ENGI 9861: HIGH-PERFORMANCE COMPUTER ARCHITECTURE

Energy-efficient DRAM systems for GPUs

Name: Viren Sagpariya

Student ID: 201991237

Supervisor: Dr. Ramachandran Venkatesan

Table of Contents

Table of Figures 3
Table of Tables 3
Abstract 4
Introduction 5
What is DRAM 6
Advantage of DRAMs 7
Challenge For DRAMs 7
The energy efficiency of DRAM 8
Reducing Activation Granularity 9
Subchannels 11
DRAM overheads 12
Reasons behind the GPU Power management 13
Technologies Used for Improving GPU Energy efficiency 13
1. DVFS 13
2. CPU-GPU Work Division based techniques 15
3. Saving Energy in GPU Components 15
4. Dynamic Allocation based techniques 16
5. Application-specific and programming-level techniques 16
Cache management in DRAM 17
Latency and Concurrency of the system 19
Multimodule GPUs 20
DRAM-System Performance 23
Conclusion 24

Table of Figures

Figure 1. DRAM systems on a chip 6
Figure 2. DDR4 memory 7
Figure 3. An average energy cost of scaling 9
Figure 4. Four mates in subarray activated in division.....10
Figure 5. Implementation of segmented wordline 10
Figure 6. The block diagram of NVIDIA GTX980 GPU board.....14
Figure 7. Scaling of GPU designs and NUMA GPU architecture 21

Table of Tables

Table 1. Specification of analyzed GPUs 22

Abstract

For any computable computer, GPUs are the most relevant. No device will operate without memory and the new tech in today's modern era. Not only for gaming, streaming video and rendering processes, and some editing method GPU is very much required. DRAM modules form an integral component of the GPUs. For new versions of CPUs for improved efficiency, it needs to be modified despite changes in power usage, electricity, and output as well. Still, with low heat generation and reduced power usage, the task is to boost efficiency.

Introduction

GPUs have always been favorite among youth, computer engineers, computer geeks, and especially gamers. With time passes, technology is being updated every 2-3 years, and it comes with more power and performance efficient CPUs and GPUs.

As for performance increases, it needs more power to generate more energy. For new high-performance clusters, the Graphics Processing Units (GPUs) have been powerful accelerators. We significantly improve the efficiency of a wide range of applications in many commercial and research fields.

The CPU-GPU hybrid computation is more energy-efficient than conventional concurrent multi-core computation. This kind of high-performance cluster, though, also absorbs a lot of resources. Significant expense remains to power the clusters. For example, the Titan supercomputer is accelerated by 18,688 NVIDIA Tesla K20X with an 8.21 million Watts power supply, which costs about $23 million per year. Efficient GPU power control is essential for GPU-accelerated data centers and supercomputers, provided that saving only a few percent of energy will reduce a significant amount of electricity costs.

Most of the tallest Obstacles to non-stop scaling of GPUs are Energy consumption by using a reminiscence system. A big component of DRAM publicity energy is linked to the reality that numerous high energy operations eat DRAM, such as row activations, And preloads should be rendered such that facts can be retrieved from a Row to DRAM (page). These operations are vital to ensure the Information from the proper row occurs in the buffer section, which Is a constrained hardware shape fixed to each DRAM Banking.

What is DRAM

Dynamic random-access memory (DRAM) is a form of semiconductor memory that is usually used by a CPU processor to work for the data or software code required. DRAM is a common type of reminiscent random entry (RAM) used in private computer systems (PCs), workstations, and servers. Random access accepts the CPU processor as an option to needing to move sequentially from a starting point to gain entry to some portion of the memory directly. RAM is positioned near the processor of a computer and allows for faster access to facts than storage media such as HDD and SSD. As shown in the figure, circuit boards containing DRAM looks similar to RAM of any PC.

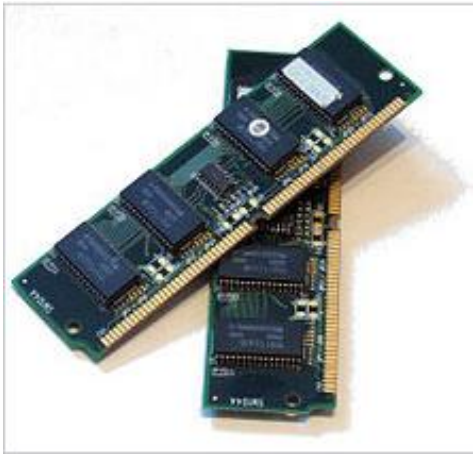

Figure 1. DRAM systems on a chip [8]

Advantage of DRAMs

The advantage of a DRAM is that the cell's versatility-it needs only one transistor as against around six in an exceedingly standard static RAM, SRAM memory cell. Despite its versatility, the expense of DRAM is way smaller than that of SRAM, so that they can have even higher memory capacity rates. Given that the DRAM needs the power to hold its records, it's what's called a fragile memory.

Challenge For DRAMs

Graphics processing units (GPUs) and other systems for processing speed have scalable efficiency through a parallel increase in computational power and overall memory bandwidth. Current computers would quickly need over 1 TB / s of bandwidth to proceed on this route, and devices used in Exascale computers are expected to need 4 TB / s of bandwidth for each device. Satisfying this increasing requirement for bandwidth without a major rise in the DRAM 's power budget is a key challenge.[1] The figure shows a group of vengeance LPX DDR4 DRAM.

Figure 2. DDR4 memory [8]

The energy efficiency of DRAM

The energy used to transfer data between the DRAM and processor is called I/O energy, which is required to transfer data from DRAM bit cells to the I/O interface[4].

Including the I / O energy used to move the data from the DRAM to the host device, the energy for a DRAM connection may be decomposed into two primary components, the row energy or energy for preloading a bank and enabling a row, and the column energy or energy for accessing and transferring a subset of data from the active row to the I / O pins[4].

Engineers who work on this suggest an energy-optimized DRAM architecture and memory controller. It can also improve the efficiency of other memory-intensive GPU programs, called subchannels. The main concept is to partition the DRAM storage in an area-efficient manner to minimize waste energy from activation and partition the Datapath in the DRAM array and the periphery to allow continued use of the parallel operations[4].

The energy-optimized DRAM and memory controller architecture, or subchannels, can even improve the efficiency of certain power-intensive GPU applications, to resolve problems. The main principle is to partition the DRAM power in an area-efficient way to eliminate excess activation energy and partition the datapath in the DRAM array and the periphery such that the full DRAM bandwidth is utilized by the concurrent processes.[4] As shown in figure 3, energy consumption increases with GPU capability that is performance

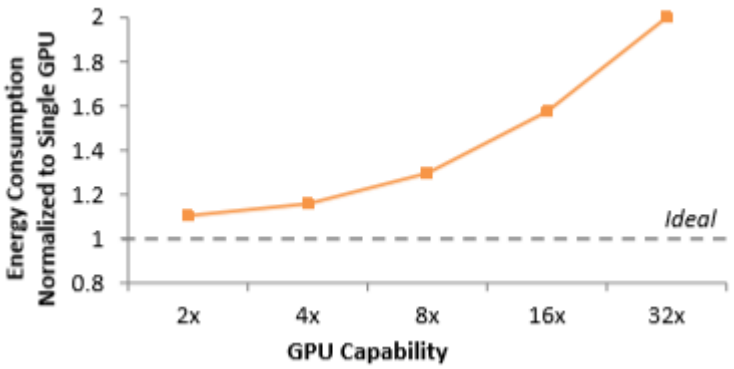

Figure 3. An average energy cost of scaling [5]

To reduce row energy of DRAM, some changes can be applied to achieve efficiency

- Reducing Activation Granularity
- Subchannels
- DRAM overheads

Reducing Activation Granularity

It is the number of bits accessed by an activate command.

When the master wordline (MWL) is proclaimed in current DRAMs, it drives a local wordline (LWL) in each of a subarray mats by triggering the corresponding localwordline driver (LWD) in the stripe of the sheet. To obtain partial row activation, the wordline signal will enter exactly certain cells that we want to enable in the control transistor. Another approach to do that will be to fragment the LWL, such that just a section of the LWL is switched on in any pad. However, because the LWL is laid out within the densest part of the DRAM chip in silicided polysilicon, disruptive modifications that disturb the LWL arrangement are quite inefficient in the region.

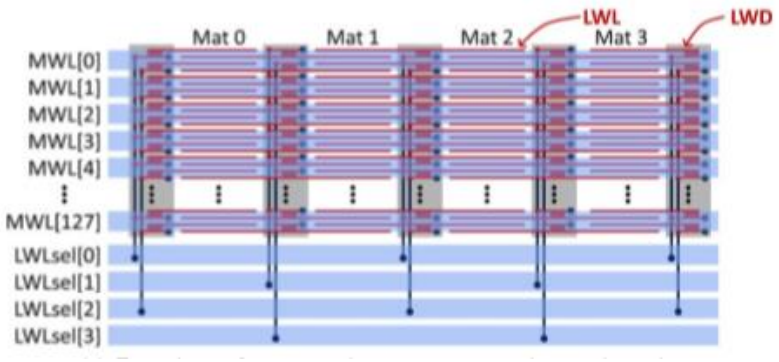

Figure 4. Four mates in subarray activated in division [4]

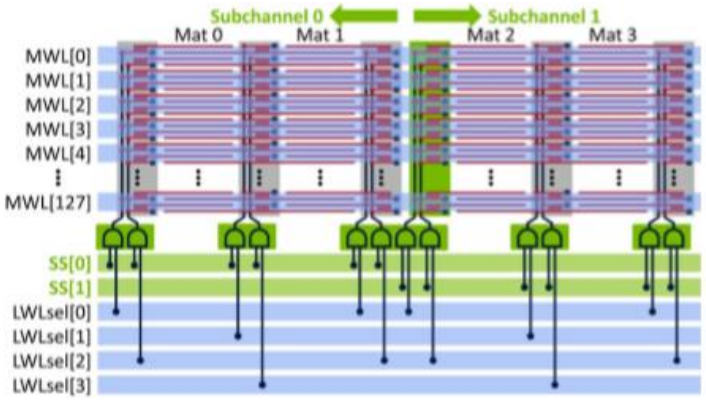

Figure 5. Implementation of segmented wordline [4]

Alternatively, the partnership between a MWL and its subservient LWLs cab is changed in such a way that declaration of a MWL contributes to the affirmation of the LWLs in only a handful of the subarray mats whilst the other mats stay untouched. Figure 4 and figure 5 demonstrates this procedure used to trigger just one-eighth of a section. While the entire MWL was powered, the corresponding LWLs were only triggered in mats 4 through 7, which consumed an eighth of the original row-energy. For selecting the appropriate portion of the row to trigger, the segment-select signals (SS) are properly asserted when the MWL is driven.[4]

To utilize this interface, the memory controller not only gives a trigger to the row address but also details regarding which section will be disabled in the form of a mask. This mask knowledge is used to say that part(s) of the row are triggered by SS signal air. Sending a mask often allows for the simultaneous activation of several segments from

a list. Therefore, not only separate non-contiguous parts of a row may be switched on but also the whole row as in the baseline.

Subchannels

Many narrow slices of the DRAM bank storage and datapath from the cells to the I / O strings, each of which is considered a subchannel.

Throughout the baseline, per 32B DRAM atom is interleaved through all the mats in a subarray and read out onto the 256 global sense-amplifiers (GSAs) using all the MDLs in one internal DRAM loop.

This configuration is changed in such a way that the DRAM atom is interleaved just over the mats, which are triggered in synchronization, such that it can be retrieved from such mats alone in its entirety. DRAM atom from the 8th proportion of the mats in one step will involve increasing the output width from 8 bits to 64 bits for each mat. Nevertheless, by 8 increasing the data path width (LDLs and MDLs) reduces the wiring tracks in the layers M2 and M3, thereby increasing the height and width of the mat and raising the GSA region required in the periphery proportionally. It resulted in a prohibitively large overhead region of 34.5 percent for broad bandwidth mats that was overlooked reading an atom from a subset of mats. Through a DRAM transfer point of view, it now appears that the data is processed and recovered from a channel that is 1/8th the bandwidth of a reference channel with proportionally less mats and datapath services (LDLs, MDLs, GSAs, and I / Os).

Subchannels can be enhanced by using parallel execution that is called a **Pipelined burst**.

In a single internal process, the whole DRAM atom is fetched from the row buffer into the I / O buffer. Nonetheless, the same switch requires 8 internal cycles with 8 subchannels, owing to the short datapath.

Instead of waiting for the whole DRAM atom to fill in the I / O buffer until the data burst begins on the main I / O path, it is feasible to pipeline the internal and external bursts. So the first data beat occurs on the external I / O when the delay period for the CAS, tCAS, has passed after the column-read instruction, the same as in the baseline. The sprint currently requires 8 cycles DDR instead of 1 cycle DDR. For a single burst, multiple Column-SelectLines (CSLs) must be asserted sequentially in a subchannel.

DRAM overheads

The subchannel architecture requires two overheads from field sources. Next, the seven additional Local Wordline Driver (LWD) strips needed to split a row into eight segments increase the subarray area by 1.19 percent (each LWD stripe is mat size of 5.7 percent). Furthermore, the segment required additional metal wires for the selection signals.[4]

The HBM design provides two different implementations for the commands. The GUI of column-command transfers read/write commands and addresses of columns on two sides of a clock. The column-command interface then needs four additional signals to submit the 8-bit subchannel mask for each input. The row-command interface sends an activation order and four edges of the clock's row address. Therefore, transmitting the mask over four edges requires two additional signals from the 8-bit subchannel layer. The precharge commands have four unused bits, which along with the two additional signals, will pass the subchannel mask over the two edges of a precharge request. So an 8-channel HBM stack needs 48 additional signal bumps. Such additional bumps mark an improvement of 3.1 percent in the amount of I / O signals required by each stack.[4]

Reasons behind the GPU Power management

1. Addressing Inefficient Resource Usage
2. Ensuring Reliability
3. Providing Economic Gains
4. Enabling Performance Scaling
5. Enabling Deployment in Applications
6. Sustainable Computing

Technologies Used for Improving GPU Energy efficiency

There are mainly five categories that are used to improve Efficiency in GPUs listed below

1. DVFS
2. CPU-GPU workload division based techniques
3. Architectural techniques that save Energy in GPU Components
4. Workload variation techniques that distribute resources dynamically
5. Application-specific and programming-level techniques

1. DVFS

Dynamic Voltage and Frequency Scaling (DVFS), It is one of the promising electricity administration strategies which refers to altering the voltage/frequency of the processor at some stage in undertaking processing.[1] It is beneficial either for power conservation or effectivity enhancements. The CPU DVFS platform has been properly acknowledged and used in each non-public computer structures as well as large-scale clusters. The GPU DVFS analysis started simply a few years ago, given the complexity of CPU DVFS.[1]

For CMOS circuits, total power consumption is denoted by P_{total}

$P_{total} = P_{dynamic} + P_{leakage} + P_{short\text{-}circuit} + P_{DC}$

, where $P_{dynamic}$ stands for the dynamic power, which is generated when transistors switch their states. $P_{leakage}$, $P_{short\text{-}circuit}$ and P_{DC} together stand for the static power.[2]

DVFS changes the voltage and frequency of the runtime supply, and it mainly affects the dynamic power. In the early processor systems, the dynamic power accounts for the bulk of electricity usage, but nowadays, the static power still contributes considerably. The GPU boards usually have two sets of customizable voltage/frequency: the main voltage/frequency, and the voltage/frequency of the memory. The core and memory voltage relate to the GPU SMs and the DRAM supply voltage. The central frequency influences the speed of operation of SM when the frequency of the memory directly affects the I / O performance of DRAM.[1]

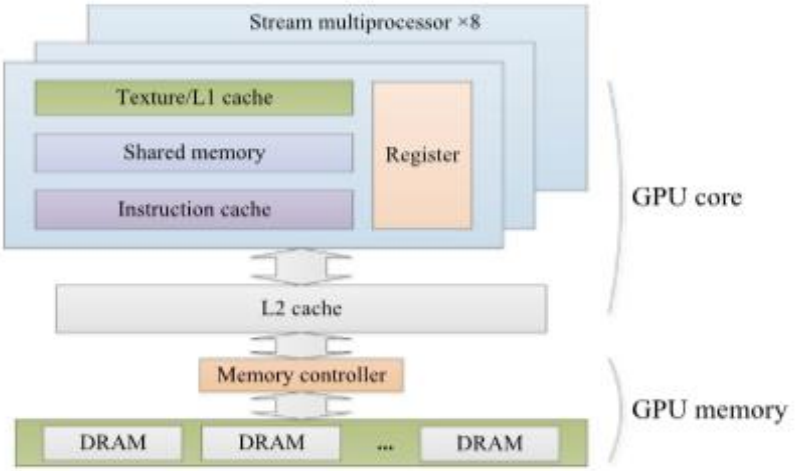

Figure 6. The block diagram of NVIDIA GTX980 GPU board [2]

Challenges of DVFS

There are some challenges that DVFS is facing listed below,

Initially, knowledge regarding GPU architecture and power management is quite minimal. Second, detailed computational GPU DVFS performance/power calculation methods are missing here. Finally, the design of the GPU architecture is advancing very rapidly,

and performing the same DVFS strategy may have different results on different generations of GPUs.

2. CPU-GPU Work Division based techniques

It is a method for the energy control of heterogeneous GPU-CPU architectures. The technique works in two stages. Throughout the first stage, the load of work is split between the CPU and GPU depending on the requirements of the workload in such a way that both sides will achieve their tasks roughly concurrently. The function is done by the CPU and GPU, for example, maybe 15% and 85%, respectively. This move ensures load balancing, which is also prevented due to idling energy loss. In the second stage, the frequency of GPU cores and memory are balanced along with the frequency and voltage of the CPU to achieve the highest possible energy savings with a limited deterioration of the output.[1]

3. Saving Energy in GPU Components

Various methods allow architecture-level improvements to GPUs in order to maximize the energy wasted on individual GPU components. Such strategies use the basic consumption pattern of GPU components to allow energy-saving run-time adaptations.

There is an Energy-saving strategy in GPU 's main datapath. Because GPUs hire a large number of threads, it requires a significant amount of on-chip storage to store the registry history of such threads. The GPU's thread scheduler always has to pick a thread from a wide number of threads to run. Accessing massive registry files and scheduling for a huge number of threads requires a substantial amount of resources for these purposes.

There are two ways to fix this. First, to record data, a tiny storage system is introduced, which functions as a cache and holds the operating range of registers to reduce energy usage. Furthermore, the

threads are split arbitrarily into two types: current threads and pending threads. Thus, the scheduler wants to regard only the active threads, which are far smaller in size, in each loop, which results in significant energy savings.[1]

4. Dynamic Allocation based techniques

It is well established that there are wide differences in the intra-application and inter-application specifications of various applications in their source. Indeed, even real-world implementations barely use any of the GPU's computational capabilities. Important energy savings may thus be accomplished by dynamically adjusting the components which exhibit low rates of consumption.

AMD uses PowerPlay technologies for dynamic power management between commercial products. This switches the GPU continuously between low, medium, and high states, depending on the GPU load. For example, when a graphics program is operating, there is a heavy demand on the GPU, and so it operates in a high-power environment. Conversely, the demand on the GPU is negligible when typing emails, and thus it operates in a low-power mode. The power-saving also reduces fan noise and system temperatures. NVIDIA also uses PowerMizer technologies to control complex power.

5. Application-specific and programming-level techniques

It has been observed that transformations at the source-code level and application-specific modifications will dramatically increase GPUs' resource usage, consistency and energy consumption. So, significant energy savings can be achieved by manually or automatically optimizing GPU implementation and solving bottlenecks in performance.

One way to conserve resources is by utilizing Kernel Fusion GPUs. Kernel fusion incorporates two processor computations into a single

line. It therefore contributes to a balance of the need for hardware services, which increases resource usage and thereby boosts energy performance. The authors articulate the kernel fusion function as a question of dynamic programming, which can be solved with traditional resources.

The second strategy for saving energy is the task-parallel execution of complex linear algebra operations by replacing the busy waits intelligently with a power-friendly blocking environment. Execution of these functions requires a Processor thread releasing the kernel and then waiting in a busy polling loop for the next ready job. This adds to carbon wastage. To stop this, their strategy blocks the Processor thread on an initial synchronization while waiting for the GPU to complete its job, resulting in energy savings.

Cache management in DRAM

Energy-efficiency has become a crucial element of architecture as the semiconductor industry shifts from multicore processors to manycore processors. Heterogeneous systems have been suggested to increase energy performance as various models of computing engines may be adapted for specific forms of patterns of computation. E.g., in the high-performance computing environment, using CUDA or OpenCL for general-purpose computation on graphics processing units (GPGPUs) has become omnipresent due to its improved energy-efficiency over traditional multicore CPUs when working with data-parallel kernels.[3]

Caches have been covered in GPUs to take advantage of the reuse of on-chip data, which can provide good-sized tempo to compose clearly for different applications.[3] However, the current architecture of GPU cache and it's manage schemes are inefficient when going for walks

memory-intensive applications that hamper system balance and energy-efficiency.[5]

Typical CPU cache design is designed for memory latency but does not automatically favour throughput-oriented processors such as GPUs, since massive multithreading renders cache localization difficult to grab, Unlike CPU caches, the output of GPU caches is hampered by thrashing, particularly inter-warp contention, which is far more prevalent in GPUs due to massive multithreading.

GPUs usually operate concurrently with heaps or lots of threads. They hence showcase a lot decrease cache area per thread, and a whole lot shorter cache line lifespan than CPUs, with a number of cache traces being eliminated before being reused.[3] Inter-warp combat takes place amongst warps designed to the identical middle of the SIMT. It contributes to the terrible temporal role and, consequently, to the deterioration of powerful results. Since the utility work set is generally a whole lot larger than the size of the cache, advanced alternative policies can now not resolve the GPU rivalry hassle.

There is a comprehensive cache management framework for GPGPUs, to address the drawbacks of mere bypassing and mere warp throttling. At runtime, cache contention and space congestion induced by major multithreading are observed, and the detector notifies the controller to enable dynamic bypass of the cache. The bypass strategy focused on distance reuse prevents hot cache lines from early removal, which is combined with warp throttling to prevent over-saturating on-chip space.[3]

There is a basic prediction utilizing a gradient approach focused on a hierarchical sampling of contention and concentration severity to determine the optimum number of active warps.

GPGPUs are throughput-oriented processors that can hide memory latency by massive multithreading. However, GPU caches are not designed with enough awareness of massive multithreading, leading to poor efficiency. A specialized cache management design for GPU computing is essential to improve performance and energy-efficiency.[3]

Latency and Concurrency of the system

The overhead of the essential reminiscence machine would not be reduced to zero even with zero-latency DRAM access, considering the fact that bus transactions nonetheless require time. For example, there is the apparent time to go addresses and records from and to the DRAM module over the bus.[5]

There are three approaches to improve DRAM devices.

The first is to improve the overhead of the memory bus. A variety of solutions may be taken, including modifying the structure of the requested medium, raising the accessible database system capacity, reducing memory system latency, and targeting overhead mechanisms due to queueing, turnaround, etc.[6]

The second approach is to increase the available concurrency on the memory channel. This can be done by encouraging the simultaneous access to different independent DRAM banks, either via the same channel or through Replicate services to have growing, independent networks for multiple DRAM banks or by a mixture of both.[6]

The third approach is to raise on-channel latency. That is not the delay attributed to the DRAM core access time; it is the delay which is attributable to memory-bus resource stalling. E.g., if two read requests come back-to-back on the memory network, the second will

wait before the first stops the data bus and decreases its delay by the first length.[6]

Multimodule GPUs

Multimodule GPUs bring into a new age of design concerns where duplication of individual GPMs is fast. Still, it is challenging to sustain aggregated architecture with adequate interconnected bandwidth and the position of the memory device. Style criteria suitable for tiny Numbers of GPMs (or monolithic GPUs) may no longer be optimal for large numbers of GPMs due to differences in the field of physical integration. To understand and explain these design decisions in scalable GPU systems, we are proposing a new metric called Energy-Delay-Product Scaling Efficiency (EDPSE), which takes into account both strong scaling performance and energy efficiency. This allows us to compare two GPU designs, which can have dramatically different hardware resources and understand whether the design is living up to its potential for performance and energy efficiency. [5]

The figure shows the scaling of GPU designs for recent GPUs and transitions of DRAM and GPU modules on different levels.

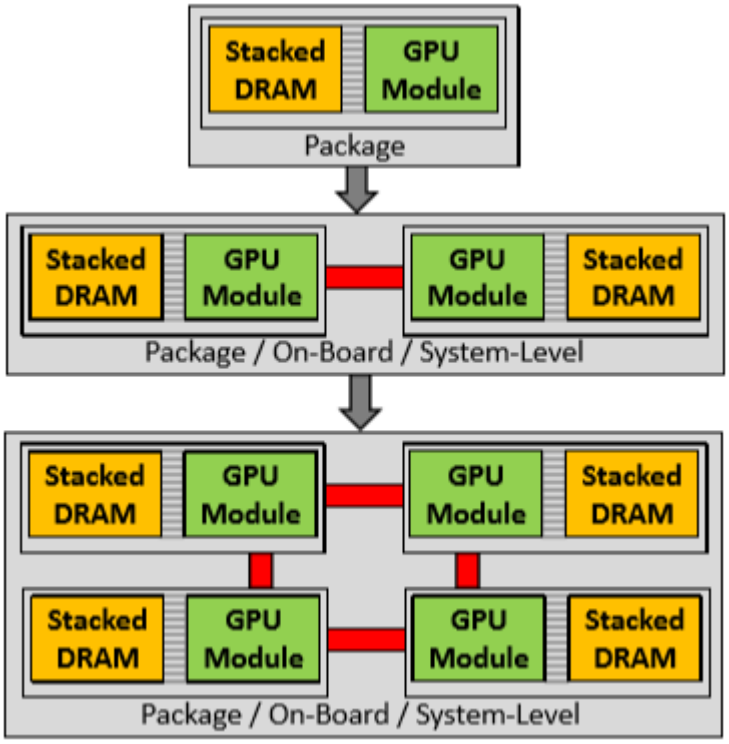

Figure 7. Scaling of GPU designs and NUMA GPU architecture [5]

EDPSE is a term that measures GPU energy efficiency at scale.

Metrics like output, power or energy may be used to equate two different hardware designs. In devices with a largely defined range of hardware resources, this typically helps one to concentrate on results as a merit metric. There are also measures such as the energy-delay product (EDP) or ED2, which combine time and output to make comparisons between two projects on an equal basis, for detailed energy efficiency assessments.[5]

While it is useful to compare designs with comparable tools, metrics such as EDP are not suitable to explore the efficiency of various design points when we scale GPU nodes from 1 to N.[1][5]

To understand multimodule GPU EDPSE patterns, there is a modern, instruction-based GPU energy estimation system, named **GPUJoule**.[5]

Not only is GPUJoule reliable, but it is also completely decoupled from microarchitectural complexities in the GPU. This is built on the basis of the following concept – the average usage of GPU resources is the combined energy expense of different forms of instructions executed on a GPU (or GPM), plus the flow of data inside the device to facilitate such instructions.[5] Through concentrating on instruction execution and macro-level data transfer, GPUJoule is ideal for device-level scaling studies where, for example, various multimodule GPU architectures may be applied through multiple caching organizations where the precise cache architecture specifics are less relevant.

Techniques such as locality-conscious thread-block (CTA) scheduling and data positioning, advanced-cache control approaches, and data compression techniques need to be re-applied not only inside today's GPUs, but also across GPU modules. Furthermore, system-level techniques to reduce the impact of constant power in the presence of large numbers of GPU modules would be crucial. Techniques such as advances in the interface process, smart clock-gating, and power-gating will increase the energy performance of multi-module GPUs.

Table 1. Specification of analyzed GPUs [7]

Architecture		Fermi	Kepler					Maxwell		Pascal	
Chip Model		GF110	GK104		GK110			GM200	GM204	GP102	GP104
Device		GTX580	GTX660 Ti	GTX 680	Tesla K20	Tesla K40	GTX 780 Ti	Titan X	GTX980	Titan X	GTX1080
Core Frequency (MHz)		1594	1058	1006	705	745	875	987	1170	1417	1607
Memory Frequency (MHz)		2024	3004	3004	2600	3004	3500	3505	3505	5000	5000
SMs		16	7	8	13	15	15	24	16	28	20
Cores per SM		32	192	192	192	192	192	128	128	128	128
SP/DP units per SM		32/4	192/8	192/8	192/64	192/64	192/8	128/4	128/4	128/4	128/4
FP Performance (FMA)	SP Flops/clk (GFlops/s)	1024 (1632.26)	2688 (2843.90)	3072 (3090.43)	4992 (3519.36)	5760 (4291.20)	5760 (5040.00)	6144 (5376.00)	4096 (4042.75)	7168 (5078.53)	5120 (8227.84)
	DP Flops/clk (GFlops/s)	128 (204.03)	112 (118.50)	128 (128.77)	1664 (1173.12)	1920 (1430.40)	240 (210.00)	192 (168.00)	128 (126.34)	224 (317.41)	160 (257.12)
Shared / L1	Transactions/clk	8 / 8	7 / 7	8 / 8	13 / 13	15 / 15	15 / 15	24 / -	16 / -	28 / -	20 / -
	SP (GB/s)	1520.16	882.86	959.40	1092.55	1332.16	1564.62	2503.40	1882.55	4729,75	3831,39
	DP (GB/s)		1765.73	1918.79	2185.11	2664.33	3129.24				
L2 Cache	Banks	6	3	4	5	6	6	12	8	-	-
	Slices per Bank	2	4	4	4	4	4	2	2	-	-
Device Memory	Bus width	384	192	192	320	384	384	384	256	384	256
	Bandwidth (GB/s)	179.17	134.29	179.05	193.72	268.58	312.92	313.28	208.85	447,04	298,02

As shown in table 1, GPUs are different as device changes with specification and performance changes.

DRAM-System Performance

Memory is one of the critical bottlenecks of modern structures. However, a variety of experiments suggest that the memory bus accounts for a large portion of the overhead of the main memory inside the information network.[7]

There are several strategies to reduce the overhead of the primary memory system. Those have been generally categorized between strategies that concentrate on interface DRAM and those that rely on system or bus component.

Increasing the DRAM density has become a clear DRAM driven solution.

Another solution is to increase latency at DRAM. Recently DRAM vendors revealed various key variations that increase access speed.[7] Another solution is to allow the power channel more efficient. It may be achieved by facilitating continuous exposure via the same platform to separate, independent DRAM banks, or by replicating services and supplying several, similar channels to specific DRAM banks — or by merging the two.[7]

The other solution is to raising on-channel latency. It isn't the latency attributable to the DRAM core access time; it's the latency related to memory-bus network stalling.[6][7]

Performance can be increased by using parallel system execution and using pipeline functionality into the system for dividing the workload

Conclusion

Without advances in the design and device level, multi-module GPUs would easily face energy consumption issues while attempting to scale output at all costs. However, focussing on the DRAM alone is not enough SRAM as well as other components are also needed for improving performance with reduced power consumption.

We have seen some changes and strategies to improve the efficiency of DRAM and some technologies that used to increase GPU energy efficiency and some reasons for the improvement.

The requirement to minimize resource usage of emerging technology would remain the same as today, as improved output would allow more resources to be measured and operated in conjunction with other devices and systems.

References

[1]"A Survey of Methods for Analyzing and Improving GPU Energy Efficiency | ACM Computing Surveys", *Dl.acm.org*, 2020. [Online].

[2] X. Mei, Q. Wang and X. Chu, "A survey and measurement study of GPU DVFS on energy conservation", *Digital Communications and Networks*, vol. 3, no. 2, pp. 89-100, 2017.

[3] X. Chen, L. Chang, C. I. Rodrigues, J. Lv, Z. Wang and W. Hwu, "Adaptive Cache Management for Energy-Efficient GPU Computing," 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, 2014, pp. 343-355, doi: 10.1109/MICRO.2014.11.

[4] N. Chatterjee et al., "Architecting an Energy-Efficient DRAM System for GPUs," 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), Austin, TX, 2017, pp. 73-84, doi: 10.1109/HPCA.2017.58.

[5] A. Arunkumar, E. Bolotin, D. Nellans and C. Wu, "Understanding the Future of Energy Efficiency in Multi-Module GPUs," 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), Washington, DC, USA, 2019, pp. 519-532, doi: 10.1109/HPCA.2019.00063.

[6] H. Wang and A. Jog, "Exploiting Latency and Error Tolerance of GPGPU Applications for an Energy-Efficient DRAM," 2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Portland, OR, USA, 2019, pp. 362-374, doi: 10.1109/DSN.2019.00046.

[7] V. Cuppu and B. Jacob, "Concurrency, latency, or system overhead: Which has the largest impact on uniprocessor DRAM-system performance?," Proceedings 28th Annual International Symposium on Computer Architecture, Goteborg, Sweden, 2001, pp. 62-71, doi: 10.1109/ISCA.2001.937433.

[8] https://en.wikipedia.org/wiki/Dynamic_random-access_memory