one thread per block and 107 pJ for 1024 threads per block. As described in Section V, the energy consumption of one memory access can vary depending on the number of threads per block. Hence, we repeat this calibration process for the L1 and L2 caches using multiple threads per block, up to 1024 threads per block (i.e., the maximum number of threads per block on the A100 GPU for 1D grids). We present the results of this calibration in Fig. 6a, along with the L2 cache calibration (Fig. 6b). We can see that below 4 threads per block, the evaluated energy per access is similar (nearly 1600 pJ). With a low number of threads per block, we observe high signal-to-noise, hence the variation of the projections for the L2 cache (ranging between 8490 and 4260 pJ). As mentioned in Section V-B, due to the access granularity, using less than 4 threads per block the proposed microbenchmark will always prompt the same number of sector accesses. When increasing the number of threads per block, we observe a plateau in the evaluated energy consumption of each memory access decreasing to around 107 pJ for L1 and 378 pJ for L2. These results show that ignoring multithreaded memory accesses leads to near 15× overevaluated energy per access (from 1600 to 107 pJ). We consider the lower bound of the energy consumption (i.e., 107 pJ for L1 and 378 pJ for L2).

We repeat this calibration process for DRAM and shared memory (i.e., 50 MB and 48 kB array size, respectively) using multiple threads per block. We present the calibration results in Fig. 7, showing the projection of the energy consumption of each memory access for different numbers of threads per block. Similarly to the L1 cache calibration, we observe that the energy consumption per access decreases when increasing the number of threads per block, for all memory levels. We evaluate the lower bound of the energy consumption of one access at a given memory level by taking the minimum value across all evaluated numbers of threads per block. Hence, we evaluate the minimum energy consumption of a sector access (i.e., 32 bytes) to the shared memory, L1 cache, L2 cache, and DRAM to be: 82.1, 107, 368 and 2090 pJ, respectively.

The difference in power consumption between L1 and shared memory (located on the same physical memory) can be explained by the difference in how the memory is addressed (i.e., shared memory is directly addressed, L1 cache has a tag stage checking if the data is present in the cache).

D. Cross-validation of the calibration results

The DRAM memory of the A100 uses HBM2 technology. Previous works estimate HBM2 memory energy consumption at 500 pJ per 32B access [31]. With our methodology, we evaluate this energy for the A100 at 2090 pJ. Considering additional energy overhead from DRAM and cache controllers, we consider this value consistent with the literature.

To challenge the consistency of our results, we use a modified version of the *LOAD* microbenchmark, with a supplemental *DIV* instruction after each memory access (i.e., *LOAD+DIV*), and perform the same calibration. We select the DIV instruction as it consumes significant energy compared to other compute instructions [2], making it simpler to isolate from the noise of the measurements. We evaluate the difference in energy consumption between the *LOAD* calibration and the *LOAD+DIV* calibration for the L1, L2, and DRAM memory levels. We present the results in Fig. 8.

We observe some variations in the evaluated energy consumption of the *DIV* instruction when using a small number of threads per block (i.e., with the highest signal-to-noise ratio). Above 8 threads per block, we observe that the energy consumption of the *DIV* instruction is consistent at around 2 nJ per instruction. This value is consistent with the literature, as the energy consumption of an unsigned division instruction was evaluated around 3.9 nJ on previous NVIDIA GPUs [2].

VII. Evaluation of Complete Applications

We use the calibrated model to evaluate energy breakdowns for two real-world applications with increasing complexity: a matrix multiplication (MatMul) and a training iteration of a deep learning model (i.e., ResNet-50). We perform 10 runs for each application and report less than 1% of standard deviation.

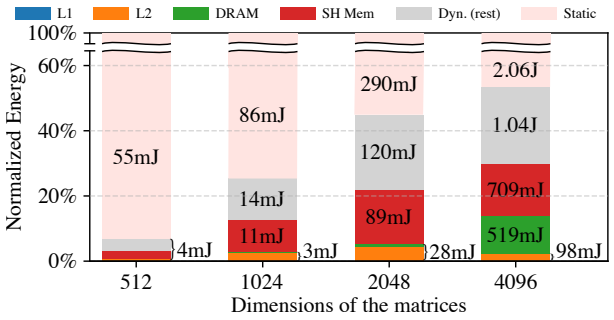

Fig. 9: Energy breakdown of MatMul for multiple sizes

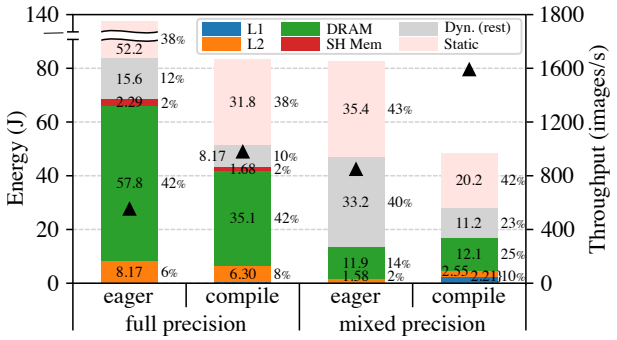

Fig. 10: Energy breakdown and throughput of ResNet50 using multiple software optimizations, executed using TensorFlow

A. Matrix multiplication

We use a reference MatMul implementation from the CUDA samples (which does not use the tensor cores [19]). We evaluate the energy consumption for multiple square matrix sizes (from 512 to 4096) and show the breakdowns in Fig. 9.

We observe that while the energy consumed during the execution of the kernel increases with the size of the matrices, it is dominated by the static energy consumption of the GPU (i.e., $> 50\%$). When ignoring the static energy consumption, we observe that a major part of the energy is consumed by moving data across the memory hierarchy (i.e., $> 50\%$ of the dynamic energy), with the shared memory being the most energy-consuming memory component.

The largest tested matrix multiplication (i.e., 4096×4096) has an arithmetic intensity of 1365 FLOPS/byte, which makes it compute-bound for the A100 GPU (i.e., arithmetic intensity $\gg 1$ FLOP/byte). Nevertheless, we observe that the energy consumption of data movement is still significant, representing around 30% of the total energy consumption of the kernel. This simple example shows the significant energy consumed in data movement and storage in the A100, even for a lower-bound evaluation of compute-bound workloads.

B. ResNet-50 training iteration

We use a reference implementation of ResNet-50 from the TensorFlow model garden [33] and evaluate the breakdown of the energy consumption of one training iteration using the A100 GPU. We run the training iteration comparing multiple software optimizations (i.e., full precision (FP32) vs. mixed precision (FP16), and eager execution vs. just-in-time (JIT) compilation using XLA) using batches of 512 images. We present the results of this evaluation in Fig. 10.

We make four observations from these results. First, the kernel's static energy is around 40% of its total energy consumption, regardless of the software optimization used. Second, data movement is the most energy-consuming component, representing the majority of the dynamic energy consumption (i.e., between 60% and 84%), except for the AMP eager execution (i.e., only 29%). Third, both of the tested software optimizations reduce the energy consumption of the kernel, showing a potential drop of 90 J going from full-precision eager to mixed-precision XLA JIT compilation. However, these optimizations influence the energy consumption breakdown differently. On the one hand, going from full-precision to mixed-precision greatly reduces the energy of data movement and storage. When using eager execution it represents a drop from 80% to 29% of the dynamic energy, and a drop from 83% to 60% when using XLA JIT compilation. On the other hand, XLA JIT compilation increases the proportion of memory energy consumption. It represents an increase from 29% to 60% of the dynamic energy consumption with mixed-precision, and from 80% to 84% when using full precision. Finally, we observe that DRAM greatly dominates the energy consumption of the memory hierarchy (above 70%). In contrast, the L2 cache represents between 2% and 8% of the total energy consumption. The L1 cache is negligible, representing less than 1% of the total energy consumption, except for the AMP XLA JIT compilation, where it represents around 5%. Shared memory is also negligible, representing less than 2% of the total energy consumption for all tested configurations.

While the proposed methodology evaluates a lower bound of the data movement and storage energy, our results show that it still represents a significant part of the total energy consumption of the GPU, even for compute-bound workloads. On the one hand, the calibration results highlight the importance of exploiting data locality when aiming for energy-efficient GPU applications. On the other hand, the energy breakdown results show that even a well-optimized deep learning model, exploiting data locality and reduced precision, is still dominated by the energy consumption of data movement.

VIII. Conclusion

In this work, we propose a methodology to estimate a lower bound of the energy consumed by data movement and storage. The proposed methodology, which we implement and verify on an NVIDIA A100 GPU, uses microbenchmarks and performance counters, and identifies the energy consumption of accesses to shared memory, L1 and L2 cache, and DRAM memory. We evaluate breakdowns of the energy consumption for increasingly complex GPU workloads: a MatMul kernel with varying matrix sizes, and a training iteration of an ML model (i.e., ResNet-50) with different software optimizations. Despite optimizations lowering the total energy consumed, we show that data movement is dominant in the GPU dynamic energy consumption, with DRAM as the main contributor.

REFERENCES

[1] J. Chen, B. Li, Y. Zhang, L. Peng, and J. Peir, "Statistical GPU power analysis using tree-based methods," in *Proceedings of IGSC*. IEEE, 2011.

[2] Y. Arafa, A. ElWazir, A. ElKanishy, Y. Aly, A. Elsayed, A.-H. Badawy, G. Chennupati, S. Eidenbenz, and N. Santhi, "Verified instruction-level energy consumption measurement for NVIDIA GPUs," in *Proceedings of CF*. ACM, 2020.

[3] N. Bombieri, F. Busato, F. Fummi, and M. Scala, "MIPP: A microbenchmark suite for performance, power, and energy consumption characterization of GPU architectures," in *Proceedings of SIES*. IEEE, 2016.

[4] S. Hong and H. Kim, "An integrated GPU power and performance model," in *Proceedings of ISCA*. ACM, 2010.

[5] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, "Statistical power modeling of GPU kernels using performance counters," in *Proceedings of IGSC*. IEEE, 2010.

[6] C. Luo and R. Suda, "A performance and energy consumption analytical model for gpu," in *International Conference on Dependable, Autonomic and Secure Computing*. IEEE, 2011, pp. 658–665.

[7] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. (2014) cuDNN: Efficient primitives for deep learning.

[8] S. Song, C. Su, B. Rountree, and K. W. Cameron, "A simplified and accurate model of power-performance efficiency on emergent GPU architectures," in *Proceedings of IPDPS*. IEEE, 2013.

[9] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou, "GPGPU performance and power estimation using machine learning," in *Proceedings of HPCA*. IEEE, 2015.

[10] J. Guerreiro, A. Ilic, N. Roma, and P. Tomás, "GPU static modeling using PTX and deep structured learning," *IEEE Access*, vol. 7, pp. 159 150–159 161, 2019.

[11] L. Braun, S. Nikas, C. Song, V. Heuveline, and H. Fröning, "A simple model for portable and fast prediction of execution time and power consumption of GPU kernels," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 18, no. 1, pp. 1–25, 2020.

[12] C. A. Metz, M. Goli, and R. Drechsler, "Pick the right edge device: Towards power and performance estimation of CUDA-based CNNs on GPGPUs," in *Proceedings of DATE SLOHA Workshop*, 2021.

[13] D. Moolchandani, A. Kumar, and S. R. Sarangi, "Performance and power prediction for concurrent execution on GPUs," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 19, no. 3, pp. 1–27, 2022.

[14] "ADAC GPU Power Experiments." [Online]. Available: https://gite.lirmm.fr/adac/gpu-power-experiments

[15] T. Allen and R. Ge, "Characterizing power and performance of GPU memory access," in *Proceedings of E2SC*. IEEE, 2016.

[16] V. Kandiah, S. Peverelle, M. Khairy, J. Pan, A. Manjunath, T. G. Rogers, T. M. Aamodt, and N. Hardavellas, "Accelwattch: A power modeling framework for modern gpus," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 738–753.

[17] G. Alavani, J. Desai, S. Saha, and S. Sarkar, "Program analysis and machine learning–based approach to predict power consumption of CUDA kernel," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, vol. 8, no. 4, pp. 1–24, 2023.

[18] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proceedings of IISWC*. IEEE, 2009, pp. 44–54.

[19] NVIDIA, "NVIDIA CUDA samples." [Online]. Available: https://github.com/NVIDIA/cuda-samples

[20] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: A system for large-scale machine learning," in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, 2016.

[21] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "PyTorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, 2019.

[22] D. Bedard, M. Y. Lim, R. Fowler, and A. Porterfield, "Powermon: Fine-grained and integrated power monitoring for commodity computer systems," in *Proceedings of SoutheastCon*. IEEE, 2010.

[23] M. Burtscher, I. Zecena, and Z. Zong, "Measuring GPU power with the K20 built-in sensor," in *Proceedings of GPGPU Workshop*. ACM, 2014.

[24] "NVIDIA Nsight Compute." [Online]. Available: https://developer.nvidia.com/nsight-compute

[25] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: methodology and empirical data," in *Proceedings of MICRO*. IEEE, 2003.

[26] J. Jeffers and J. Reinders, *High performance parallelism pearls volume two: multicore and many-core programming approaches*. Morgan Kaufmann, 2015.

[27] NVIDIA, "NVIDIA Volta architecture whitepaper," 2017. [Online]. Available: https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

[28] NVIDIA, "NVIDIA A100 GPU whitepaper." [Online]. Available: https://www.nvidia.com/en-us/data-center/a100/

[29] X. Mei and X. Chu, "Dissecting GPU memory hierarchy through microbenchmarking," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, pp. 72–86, 2016.

[30] Q. Huppert, T. Evenblij, M. Perumkunnil, F. Catthoor, L. Torres, and D. Novo, "Memory hierarchy calibration based on real hardware in-order cores for accurate simulation," in *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021.

[31] J. M. O'Connor *et al.*, "Energy efficient high bandwidth dram for throughput processors," Ph.D. dissertation, UT Austin, 2021.

[32] NVIDIA, "NVIDIA Ampere GPU architecture tuning guide." [Online]. Available: https://docs.nvidia.com/cuda/ampere-tuning-guide/

[33] H. Yu, C. Chen, X. Du, Y. Li, A. Rashwan, L. Hou, P. Jin, F. Yang, F. Liu, J. Kim, and J. Li. TensorFlow Model Garden. [Online]. Available: https://github.com/tensorflow/models