Understanding the Future of Energy Efficiency in Multi-Module GPUs

Akhil Arunkumar*, Evgeny Bolotin†, David Nellans†, and Carole-Jean Wu*
*Arizona State University, † NVIDIA
Email: {akhil.arunkumar, carole-jean.wu}@asu.edu, {ebolotin, dnellans}@nvidia.com

***Abstract*—As Moore's law slows down, GPUs must pivot towards multi-module designs to continue scaling performance at historical rates. Prior work on multi-module GPUs has focused on performance, while largely ignoring the issue of energy efficiency. In this work, we propose a new metric for GPU efficiency called EDP Scaling Efficiency that quantifies the effects of both strong performance scaling and overall energy efficiency in these designs. To enable this analysis, we develop a novel top-down GPU energy estimation framework that is accurate within 10% of a recent GPU design. Being decoupled from granular GPU microarchitectural details, the framework is appropriate for energy efficiency studies in future GPUs. Using this model in conjunction with performance simulation, we show that the dominating factor influencing the energy efficiency of GPUs over the next decade is GPU-module (GPM) idle time. Furthermore, neither inter-module interconnect energy, nor GPM microarchitectural design is expected to play a key role in this regard. We demonstrate that multi-module GPUs are on a trajectory to become 2× less energy efficient than current monolithic designs; a significant issue for data centers which are already energy constrained. Finally, we show that architects must be willing to spend more (not less) energy to enable higher bandwidth inter-GPM connections, because counter-intuitively, this additional energy expenditure can reduce total GPU energy consumption by as much as 45%, providing a path to energy efficient strong scaling in the future.**

I. Introduction

GPUs are the defacto standard for application acceleration in the high performance computing (HPC), datacenter, and machine learning domains. This is because each new generation of GPUs has supported both the expansion of problem sizes (weak scaling) and provided seamless performance scalability for constant problem sizes (strong scaling). To support the strong scaling demands of applications, GPU vendors have leveraged shrinking process features to dramatically increase transistor density, while also manufacturing ever larger dies [1], [2], [3], [4]. However, this approach is coming to an end due to the slowdown in transistor scaling and the optical limitations in lithography that restrict the area growth of individual dies [5]. These issues are leading both GPU vendors and GPU researchers to identify alternative approaches to improve GPU performance scalability, most often employing multi-module GPU designs using a variety of integration domains [5], [6], [7], [8], [9].

Figure 1 shows the expected evolution of GPU architectures from a single monolithic die into multi-module GPU designs. Multi-module GPUs will be comprised of multiple

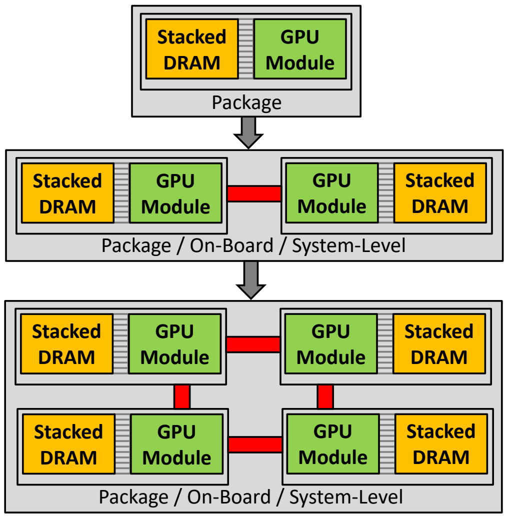


Figure 1: Scaling of future GPU designs via recent multi-module GPU proposals.

GPU modules (GPMs), each with its own local memory, interconnected either on-package [5], [9], on-board [6], [7], [8], at the system level, or a combination of the above. In order to deliver seamless performance scaling to GPU users, the design goal for these systems must be that despite its modular nature, the individual components must act in unison to provide the illusion of a single unified GPU.

Prior work has explored the feasibility of multi-module GPU scaling from a performance standpoint in both on-package and off-package setups [5], [8], [10], [11], [12], [13], [14]. Despite the growing energy constraints present in today's datacenters, prior work has failed to address performance scalability while also considering energy efficiency as a first class objective. While several groups have focused on GPU energy efficiency [15], [16], [17], [18], [19], these prior works have been largely focusing on the microarchitectural improvements within monolithic GPUs, rather than questions of efficient scaling in future multi-module GPU platforms.

Multi-module GPUs usher in a new era of design considerations where replication of individual GPMs is easy, but supporting the aggregated design with sufficient interconnect bandwidth and memory system localization is hard. Additionally, design considerations that are appropriate for small numbers of GPMs (or a monolithic GPU) may no longer be optimal for large numbers of GPMs due to differences in the physical integration domain. To understand and reason about these design decisions in scalable GPU systems, we propose a new metric called *Energy-Delay-Product Scaling Efficiency* (EDPSE) that takes into account both strong scaling performance and energy efficiency. This allows us to compare two GPU designs that may have dramatically different hardware resources and understand if the design is living up to its performance and energy efficiency potential. Similar to the commonly used parallel efficiency metric [20], we believe that future multi-module GPU designs will have to exceed specific EDPSE thresholds (e.g. 50% or better) to justify their production.

To understand multi-module GPU EDPSE trends we develop a new, top-down, instruction-based GPU energy estimation framework, called *GPUJoule*. GPUJoule is not only accurate but is also largely decoupled from GPU microarchitectural nuances. It is designed based on the following insight – the overall GPU energy consumption is the total energy cost of different types of instructions executed on a GPU (or GPM), plus the movement of data within the system to support these instructions. By focusing on instruction execution and macro-level data movement, GPUJoule is appropriate for system level scaling studies where different multi-module GPU designs may implement, for example, different caching organizations where the specific detail of the cache design is less important.

Utilizing GPUJoule in conjunction with an industry cycle-level GPU performance simulator allows us to explore EDPSE trends in multi-module GPU designs as GPUs grow from monolithic designs to as many as 32 GPMs in the future. Through this analysis we draw several unique insights and conclusions about the future of GPU strong scaling efficiency that will change the way GPU architects think about architecting modular designs. This paper builds upon prior work by making the following contributions:

- To the best of our knowledge, this is the first work to explore the energy efficiency of package and board level integrated GPUs. To enable this analysis, we propose a new metric called EDP Scaling Efficiency, that captures both the performance and energy cost for scaling GPU hardware resources. EDP Scaling Efficiency allows us to understand and quantify the scaling efficiency with respect to resources for future GPUs.
- We develop and validate GPUJoule, an agile instruction-based energy estimation framework for GPUs, appropriate for system scale GPU energy projections. GPUJoule is accurate within 10% of real silicon when empirically evaluated against a suite of HPC applications. GPUJoule is publicly available at https://github.com/akhilarunkumar/GPUJoule_release.
- Using these tools, we demonstrate that previously proposed multi-module GPUs, which considered performance alone, are likely to decrease its overall energy efficiency by nearly 2-fold, making them unsuitable for energy constrained environments.
- We demonstrate that neither the efficiency of GPM microarchitecture nor the energy cost of the inter-module interconnects will play a significant role in overall multi-module GPU energy efficiency. Our results show that inter-GPM bandwidth, largely irrespective of its energy per bit, will be the primary factor in determining the energy efficiency of these GPUs. Thus, to maximize future GPU efficiency, the research community should focus on system level data locality optimization and improved interconnect technologies rather than GPU microarchitectural optimization.

II. Background and Motivation

Scaling GPU performance through the integration of multiple GPMs along with their local and remote memories, results in a NUMA GPU architecture as shown in Figure 1. Such GPUs are expected to have unique characteristics affecting their performance and energy efficiency. For example, work partitioning and data locality nuances will dictate the inter-module communication needs. Additionally, the available inter-module bandwidth may vary significantly based on the integration domain and signaling technology employed. While state-of-the-art on-board interconnects can provide 300 GB/s of interconnect bandwidth per GPU [21], [22], this is still 3$\times$ lower than the 900 GB/s of DRAM bandwidth available to GPUs today. Future on-package integration technologies utilizing high density interconnect along with high speed signaling [5], [9] are expected to provide substantially higher bandwidth that may match or exceed today's DRAM bandwidth. Undoubtedly, DRAM bandwidth will continue to grow as well.

Depending on the integration domain, inter-module communication costs may vary substantially. On-package communication has an energy overhead of 0.54 pJ/bit [23] which is an order of magnitude lower than the energy overhead of on-board integration (10 pJ/bit) [5]. While it is advantageous to integrate in the most efficient domain possible, on-package integration is not expected to scale to a very large number of modules due to package size limitations. On-board integration can also leverage efficient interconnect topologies such as high-radix switch chips [22]. On-package environments will likely utilize multi-hop technologies without having dedicated switches, as these are more suitable for their planar nature and limited resources [5].

Although it is evident that multi-module GPUs can provide significant strong scaling performance improvements, they are likely to come with high power/energy overheads compared to traditional monolithic GPU designs. While power consumption continues to be an important point of consideration, the energy costs of computation are also becoming crucial given the growing electricity costs and

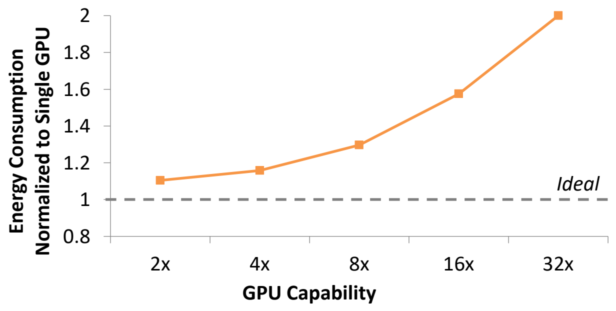


Figure 2: The average energy cost of strong scaling, when growing the number of GPMs using on-board integration.

carbon footprints of today's massive data centers. As a result, when designing and evaluating multi-module GPU architectures it is important to take into consideration not only performance or power in isolation, but also analyze the overall energy efficiency. For instance, prior work has shown that it may be possible to build 4, 8, or even 16× larger GPUs than those existing today by integrating them either on-package or on a single PCB. However, using the tools developed in this paper, Figure 2 shows that there is a significant increase in the energy required to compute a solution (averaged across 14 workloads), with a GPU built out of multiple GPMs. While this hypothetical 32× larger GPU may be able to ideally compute the solution to a problem 32× faster (if it could achieve ideal strong scaling), it is going to consume 2× more energy, on average. Thus, for a fixed problem size, this hypothetical multi-module-GPU is only half as energy efficient as the baseline design.

For many GPU users, this energy differential may not be of consequence as they might solely be focused on time to solution or their private datacenter might have additional energy headroom to spare. However, professional datacenters and cloud service providers often operate at near peak energy thresholds in order to achieve cost efficiency. Therefore, upgrading components to less energy efficient versions (despite them being higher performance) will likely lead to issues in power delivery, cooling, and provisioning. This trend towards energy *inefficiency* is what motivates us to quantify and analyze the energy characteristics of future strong scaling. By doing so, we can then understand what the main architectural bottlenecks are that affect such energy-performance trade-offs and thus, how to optimize them.

To be able to effectively model energy efficiency, accurate performance and energy estimation models and simulators are needed. Commonly used GPU energy estimation frameworks adopt a bottom-up modeling approach [15], [24], where the energy cost of each key microarchitectural component is combined with the corresponding switching activities to determine the overall GPU energy consumption. While this potentially results in very accurate analysis, it also comes with some significant drawbacks. First, since most of the microarchitectural details of modern GPUs are confidential, researchers are forced to guess and assume particular organizations. This leads to a lack of accuracy in energy projections, and requires adoption of additional statistical (or hand-tuned) error correction methods (fudge factors), that in turn reduce the overall flexibility of the model [25]. In addition, a relatively complex retraining process needs to be carried out for every new GPU generation in order to achieve acceptable levels of fidelity. For example, our analysis showed that adopting a commonly used bottom-up energy model that was tuned for NVIDIA's Fermi architecture without retuning it to NVIDIA's Kepler generation architecture, led to an average error of over 100% in energy estimations of an NVIDIA Kepler GPU.

Such retuning process requires maintaining a set of complicated microbenchmarks that are designed to reverse engineer and isolate various microarchitectural details of any given design. With the rate of GPU microarchitectural innovation, reverse engineering the details of every new GPU generation is a very challenging task and models that rely on this process unsurprisingly lag behind hardware and become irrelevant. Therefore, to understand and project GPU energy into the future, GPU researchers need a different modeling approach that is more sustainable. Prior work by Shao et al. [26] has shown that top-down models can in fact be both accurate and flexible in the CPU domain and we believe that a similar approach should be able to provide these qualities for GPUs as well. We address this need by developing a novel top-down energy model for GPUs in Section IV.

III. EDPSE: Quantifying GPU Energy Efficiency at Scale

Comparing two individual hardware designs can be done using metrics like performance, power or energy (e.g. design A achieves performance X, while design B achieves performance Y). For systems with a mostly fixed set of hardware resources, this generally lets us focus on performance as the figure of merit. For precise energy efficiency analyses there are even metrics like energy delay product (EDP) or ED^2, that combine energy and performance to allow comparisons between two designs on equal footing.

While being useful in comparing designs having similar resources, metrics such as EDP are not suitable for exploring the efficiency of different design points when we scale the GPU nodes from 1 to N. For example, let us assume that doubling the hardware resources while scaling from Design A to Design B provided an EDP that is 0.7x of Design A. Although the EDP of the Design B is lower, we are unable to infer if this was a good hardware investment. For instance, in the ideal case, we should have achieved an EDP that is 0.5x of Design A. This gap is mainly because the EDP metric does not take into account the amount of resources between two design points.

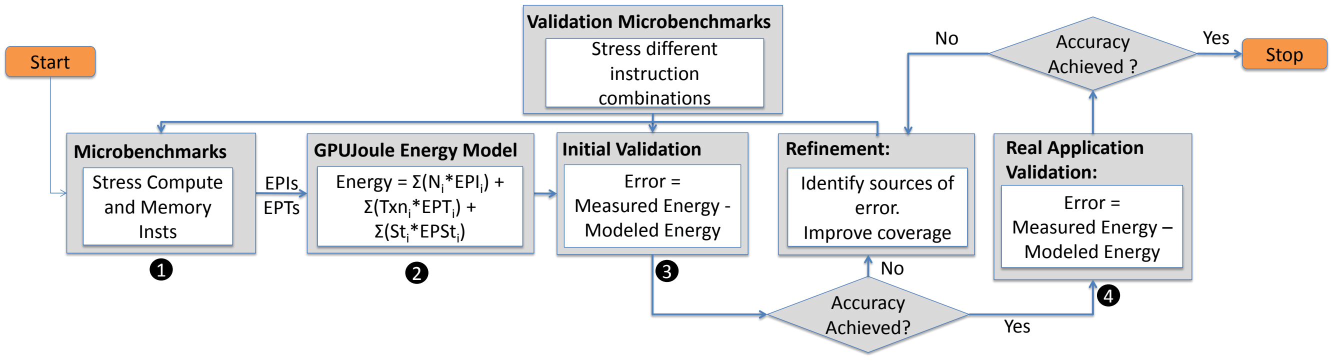

Start
Validation Microbenchmarks
Stress different instruction combinations
No
Accuracy Achieved ?
Yes
Stop
Microbenchmarks
Stress Compute and Memory Insts
EPIs
EPTs
GPUJoule Energy Model
Energy = Σ(N_i*EPI_i) + Σ(Txn_i*EPT_i) + Σ(St_i*EPSt_i)
Initial Validation
Error = Measured Energy - Modeled Energy
Refinement:
Identify sources of error. Improve coverage
Real Application Validation:
Error = Measured Energy – Modeled Energy
1
2
3
No
Accuracy Achieved?
Yes
4

Figure 3: GPUJoule top-down instruction-based energy modeling methodology

One prior approach that attempts to capture this effect is Parallel Efficiency, a commonly used metric to quantify the efficiency of strong performance scaling on parallel systems [20]. Parallel efficiency is a measure of the fraction of ideal speedup that is realized with scaling. If the execution time with 1 processor is t_1 and the execution time with N processors is t_N, then:

$$Parallel\ Efficiency = \frac{t_1 \times 100}{N \times t_N} \tag{1}$$

However, we can see from equation 1, parallel efficiency does not take into account the energy cost of scaling. Therefore, in order to make comparative decisions about energy efficiency scaling across a variety of design points, a metric that considers performance, energy, and the number of replicated resources is needed. To address this need, we propose *EDP Scaling Efficiency* (EDPSE).

EDPSE is defined as:

$$EDP\ Scaling\ Efficiency = \frac{EDP_1 \times 100}{N \times EDP_N} \tag{2}$$

EDP_1 is the EDP of one processor, N is the number of processors in the scaled configuration and EDP_N is the EDP of N processors. Extending the rationale of parallel efficiency (Equation 1), EDPSE is designed to measure the fraction of linear EDP scaling that is realized in a particular design. In a design that achieves linear performance speedup, strong scaling with N processors would reduce execution time (or delay) by a factor of N, while keeping the energy consumption constant; this would lead to an EDPSE of 100%. If performance achieves sub-linear scaling or energy consumption increases as the GPU design grows, both factors will reduce the design's EDPSE[1].

EDPSE can also be extended to provide different relative weights to performance and energy factors, reflecting different design priorities for a given architecture. In general, if an energy delay combination that takes the i^{th} power of delay i.e. ED^{i}P is deemed to be the metric of interest, then ED^{i}PSE can be defined as:

$$ED^iPSE = \frac{ED^iP_1 \times 100}{N^i \times ED^iP_N} \tag{3}$$

where ED^iP_1 and ED^iP_N are the ED^{i}P metric with one and N processors, respectively.

For GPU architects, evaluating designs based on EDPSE provides a relatively simple metric to understand the scaling efficiency achieved by future designs. Simply put, the higher the EDPSE, the better the design. Considering the growing significance of energy efficiency in today's datacenters and personal computers, we believe that GPU architects of the future will have to satisfy EDPSE design thresholds (e.g. 50%) to justify hardware improvements.

IV. GPUJoule: A GPU Energy Estimation Framework

Having defined an appropriate metric to reason about the scaling efficiency of multi-module GPUs, we now present our instruction based energy estimation framework for GPUs, called *GPUJoule*, that feeds into our EDPSE analyses. GPUJoule is based on publicly available information about GPU architectures to provide an accurate and reproducible estimation of GPU energy consumption. Its design leverages the insight that the total energy consumption of a GPU is the sum of the energy consumption of each instruction executed on the GPU, plus any constant overheads within the GPU.

For example, say a GPU is capable of executing compute instructions of type $1, 2, ...N_c$ and generates memory transactions of type $1, 2, ...N_m$ across the various levels of the memory hierarchy. To predict the total energy consumption of the GPU, we can thus estimate the Energy-Per-Instruction (EPI) and Energy-Per-Transaction (EPT) for both GPU pipeline instructions and memory system operations, while knowing very little about the specific microarchitectural implementation of the GPU itself. Utilizing the EPI and EPT information, the energy consumption of the GPU is predicted

[1]It is possible to achieve an EDPSE that is greater than 100% in cases of super-linear speedup or a decrease in energy.

as follows:

$$\begin{aligned} E_{GPU} = & \sum_{c=1}^{c=N_c} (EPI_c \times IC_c) + \\ & \sum_{m=1}^{d=N_m} (EPT_m \times TC_m) + \\ & (EPStall \times stalls) + \\ & (Const\_Power \times Execution\_Time) \end{aligned} \quad (4)$$

EPI_c and IC_c represent the energy-per-instruction and instruction count of *compute* instruction of type *'c'*, respectively. Similarly, EPT_m and TC_m represent the energy-per-transaction and transaction count of *memory* transaction of type *'m'*. $EPStall$ and $stalls$ represent the energy-per-stall of a compute lane stall and number of lane stalls, respectively which together account for the varying degree of parallelism in the GPU applications. $Const\_Power$ is the baseline idle power of the GPU which accounts for the power consumed by voltage regulators, the power delivery network, I/O to the host, and the static power of the GPU.

Figure 3 shows the modeling framework of GPUJoule. Before diving into the specifics of each step we provide a high level overview of the broad GPUJoule workflow. At a high level, GPUJoule uses an exhaustive set of microbenchmarks to stress the execution of different GPU instructions in isolation ❶, while also utilizing the GPU's parallelism to average the behavior of each instruction across thousands of iterations and all compute units (also known as SMs) in the system. Similarly, the memory microbenchmarks are carefully designed to isolate accesses and cause data movement between different levels of the GPU memory hierarchy. These microbenchmarks allow us to derive EPI_c and EPT_m values for individual native ISA (PTX) instructions. The EPI and EPT estimates then combine with the compute instruction and memory transaction event counts to constitute the initial GPUJoule energy model ❷. GPUJoule then iteratively improves upon the initial microbenchmark suite using two validation steps. First we design a set of synthetic microbenchmarks that combine different instruction types and correlate the modeled and measured energy values from real hardware to expose any coverage or instruction interaction issues overlooked in the initial microbenchmark design process ❸. Finally, we validate the energy model by correlating the modeled and measured energy for real GPU applications ❹. Overall, GPUJoule's estimation and validation process is fairly comparable to prior works that focus on instruction-based energy estimation in the server [27], mobile [28], and many-core architecture [26] domains but adjusted for the unique properties of GPU architectures.

A. Micro-Benchmark Construction

Modern GPUs support many native general purpose computation and data movement instructions such as ADD, MUL, SQRT, AND [29]. Depending on the instruction and data type, each instruction will utilize a variety of functional units. To extract the energy expenditure across all instructions, we develop two classes of microbenchmarks; compute microbenchmarks that work at the PTX ISA level, and data movement microbenchmarks that move data between different levels of the GPU memory hierarchy.

Algorithm 1 Compute instruction microbenchmark example - FMA instruction

```
1: procedure FMA-KERNEL($res, N$)      ▷ Location for result and number of iterations
2:                                     ▷ Declare and initialize registers
          $\_\_asm\ volatile($
                    $“\ .reg\ .f32\ \%r1;”$
                    $“\ mov.f32\ \%r1, k1;”$
                    $...);$
3:     for $i = 0$ To $i < num\_iterations$ do      ▷ Benchmark ROI operation
       $\_\_asm\ volatile($
                 $“fma.rn.f32\ \%r3,\ \%r1,\ \%r3,\ \%r2;”$
                 $...);$
4:     end for
5: end procedure
```

The compute microbenchmarks are designed to execute a particular instruction repeatedly so that the steady state power and energy consumption of the instruction execution can be determined. When designing these benchmarks we intentionally disable compiler optimizations and use inlined assembly to ensure that our benchmarks are faithfully executed on the hardware. Algorithm 1 shows an example microbenchmark designed to stress an FMA instruction. In each benchmark, we first declare and initialize the registers (line 2) used by the microbenchmark before the region-of-interest (ROI) of the microbenchmark is reached. In the ROI of the microbenchmark (lines 3 and 4) the instruction of interest is executed on the GPU iteratively and the corresponding power and energy consumption are measured using NVIDIA provided power measurement tools. Microbenchmarks similar to Algorithm 1 have been developed for all compute instructions in the PTX ISA.

To stress the data movement operations that fetch data from different levels of the GPU memory hierarchy such as shared memory, L1 cache, L2 cache, and the DRAM, a different type of benchmark is needed. These microbenchmarks are designed to first initialize a data set that completely fits within the particular level of the memory hierarchy (for which specifications are typically publicly available), and then consistently access the data set within that level.

NVIDIA Tesla K40	
Architecture	Kepler
Process node	28nm
SM count	15
Shared memory/L1 cache	16KB/48KB, 32KB/32KB, 48KB/16KB
L2 cache	1.5MB
DRAM	12GB, 280 GB/s

(a) Important specifications of the GPU.

	EPI/EPT [nJ]	Energy per bit [pJ/bit]
PTX Instructions		
32b float ADD, MUL, FMA	0.06, 0.05, 0.05	N/A
32b int ADD, SUB	0.07, 0.07	N/A
32b bitwise AND, OR, XOR	0.06, 0.06, 0.06	N/A
32b float SINE, COS	0.10, 0.10	N/A
32b int MUL, MAD	0.13, 0.15	N/A
64b float ADD, MUL, FMA	0.15, 0.13, 0.16	N/A
32b float SQRT, LOG2, EXP2, RCP	0.02, 0.03, 0.08, 0.31	N/A
Data Movement Transactions		
Shared Memory to Register File	5.45	5.32
L1 Cache to Register File	5.99	5.85
L2 Cache to L1 Cache	3.96	15.48
DRAM to L2 Cache	7.82	30.55

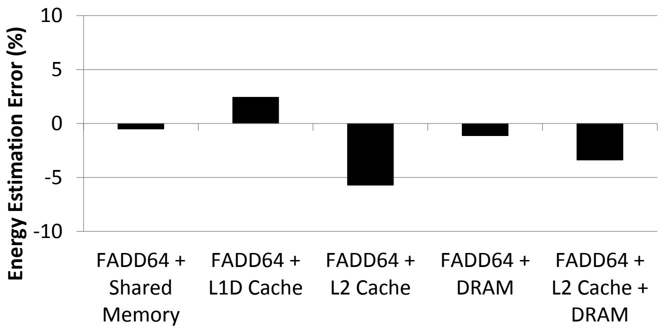
(b) Energy of operations as measured on real HW.

Table I: The NVIDIA Tesla K40 experimental GPU.

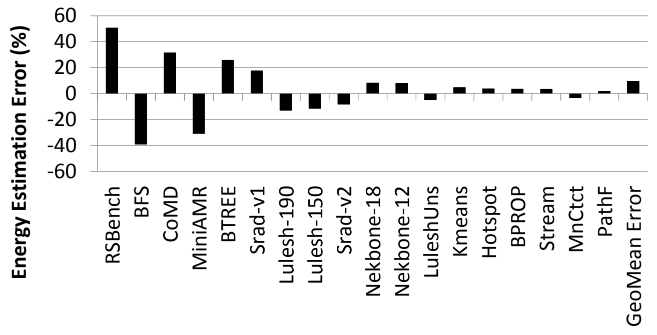
In order to isolate the data movement operations, we employ pointer-chasing microbenchmarks similar to prior works [27], [28]. To account for the nuances of the GPU execution model, we ensure that memory accesses originating from a warp can be coalesced to a single cacheline. We also ensure accesses stress the right level of the memory hierarchy by managing warp-level and thread-block level data locality as needed. For example, when stressing the L2 cache, we ensure there are no hits due to inter-warp or intra-warp locality in the L1 caches. By combining both SM pipeline stalls and data transfer energies, our model takes into consideration the energy impact of memory divergence. Our model doesn't take into account the effect of control divergence though, due to unavailability of performance counters that reflect partial SM pipeline utilization.

B. Generating Energy Estimates

In this work we choose to validate GPUJoule using a NVIDIA Tesla K40 GPU [30] for which specifications are provided in Table Ia. We use the NVIDIA management library (NVML) [31] to query the on-board power sensor and look at the steady state power consumption incurred by different microbenchmarks to discern the EPI and EPT values for this GPU. The EPI for an instruction is computed



(a) Energy estimation errors for microbenchmarks using combination of different types of instructions.



(b) Energy estimation errors for real GPU Applications.

Figure 4: GPUJoule energy estimation accuracy. Comparison of energy estimations with hardware measurements on the Tesla K40 GPU.

as following:

$$EPI = \frac{(Power_{active} - Power_{idle}) \times Exec.\ Time}{Num.\ of\ Instructions} \quad (5)$$

The EPT for data transfer operations is similarly determined.

1) Energy Per Instruction Estimates: Table Ib shows the EPI values for GPU compute instructions and the EPT values for GPU memory system transactions. We observe that the energy consumption of GPU pipeline instructions have some variability depending on the instruction type and that both data type and width, perhaps unsurprisingly, affect the energy expended per operation. We can see that the energy cost of moving data from the shared memory or L1 cache to the registers is much lower than among other component of the memory hierarchy. Also, per bit energy expenditure increases as the data is brought from farther levels of the memory hierarchy. Moving data from the DRAM to the register file costs nearly 10$\times$ the energy of moving data from L1 cache/shared memory to the register file, and delivering data to a floating point unit from the DRAM expends 80$\times$ the energy of performing the computation on that data.

2) GPUJoule Validation to Silicon: To validate the energy estimates against silicon, we leverage the validation microbenchmarks shown in Figure 3. Furthermore, we use an additional set of 18 GPU applications from Rodinia [32] and the CORAL benchmark suite [33], summarized later in Table II. Figure 4a shows the accuracy observed for the different microbenchmarks with mixed instruction types. We see that the energy estimation error is within +2.5% and -6% indicating GPUJoule has good fidelity compared to real hardware on a per instruction basis. Figure 4b shows the end-to-end relative error in energy estimation for our entire workload suite. While the correlation is very good for most benchmarks, with a 9.4% mean absolute error across all benchmarks, we see that four applications experience an absolute error >30%. For applications such as RSBench and CoMD, we find the utilization of the memory subsystem is very low and the GPUJoule energy model may underestimate memory energy consumption. In BFS and MiniAMR applications the prediction inaccuracies are largely due to the limitations of the on-board power sensor. These applications have many short duration kernel invocations (100s of microseconds long), and the resolution of the on-board power sensor (15ms refresh period [34]) is not fine enough to capture the kernel power consumption with high precision. While we could have introduced fudge factors to try and improve the correlation of all outliers, that would reduce the generality of our model and its appropriateness for GPU scalability studies.

3) Applicability to Other GPUs: The GPUJoule methodology has been designed to be easily applicable to any current or future GPUs. By utilizing simple microbenchmarks created for the publicly available set of PTX instructions and data movement instructions, GPUJoule's energy model can be easily regenerated for any GPU of interest. Furthermore, as we leverage the higher level PTX (and not SASS) instructions' EPIs and EPTs as basic building blocks, the GPUJoule methodology has specifically been designed to be microarchitecture agnostic without loss of fidelity. Although the specific EPI and EPT values might change depending on the advances in the process nodes or microarchitectural features, the overall accuracy of the GPUJoule energy model can be expected to be similar to what we have demonstrated here. Due to its generality, GPUJoule can become a handy tool for various GPU architecture and system architecture explorations. Therefore, we have open-sourced GPUJoule and made it available for the broader research community.

V. Energy Efficiency and the Future of Multi-Module GPUs

In this section we lay out the final details of our methodology and simulated multi-GPM architecture. Then using GPUJoule and EDPSE, we perform a detailed analysis of the energy efficiency and performance scalability of several multi-module GPU configurations. We identify the key limitations affecting both energy and performance scaling and explore the solution space on the road to efficient scaling using multi-module GPUs.

Benchmark	Input	Abbr.	Cat.
Back Propagation [32]	65536	BPROP	C
B+Tree [32]	1 Million	BTREE	C
Classic Molecular Dynamics [33]	49 bodies	CoMD	C
Hotspot [32]	1024x1024	Hotspot	C
Lulesh [33]	Unstrc Mesh	LuleshUns	C
Path Finder [32]	1 Million	PathF	C
RSBench [33]	1 Million	RSBench	C
SRAD (v1) [32]	100, 0.5, 502, 458	Srad-v1	C
Adaptive Mesh Refinement [33]	15,000	MiniAMR	M
Breadth First Search [32]	Graph1MW	BFS	M
Kmeans clustering [32]	819200	Kmeans	M
Lulesh [33]	size 150	Lulesh-150	M
Lulesh [33]	size 190	Lulesh-190	M
Nekbone solver [33]	size 12	Nekbone-12	M
Nekbone solver [33]	size 18	Nekbone-18	M
Mini Contact [33]	Mas1_2	MnCtct	M
SRAD (v2) [32]	2048x2048	Srad-v2	M
Stream Triad [35]	2^{26} elements	Stream	M

Table II: GPU applications and their inputs. C: Compute intensive, and M: Memory bandwidth intensive benchmarks.

A. Experimental Methodology

In this work, we integrate GPUJoule energy model with an NVIDIA proprietary, cycle-level, trace driven GPU performance simulator. The simulator faithfully models key GPU features such as warp and thread-block scheduling, warp-level parallelism and latency tolerance, the presence of a multi-level memory hierarchy, software based coherence of private caches, and memory and interconnect bandwidths. We utilize a subset of 14 workloads (all, except BFS, LuleshUns, MnCtct, and Srad-v1) selected from Table II that have enough inherent parallelism to fill a GPU with 32× the capability of our basic GPU module building block.

1) Simulated Architecture: Our simulations configure a basic GPM to be similar to an NVIDIA Tesla K40 GPU. The simulated basic GPU module comprises 16 SMs equipped with a 32 KB L1 cache each, a shared L2 cache of 2 MB, and one HBM [36] memory stack with the DRAM bandwidth of 256 GB/s (see the 1-GPM configuration in Table III and Figure 5(a)). The GPMs are expected to be individually smaller than the largest GPUs available today [37] because smaller die sizes yield significant cost and yield advantages for manufacturers [9], [38]. Our GPUJoule energy model has been purposely trained on the Tesla K40 GPU to increase the confidence of our projections and we note that because the compute to memory bandwidth ratio has remained relatively similar across GPU generations [2],

Configuration	1-GPM	2-GPM	4-GPM	8-GPM	16-GPM	32-GPM
Number of Modules	1	2	4	8	16	32
Total SM count	16	32	64	128	256	512
L1 cache per SM	32 KB	32 KB	32 KB	32 KB	32 KB	32 KB
Total L2 cache	2 MB	4 MB	8 MB	16 MB	32 MB	64 MB
Total DRAM Bandwidth	256 GB/s	512 GB/s	1024 GB/s	2048 GB/s	4096 GB/s	8192 GB/s

Table III: Simulated multi-module GPU configurations.

Configuration	Inter-GPM BW	Inter-GPM to DRAM BW	Integration Domain
1x-BW	128 GB/s	1:2	on-board
2x-BW	256 GB/s	1:1	on-package
4x-BW	512:GB/s	2:1	on-package

Table IV: Simulated per-GPM I/O bandwidth.

[4], [37]; the conclusions drawn with our configurations should be applicable to multi-module GPUs with larger and more capable GPMs, as long as the baseline ratios between compute throughput, DRAM bandwidth, and I/O bandwidth do not change dramatically for a given GPM.

As shown in Table III, we scale the number of GPU modules in the system from 1–32. For the multi module configurations, we employ distributed thread-block scheduling and first touch page placement to capture data locality within the GPMs, as proposed in prior works [5], [8]. In-line with these prior multi-module works, in 2-GPM configurations and beyond we move the L2 cache from being a memory-side cache to become a module-side cache, and the GPMs are interconnected in a ring topology. We evaluate three different per-GPM I/O bandwidth settings. The inter-GPM bandwidths are configured in relation to the available local GPM DRAM bandwidth (see Figure 5(b)). The different bandwidth settings are as shown in Table IV, representing inter-GPM bandwidth that is 2× lower, equal, and 2× larger than local GPM DRAM bandwidth. For example, current NVIDIA Volta GPUs support a on-board inter-GPU to DRAM bandwidth ratio of 1:3. In this analysis we assume a slightly improved I/O to DRAM bandwidth ratio of 1:2 (called 1x-BW), to reflect future on-board integration capabilities. Similarly, a bandwidth ratio of 1:1 (called 2x-BW) reflects current projections for on-package I/O bandwidth used by Arunkumar et al. [5] and the 2:1 ratio (called 4x-BW) corresponds to higher on-package BW settings that may become available through use of next generation signaling technologies [23].

2) *Energy Model Considerations:* Given that each GPM in our multi-module designs is similar to the Telsa K40 GPU, we can expect the power consumption of the multi-module GPUs to be a linear multiple of the power of the K40 GPU (before considering any power reductions brought by newer process nodes or power management techniques such as DVFS, power/clock gating etc.) Thus it follows that the vast majority of the EPI and EPT parameters can be used directly from GPUJoule, as described in Section IV. To

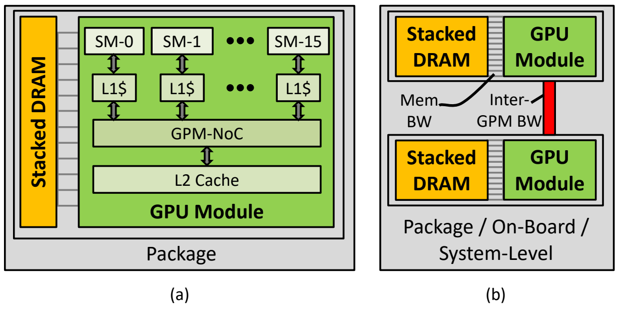


Figure 5: (a) Architecture details of the simulated 1-GPM GPU and the basic GPU Module. (b) A 2-GPM GPU configuration, inter-GPM BW is configured relatove to local memory bandwidth (4-32 GPMs points are configured similarly.

accurately reflect future architectures we augment the energy model to reflect the use of HBM (vs GDDR5 memory) and the inclusion of signaling overheads for on and off package links. Specifically we use the published energy costs of GDDR5 and HBM technologies [39] to approximate the DRAM to L2 cache energy cost of 21.1 pJ/bit for the GPU system with HBM and we use the published [23] energy costs of on-package integration as 0.54 pJ/bit, and estimate 10 pJ/bit for on-board links though we later discuss the implications of this link potentially having up to 4× this energy overhead.

On-board integration of discrete GPM components comes with many per-GPM static energy overheads including voltage regulators, fans, system I/O, etc. To model on-board integration, we scale the static energy component linearly with the number of GPMs. As we will show later in this section, we can assume that certain on-platform components can be shared across multiple GPMs on package however. Thus we scale only part of the measured static energy per GPM with the number of GPMs, we term this *Constant Energy Amortization*. To account for constant energy amortization under on-package integration, we assume that only 50% of the original per-GPM constant energy grows linearly with the number of GPMs. We also study the sensitivity to this parameter later in Section V-C.

B. *Understanding Energy Efficiency*

Figure 6 shows the EDPSE of future GPU designs as they scale the number of GPMs, using an on-package integration domain. We observe that compute intensive workloads

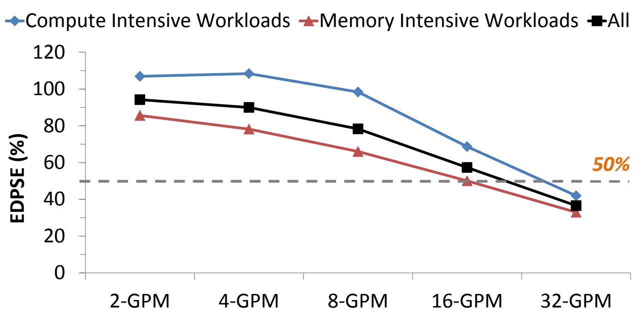


Figure 6: EDPSE of compute intensive, memory intensive, and all workloads, for baseline on-package configuration (2x-BW).

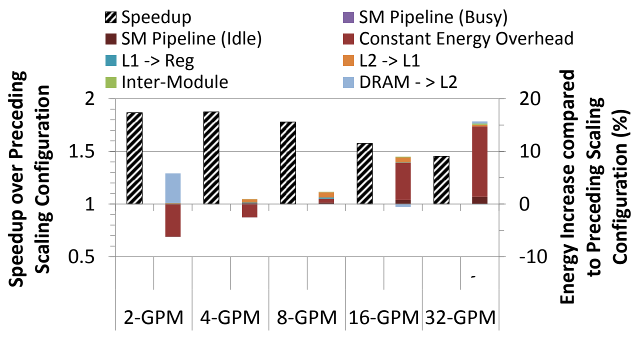


Figure 7: Performance speedup (left axis) and energy increase (right axis), compared to each preceding multi-GPM configuration. Energy consumption is broken down by component.

achieve significantly higher EDPSE than their memory intensive counterparts. In fact, compute intensive workloads achieve an EDPSE that is higher than 100% for small GPM counts, as these workloads are able to leverage the increased caching resources and reduce their dependence on memory bandwidth, while incurring negligible or low energy costs.

A second trend is that EDPSE starts to decrease dramatically at high GPM counts. Maximum EDPSE achieved (when averaged across all workloads) is 94% in the 2-GPM configuration, and decreases to 36% with 32 GPMs. We find this is primarily due to the NUMA bandwidth limitations that are amplified when growing the number of GPMs in a ring topology. Extending the commonly used parallel efficiency threshold of 50% to EDPSE, we observe that on-package multi-module GPUs may start running into energy efficiency limitations when scaled beyond 16 GPMs.

To understand the decreasing EDPSE values we analyzed the speedup and energy consumption at each scaling step. Figure 7 shows the incremental speedup and energy increase *compared to the preceding point*, as the number of GPMs is scaled. We observe that the incremental speedup achieved at each scaling step decreases, sometimes dramatically. For example, scaling from the the 1-GPM to 2-GPM configuration achieves 86.8% speedup, whereas scaling from the 16-GPM to the 32-GPM achieves only 47% speedup. We note, our experiments on the same set of applications on similarly equipped hypothetical monolithic GPU achieves 80.8% speedup when scaling from the 16-GPM to 32-GPM point. We conclude that the observed performance penalty can primarily be attributed to the NUMA-related bottlenecks of multi-module GPUs.

Figure 7 also demonstrates the growing relative energy cost at each scaling step across a range of metrics available in the GPUJoule energy model. Scaling from the 16-GPM to 32-GPM configuration results in a 15.7% increase in energy consumption. Further analyzing the sources of energy growth at each step, we observe that when we first scale from the 1-GPM to 2-GPM configuration, the energy cost associated with new NUMA architecture becomes immediately visible in the form of increased DRAM to L2 cache energy consumption. However, due a slight super-linear performance speedup in some benchmarks and a reduction in constant energy overhead (as redundant components can be eliminated via on-package integration) this additional data movement energy overhead is mostly offset. As a result, the 2-GPM configuration experiences only a minor decrease in EDPSE overall.

Unfortunately, as the number of GPMs grows the energy consumption grows significantly. Some of this energy increase comes from the increase in L2 -> L1 data movement as L1 cache locality is lost by spreading the thread blocks over larger number of SMs. However the dominant source of energy increase happens to be the constant energy overhead. Analysis shows that this unfortunate effect is associated with an increasing number of GPM pipeline stalls (GPMs being idle and waiting on remote memory access) due to increased inter-module bandwidth pressure at large GPM counts. This, in turn, increases the relative contribution of static energy and also contributes to the sharp decline in EDPSE. We conclude that as module based GPU designs become commonplace, NUMA effects and specifically inter-GPM bandwidth will be the primary challenge in achieving future GPU energy efficiency.

C. Optimizing for Energy Efficiency

Interconnect Bandwidth

Thus far, we have evaluated multi-module GPUs using the on-package integration domain. However the integration domain alone does not determine the inter-GPM bandwidth that is available and architect's implementation choices, may also change the available bandwidth by 2×, either lower or higher. Figure 8 shows the impact of using different interconnect bandwidths across both on-package and on-board integration domains as described in Table IV. Inter-module bandwidth has pronounced effects on EDP scaling efficiency, and at high GPM counts, EDPSE improves by a factor of 3 when inter-module bandwidth increases by a

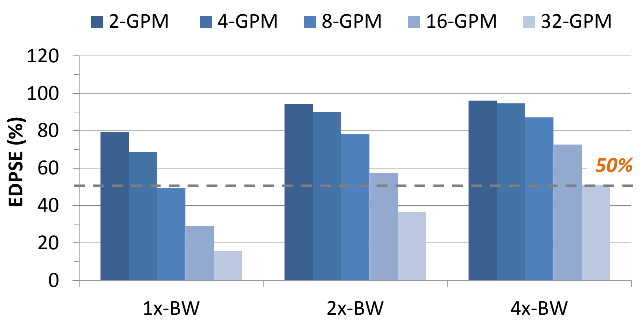


Figure 8: EDPSE as a function of interconnect bandwidth settings.

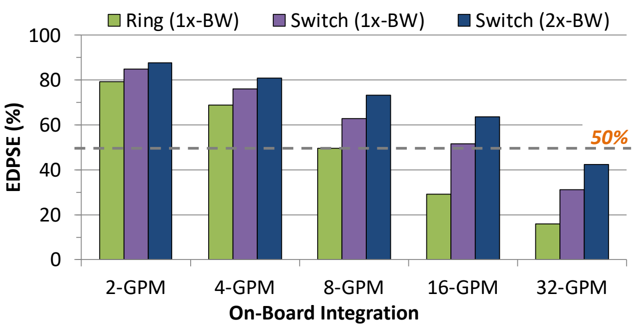


Figure 9: EDPSE for on-board ring and switched interconnection networks

factor of 4. This supports our conclusions that providing adequate levels of inter-module bandwidth is going to be the most important factor in maintaining high levels of energy efficiency in future multi-module GPUs.

Interconnect Energy

As the number of GPMs in future GPUs grow, the amount of data that is transferred on the inter-module links will increase. Common architectural practice places significant importance on reducing the intrinsic energy cost of interconnect technologies. Yet Figure 7 indicates that the overall GPU energy growth in high module-count systems that can be attributed to this increased data movement is relatively low (the "inter-module" stack in Figure 7). To delve deeper, we performed a point study on the GPU's overall energy sensitivity to increased inter-module interconnect energy consumption, while leaving bandwidth unchanged.

Using the 32-GPM design in an on-board integration domain (described as 1x-BW), we increased the per/bit interconnect energy cost by a factor of 2× and 4× over the baseline (10 pJ/bit). We observe that even with a 4× increase in the interconnect energy cost, the net impact on the EDPSE is below 1%. This is a significant result, when you consider that there is nearly a 2× improvement in EDPSE when doubling the inter-module bandwidth from the 1x-BW to 2x-BW configurations (Figure 8). We conclude that, counter to design trends in monolithic GPUs, multi-module GPU architects should not focus on driving down the intrinsic energy cost of interconnect technologies, but instead work towards maximizing bandwidth density, even at the expense of per bit transfer energy. The right architectural trade off may in fact be to prefer the highest bandwidth interconnect available, regardless of energy cost, because it leads to the highest possible global EDPSE. For example, our analysis shows that if the 4× higher interconnect energy cost could be used to achieve 2× higher interconnect bandwidth, it would cause 8.8% increase in EDPSE for a 32-GPM design.

Impact of Integration Domain

On-package integration comes not only with improved interconnect technologies that enable significant bandwidth advantages, but also the opportunities to reduce constant energy overheads. Tighter on-package integration of GPMs provides an opportunity to share the previously per-GPM energy burden of various on-platform components such as cooling, power delivery, etc. Because at high GPM counts, SM idle time, and thus constant energy overheads is a significant factor in overall energy efficiency, the effect of overhead amortization among on-package GPMs can be significant. We analyze a hypothetical 32-GPM system using on-package integration (with a 2x-BW configuration) where fixed energy overheads can not be amortized or amortized at a 50% rate, as described in Section V-A2. When compared to having no amortization, on-package integration achieves an impressive 22.3% decrease in absolute energy consumption and 8.1% increase in EDPSE. If the assumed amortization rate is reduced to 25% for example, then the energy saving would reach 10.4% on average compared to having no amortization at all, with a 3.5% increase in EDPSE.

Impact of an On-Board High-Radix Switch

To alleviate the impact of low inter-module bandwidth in on-board integration scenarios, GPU manufacturers have recently introduced high-bandwidth and high-radix switch chips for these systems [6], [7], [22]. Compared to ring topologies, switched networks reduce the number of hops between source and destination, reducing the inter-module bandwidth congestion, despite the inter-GPU link bandwidth values remaining unchanged. Figure 9 shows the EDPSE achieved for on-board multi-module GPUs in the presence of a switched network.[2] We see that the introduction of a switch over a basic ring topology can improve EDPSE by nearly 2× in the 32-GPM case, despite no change in the actual link bandwidth. This further underscores the importance of reducing NUMA-related inter-module bandwidth bottlenecks at all cost, not just via increased link bandwidth, but also via network topology innovations.

[2]We assume an additional 10 pJ/bit data movement energy cost through the switch.

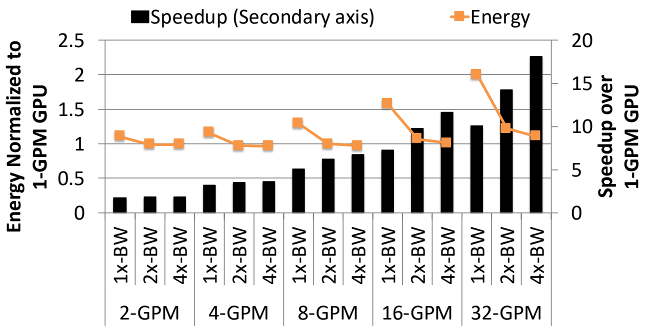


Figure 10: Speedup and energy consumption when varying interconnect bandwidth, and on-package integration at the 2x-BW and 4x-BW points.

D. *Decomposing EDSPE Improvements*

While we have studied the energy efficiency trends using EDPSE metric, it is important to note that similar trends — *diminishing energy efficiency with increasing number of modules* — will be apparent with other metrics that rely on ED^2 or performance/watt as well. It is also very important to be cognizant of the risks in focusing only on metrics that combine power, energy and performance, as each individual constraint can be overlooked by designers. Therefore, understanding the relative contributions of energy and performance into the combined metric is equally important. Figure 10 summarizes the absolute speedup and energy expenditure across all GPM counts, and at all three bandwidth configurations. This data takes into account the aforementioned optimization allowing amortization of constant energy across all GPMs when moving from on-board (1x-BW) to on-package domains (2x-BW and 4x-BW), but still assumes a ring topology in all integration domains. As we scale the number of GPMs, we can see that at a higher number of modules (GPM-8, 16, and 32), the achieved speedup is primarily associated with improvements in inter-GPM bandwidth. So significant is this effect, that a 16-GPM design with 2x-BW will outperform a 32-GPM design with only half the inter-module bandwidth, while expending just half the energy. In fact, a 32-GPM system configured with just 1x-BW (in an on-board domain) can reduce the energy required to compute a fixed size problem by doing nothing other than increasing the inter-GPM bandwidth by a factor of four, by 27.4% on average. Furthermore, if we subsequently move the integration to on-package domain and take into account the effect of constant energy amortization described earlier, this energy reduction increases to 45% on average.

We conclude that improving inter-GPM interconnect technologies and topologies, along with less traditional architecture measures, such as sharing the platform related constant energy overheads across multiple GPMs will play a crucial role in making strong scaling with multi-module GPUs worthwhile from an energy efficiency point of view. Our findings indicate, that GPU architects will have to start making new design trade offs between interconnect topologies, link bandwidth, link energy, integration domains and to create opportunities to holistically share the overall energy burden of the GPU most efficiently. At extreme scales, architects may be forced to turn to extreme measures such as reallocation of costly on-chip pin-outs to re-balance local DRAM bandwidth versus inter-GPM bandwidth if the ratio of local to remote memory access happens to skew towards the latter.

E. *Discussion*

This work illuminates multiple key energy efficiency trends in future multi-module GPUs. We identify that without architecture and system level innovations, multi-module GPUs will quickly run into energy efficiency concerns when trying to scale performance at all costs. In light of these observations, traditional approaches to energy efficiency improvement within the GPU modules themselves will only manifest as second order effects in the future. Our results highlight the pressing need for future research to focus on reducing the impact of the NUMA effects on multi-module GPUs. While prior works have attempted to address similar aspects in NUMA CPU systems [40], [41], [42], [43], techniques appropriate for multi-module GPUs deserve a close attention. Furthermore, techniques such as locality aware thread-block (CTA) scheduling and data placement [5], [44], sophisticated cache management strategies [8], [45], [46], [47], [48], [49], [50], and data compression techniques [51], [52], [53], [54], need to be re-applied not just within today's GPUs, but now among GPU modules. In addition, system-level techniques that reduce the impact of constant power in the presence of large number of GPU modules are going to be crucial. Techniques such as integration technology innovations, intelligent clock-gating and power-gating [55], can improve energy efficiency of multi-module GPUs.

VI. Related Work

GPU energy efficiency has been addressed in many prior works [15], [16], [17], [18], [19], [54], [48], [56], [57], [58], [59], [60], [61]. However these works were done in context of monolithic GPUs, and focused on minimizing energy consumption through microarchitectural innovations. More recently, Arunkumar et al. [5] and Milic et al. [8] found that inter-module bandwidth plays a key role in the performance scalability of multi-module GPUs but did not address energy efficiency in their proposals. Vijayaraghavan et al. [9], perform a first order characterization of performance, power, and temperature using a specific multi-module GPU within an exascale node architecture. However, they do not address the scalability issues of multi-module GPUs.

Inspiring the design of GPUJoule, Wu et al. [62] design a machine learning based power model that estimates the power consumption of future chips. However, such approaches do not provide insights into the energy consumption trends and specific bottlenecks. To achieve that most prior works take a bottom-up approach to power modeling; such as Leng et al. [15] that use a cycle-level architecture-level power model based on McPat [63]. Similarly, Guerreiro et al. [34] have also proposed a DVFS-aware power model for GPUs. Bottom up approaches have key advantages by offering cycle-level power estimations and an understanding of the power consumption of fine-grained microarchitectural structures. However, as described in Section II, these and other bottom-up power models [24], [64], [65], [66] are very difficult to maintain and keep current.

Most similar to GPUJoule, Kestor et al. [27], Pandiyan et al. [28] and Shao et al. [26] take top down approaches to characterize the energy consumption of server, mobile, and XeonPhi many-core processors respectively. Since these models were developed for CPU-like processors, they can not be easily re-applied to GPU, but reinforce the value of the GPUJoule approach.

VII. Conclusions

The future of GPU computing relies on scaling GPU performance, using modular designs in an energy efficient way. Our in-depth scalability analysis reveals three key findings. First, we demonstrate that prior multi-module GPUs, which only considered performance scalability are on track to incur a 2× energy penalty in order to achieve that goal. Second, our analysis reveals that the dominant factor leading to energy in-efficiency is lack of inter-GPM bandwidth. Insufficient inter-GPM bandwidth increases GPM idle time, hampering performance scalability while simultaneously exposing the relatively increasing overhead of constant energy components in the system. Finally, we demonstrate how an analysis-driven choice of interconnect technology, can provide counter-intuitive results, and encourage architects to make energy-inefficient choices locally (i.e. in the inter-GPM interconnect) to help maximize energy efficiency globally. Using this analysis, we show that it is possible to reduce future multi-module GPU energy consumption growth from over 100% to just 10% (compared to a single GPU) while improving strong scaling performance by a factor of 18×, paving the way for further architectural enhancements that will drive aggressive performance scaling, while not being at odds with GPU energy efficiency.

Acknowledgements

The authors would like to thank the anonymous reviewers for their insightful feedback, which has been used to improve the paper. The authors would also like to thank NVIDIA for the equipment donation. This work is supported in part by the National Science Foundation under CCF-1618039 and CCF-1525462.

References

[1] G. E. Moore, “Cramming More Components onto Integrated Circuits,” *Electronics*, vol. 38, no. 8, Apr 1965.

[2] “NVIDIA’s Next Generation CUDA Compute Architecture: Kepler GK110,” 2012. [Online]. Available: https://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf

[3] “NVIDIA Tesla P100 Architecture,” 2016. [Online]. Available: https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdff

[4] “NVIDIA Tesla V100 Architecture,” 2017. [Online]. Available: http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

[5] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans, “MCM-GPU: Multi-Chip-Module GPUs for Continued Performance Scalability,” in *Proc. of the Annual International Symposium on Computer Architecture*, 2017.

[6] “NVIDIA DGX-1,” 2018. [Online]. Available: https://www.nvidia.com/en-us/data-center/dgx-1/

[7] “NVIDIA HGX-2,” 2018. [Online]. Available: https://www.nvidia.com/en-us/data-center/hgx/

[8] U. Milic, O. Villa, E. Bolotin, A. Arunkumar, E. Ebrahimi, A. Jaleel, A. Ramirez, and D. Nellans, “Beyond the Socket: NUMA-aware GPUs,” in *Proc. of the Annual IEEE/ACM International Symposium on Microarchitecture*, 2017.

[9] T. Vijayaraghavan, Y. Eckert, G. H. Loh, M. J. Schulte, M. Ignatowski, B. M. Beckmann, W. C. Brantley, J. L. Greathouse, W. Huang, A. Karunanithi, O. Kayiran, M. Meswani, I. Paul, M. Poremba, S. Raasch, S. K. Reinhardt, G. Sadowski, and V. Sridharan, “Design and Analysis of an APU for Exascale Computing,” in *Proc. of the IEEE International Symposium on High Performance Computer Architecture*, 2017.

[10] T. Ben-Nun, E. Levy, A. Barak, and E. Rubin, “Memory Access Patterns: The Missing Piece of the Multi-GPU puzzle,” in *SC15: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2015.

[11] L. Chen, O. Villa, and G. R. Gao, “Exploring Fine-Grained Task-Based Execution on Multi-GPU Systems,” in *Proc. of the IEEE International Conference on Cluster Computing*, 2011.

[12] L. Chen, O. Villa, S. Krishnamoorthy, and G. R. Gao, “Dynamic Load Balancing on Single- and Multi-GPU Systems,” in *Proc. of the IEEE International Symposium on Parallel Distributed Processing*, 2010.

[13] T. Mitsuishi, J. Suzuki, Y. Hayashi, M. Kan, and H. Amano, “Breadth First Search on Cost-efficient Multi-GPU Systems,” *SIGARCH Comput. Archit. News*, vol. 43, no. 4, pp. 58–63, Apr. 2016.

[14] J. A. Stuart and J. D. Owens, “Multi-GPU MapReduce on GPU Clusters,” in *Proc. of the IEEE International Parallel Distributed Processing Symposium*, 2011.

[15] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, “GPUWattch: Enabling Energy Optimizations in GPGPUs,” in *Proc. of the Annual International Symposium on Computer Architecture*, 2013.

[16] A. Sethia and S. Mahlke, “Equalizer: Dynamic Tuning of GPU Resources for Efficient Execution,” in *Proc. of the Annual IEEE/ACM International Symposium on Microarchitecture*, 2014.

[17] O. Kayiran, A. Jog, A. Pattnaik, R. Ausavarungnirun, X. Tang, M. T. Kandemir, G. H. Loh, O. Mutlu, and C. R. Das, “uC-States: Fine-grained GPU Datapath Power Management,” in *Proc. of the International Conference on Parallel Architectures and Compilation*, 2016.

[18] M. H. Santriaji and H. Hoffmann, “GRAPE: Minimizing Energy for GPU Applications with Performance Requirements,” in *Proc. of the Annual IEEE/ACM International Symposium on Microarchitecture*, 2016.

[19] A. Sethia, G. Dasika, M. Samadi, and S. Mahlke, “APOGEE: Adaptive Prefetching on GPUs for Energy Efficiency,” in *Proc. of the International Conference on Parallel Architectures and Compilation Techniques*, 2013.

[20] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc., 1994.

[21] “NVIDIA NVLink High-Speed Interconnect,” 2016. [Online]. Available: http://www.nvidia.com/object/nvlink.html

[22] “NVIDIA NVSWITCH,” 2018. [Online]. Available: http://images.nvidia.com/content/pdf/nvswitch-technical-overview.pdf

[23] J. W. Poulton, W. J. Dally, X. Chen, J. G. Eyles, T. H. Greer, S. G. Tell, J. M. Wilson, and C. T. Gray, “A 0.54 pJ/b 20 Gb/s Ground-Referenced Single-Ended Short-Reach Serial Link in 28 nm CMOS for Advanced Packaging Applications,” *IEEE Journal of Solid-State Circuits*, vol. 48, no. 12, pp. 3206–3218, 2013.

[24] S. Hong and H. Kim, “An Integrated GPU Power and Performance Model,” in *Proc. of the Annual International Symposium on Computer Architecture*, 2010.

[25] T. Nowatzki, J. Menon, C. han Ho, and K. Sankaralingam, “gem5, GPGPUSim, McPAT, GPUWattch, "Your favorite simulator here" Considered Harmful,” 2014.

[26] Y. S. Shao and D. Brooks, “Energy Characterization and Instruction-level Energy Model of Intel’s Xeon Phi Processor,” in *Proc. of the International Symposium on Low Power Electronics and Design*, 2013.

[27] G. Kestor, R. Gioiosa, D. J. Kerbyson, and A. Hoisie, “Quantifying the Energy Cost of Data Movement in Scientific Applications,” in *Proc. of the IEEE International Symposium on Workload Characterization*, 2013.

[28] D. Pandiyan and C. J. Wu, “Quantifying the Energy Cost of Data Movement for Emerging Smart Phone Workloads on Mobile Platforms,” in *Proc. of the IEEE International Symposium on Workload Characterization*, 2014.

[29] “CUDA C Programming Guide,” 2017. [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[30] “NVIDIA Kepler GK110 Architecture,” 2012. [Online]. Available: https://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf

[31] “NVML API Reference Manual,” 2012. [Online]. Available: http://developer.download.nvidia.com/assets/cuda/files/CUDADownloads/NVML/nvml.pdf

[32] S. Che, J. W. Sheaffer, M. Boyer, L. G. Szafaryn, L. Wang, and K. Skadron, “A Characterization of the Rodinia Benchmark Suite with Comparison to Contemporary CMP Workloads,” in *Proc. of the International Symposium on Workload Characterization*, 2010.

[33] “Coral benchmarks,” 2014. [Online]. Available: https://asc.llnl.gov/CORAL-benchmarks/

[34] J. Guerreiro, A. Ilic, N. Roma, and P. Tomas, “GPGPU Power Modeling for Multi-domain Voltage-Frequency Scaling,” in *Proc. of the IEEE International Symposium on High Performance Computer Architecture*, 2018.

[35] J. D. McCalpin, “Memory Bandwidth and Machine Balance in Current High Performance Computers,” *IEEE Computer Society Technical Committee on Computer Architecture Newsletter*, pp. 19–25, Dec. 1995.

[36] J. Kim and Y. Kim, “HBM: Memory solution for bandwidth-hungry processors,” in *IEEE Hot Chips 26 Symposium*, 2014.

[37] “NVIDIA Volta Unvieled,” 2017. [Online]. Available: https://www.anandtech.com/show/11367/nvidia-volta-unveiled-gv100-gpu-and-tesla-v100-accelerator-announced

[38] A. Kannan, N. D. E. Jerger, and G. H. Loh, “Enabling Interposer-Based Disintegration of Multi-Core Processors,” in *Proc. of the International Symposium on Microarchitecture*, 2015.

[39] M. O’Connor, N. Chatterjee, D. Lee, J. Wilson, A. Agrawal, S. W. Keckler, and W. J. Dally, “Fine-grained DRAM: Energy-efficient DRAM for Extreme Bandwidth Systems,” in *Proc. of the Annual IEEE/ACM International Symposium on Microarchitecture*, 2017.

[40] R. Chandra, S. Devine, B. Verghese, A. Gupta, and M. Rosenblum, “Scheduling and Page Migration for Multiprocessor Compute Servers,” in *Proc. of the International Conference on Architectural Support for Programming Languages and Operating Systems*, 1994.

[41] K. M. Wilson and B. B. Aglietti, “Dynamic Page Placement to Improve Locality in CC-NUMA Multiprocessors for TPC-C,” in *Proc. of the ACM/IEEE Conference on Supercomputing*, 2001.

[42] Z. Majo and T. R. Gross, “Matching Memory Access Patterns and Data Placement for NUMA Systems,” in *Proc. of the International Symposium on Code Generation and Optimization*, 2012.

[43] M. Dashti, A. Fedorova, J. Funston, F. Gaud, R. Lachaize, B. Lepers, V. Quema, and M. Roth, “Traffic Management: A Holistic Approach to Memory Placement on NUMA Systems,” in *Proc. of the International Conference on Architectural Support for Programming Languages and Operating Systems*, 2013.

[44] N. Vijaykumar, E. Ebrahimi, K. Hsieh, P. B. Gibbons, and O. Mutlu, “The Locality Descriptor: A Holistic Cross-Layer Abstraction to Express Data Locality in GPUs,” in *Proc. of the Annual International Symposium on Computer Architecture*, 2018.

[45] T. G. Rogers, M. O’Connor, and T. M. Aamodt, “Cache-Conscious Wavefront Scheduling,” in *Proc. of the Annual IEEE/ACM International Symposium on Microarchitecture*, 2012.

[46] Y. Tian, S. Puthoor, J. L. Greathouse, B. M. Bechmann, and D. A. Jiménez, “Adaptive GPU Cache Bypassing,” in *Proc. of the Workshop on General Purpose Processing Using GPUs*, 2015.

[47] X. Chen, L.-W. Chang, C. I. Rodrigues, J. Lv, Z. Wang, and W.-M. Hwu, “Adaptive Cache Management for Energy-Efficient GPU Computing,” in *Proc. of the International Symposium on Microarchitecture*, 2014.

[48] M. Rhu, M. Sullivan, J. Leng, and M. Erez, “A Locality-aware Memory Hierarchy for Energy-Efficient GPU Architectures,” in *Proc. of the Annual IEEE/ACM International Symposium on Microarchitecture*, 2013.

[49] A. Arunkumar, S.-Y. Lee, and C.-J. Wu, “ID-Cache: Instruction and Memory Divergence Based Cache Management for GPUs,” in *Proc. of the IEEE International Symposium on Workload Characterization*, 2016.

[50] S. Lee and C. Wu, “Ctrl-C: Instruction-Aware Control Loop Based Adaptive Cache Bypassing for GPUs,” in *Proc. of the IEEE International Conference on Computer Design*, 2016.

[51] N. Vijaykumar, G. Pekhimenko, A. Jog, A. Bhowmick, R. Ausavarungnirun, C. Das, M. Kandemir, T. C. Mowry, and O. Mutlu, “A Case for Core-assisted Bottleneck Acceleration in GPUs: Enabling Flexible Data Compression with Assist Warps,” in *Proc. of the Annual International Symposium on Computer Architecture*, 2015.

[52] J. Kim, M. Sullivan, E. Choukse, and M. Erez, “Bit-plane Compression: Transforming Data for Better Compression in Many-core Architectures,” in *Proc. of the International Symposium on Computer Architecture*, 2016.

[53] A. Arunkumar, S. Y. Lee, V. Soundararajan, and C. J. Wu, “LATTE-CC: Latency Tolerance Aware Adaptive Cache Compression Management for Energy Efficient GPUs,” in *Proc. of the IEEE International Symposium on High Performance Computer Architecture*, 2018.

[54] S. Lee, K. Kim, G. Koo, H. Jeon, W. W. Ro, and M. Annavaram, “Warped-compression: Enabling Power Efficient GPUs Through Register Compression,” in *Proc. of the Annual International Symposium on Computer Architecture*, 2015.

[55] M. Abdel-Majeed, D. Wong, and M. Annavaram, “Warped Gates: Gating Aware Scheduling and Power Gating for GPGPUs,” in *Proc. of the Annual IEEE/ACM International Symposium on Microarchitecture*, 2013.

[56] V. Adhinarayanan, I. Paul, J. L. Greathouse, W. Huang, A. Pattnaik, and W. c. Feng, “Measuring and Modeling On-Chip Interconnect Power on Real Hardware,” in *Proc. of the IEEE International Symposium on Workload Characterization*, 2016.

[57] A. Majumdar, L. Piga, I. Paul, J. L. Greathouse, W. Huang, and D. H. Albonesi, “Dynamic GPGPU Power Management Using Adaptive Model Predictive Control,” in *Proc. of the IEEE International Symposium on High Performance Computer Architecture*, 2017.

[58] I. Paul, V. Ravi, S. Manne, M. Arora, and S. Yalamanchili, “Coordinated Energy Management in Heterogeneous Processors,” in *SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2013.

[59] J. Lee, V. Sathisha, M. Schulte, K. Compton, and N. S. Kim, “Improving Throughput of Power-Constrained GPUs Using Dynamic Voltage/Frequency and Core Scaling,” in *Proc. of the International Conference on Parallel Architectures and Compilation Techniques*, 2011.

[60] G. Pekhimenko, E. Bolotin, N. Vijaykumar, O. Mutlu, T. C. Mowry, and S. W. Keckler, “A Case for Toggle-Aware Compression for GPU Systems,” in *Proc. of the IEEE International Symposium on High Performance Computer Architecture*, 2016.

[61] M. Gebhart, D. R. Johnson, D. Tarjan, S. W. Keckler, W. J. Dally, E. Lindholm, and K. Skadron, “Energy-efficient Mechanisms for Managing Thread Context in Throughput Processors,” in *Proc. of the Annual International Symposium on Computer Architecture*, 2011.

[62] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou, “GPGPU Performance and Power Estimation Using Machine Learning,” in *Proc. of the IEEE International Symposium on High Performance Computer Architecture*, 2015.

[63] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, “McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures,” in *Proc. of the Annual IEEE/ACM International Symposium on Microarchitecture*, 2009.

[64] Y. Abe, H. Sasaki, S. Kato, K. Inoue, M. Edahiro, and M. Peres, “Power and Performance Characterization and Modeling of GPU-Accelerated Systems,” in *Proc. of the IEEE International Parallel and Distributed Processing Symposium*, 2014.

[65] S. Song, C. Su, B. Rountree, and K. W. Cameron, “A Simplified and Accurate Model of Power-Performance Efficiency on Emergent GPU Architectures,” in *Proc. of the IEEE International Symposium on Parallel and Distributed Processing*, 2013.

[66] J. Lim, N. B. Lakshminarayana, H. Kim, W. J. Song, S. Yalamanchili, and W. Sung, “Power Modeling for GPU Architectures Using McPAT,” *ACM Trans. Design Autom. Electr. Syst.*, vol. 19, no. 3, pp. 26:1–26:24, 2014.