Briefing on FlashAttention-2: Faster and More Efficient Transformers

FlashAttention-2 is a significant advancement over its predecessor, FlashAttention, designed to address the primary bottleneck in scaling Transformer models to longer sequence lengths: the attention layer. The standard attention mechanism's runtime and memory requirements scale quadratically with sequence length, hindering progress in domains requiring long-context understanding, such as processing books, high-resolution images, and video.

While the original FlashAttention provided a 2-4x speedup by optimizing memory access between the GPU's High Bandwidth Memory (HBM) and SRAM, it still achieved only 25-40% of the theoretical maximum FLOPs/s. FlashAttention-2 introduces superior work partitioning and parallelism strategies to overcome these limitations. The key innovations are:

1. **Algorithmic Tweaks:** Reducing the number of slow, non-matrix multiplication (non-matmul) FLOPs, which can be up to 16x more expensive than matmul FLOPs on modern GPUs.

2. **Enhanced Parallelism:** Parallelizing computation across the sequence length dimension, in addition to the batch and head dimensions. This significantly improves GPU occupancy and resource utilization, especially for long sequences where batch sizes are typically small.

3. **Improved Warp-Level Partitioning:** Restructuring how work is distributed among warps (groups of threads) within a GPU thread block to drastically reduce communication and data transfer through shared memory.

These improvements result in an approximate **2x speedup over the original FlashAttention**. Benchmarks on an NVIDIA A100 GPU show FlashAttention-2 reaching **50-73% of the theoretical maximum FLOPs/s**, bringing its efficiency much closer to that of highly optimized matrix-multiply (GEMM) operations. In end-to-end training of GPT-style models, FlashAttention-2 achieves a training speed of up to **225 TFLOPs/s per A100 GPU**, representing 72% model FLOPs utilization.

1. The Challenge of Scaling Attention
The core of the Transformer architecture is the attention layer, which enables the model to weigh the importance of different tokens in a sequence. However, this mechanism is computationally intensive.

1.1. Quadratic Complexity

For an input sequence of length N and a head dimension d, the standard attention computation is: `O = softmax(QK^T)V`

The primary issue lies in the computation and storage of the intermediate matrices:
• **S = QKT:** An $N \times N$ matrix.
• **P = softmax(S):** Also an $N \times N$ matrix.
Standard implementations materialize these matrices in the GPU's High Bandwidth Memory (HBM). This leads to:

• **Quadratic Memory Usage (O(N2)):** Storing the *N x N* matrices becomes prohibitive as sequence length *N* increases. For example, recent models like GPT-4 (32k context), MPT (65k), and Claude (100k) push these limits severely.

• **Slow Runtime:** The process is memory-bound, as it involves multiple read/write operations to the relatively slow HBM, limiting overall speed.

1.2. The FlashAttention Innovation

FlashAttention introduced an I/O-aware algorithm that computes the exact same output without approximation. It avoids materializing the full *S* and *P* matrices in HBM by using classical techniques:

• **Tiling:** The input matrices (Q, K, V) are loaded from HBM to fast on-chip SRAM in blocks.

• **Online Softmax:** Attention is computed for each block, and the outputs are rescaled and combined on-the-fly to produce the correct final result.

• **Recomputation:** During the backward pass, the attention matrix is recomputed from the blocks in SRAM, avoiding the need to store it.
This reduced memory reads/writes, changing memory usage from quadratic O(N²) to linear O(N) and delivering a 2-4x speedup. However, its efficiency was still far below the theoretical maximum of the hardware due to suboptimal work partitioning.
---

2. FlashAttention-2: Core Optimizations

FlashAttention-2 builds upon this foundation with three critical improvements that enhance parallelism and work partitioning at multiple levels of the GPU execution hierarchy.

2.1. Algorithmic Tweaks for FLOPs Efficiency

Modern GPUs like the NVIDIA A100 have specialized Tensor Cores that make matrix multiplication (matmul) operations up to 16 times faster than other floating-point operations (non-matmul FLOPs). FlashAttention-2 tweaks the online softmax algorithm to minimize these slower operations.

• **Forward Pass:** Instead of rescaling the output at each step of the inner loop, an "un-scaled" version is maintained. The final scaling operation is performed only once at the end of the loop over blocks.

• **Backward Pass:** For recomputation, the algorithm is modified to only require storing the log-sum-exp (L) instead of both the max (m) and the sum of exponentials (ℓ), reducing non-matmul calculations and storage.

Figure 1: Diagram of the FlashAttention forward pass, where K and V are partitioned into two blocks. By computing attention with respect to each block and rescaling, the correct output is achieved while avoiding expensive HBM reads/writes of the intermediate S and P matrices.

2.2. Enhanced Parallelism Across Sequence Length

The original FlashAttention parallelized its workload across the batch size and the number of attention heads. This model is inefficient when the sequence length is very long, as this forces a reduction in batch size to fit within GPU memory, leaving many of the GPU's streaming multiprocessors (SMs) idle.
FlashAttention-2 introduces an additional layer of parallelism across the sequence length dimension itself.

• **Forward Pass:** The outer loop of the algorithm, which iterates over blocks of the query matrix `Q`, is "embarrassingly parallel." Different thread blocks can now process different row blocks of the attention matrix independently and without communication.

• **Backward Pass:** The workload is parallelized by assigning thread blocks to column blocks of the attention matrix. Since this requires a shared update to the gradient `dQ`, efficient `atomic adds` are used for communication between thread blocks.

Figure 2: In the forward pass (left), workers (thread blocks) are parallelized, with each worker responsible for a block of rows. In the backward pass (right), each worker handles a block of columns.

2.3. Improved Work Partitioning Between Warps

Within a single thread block, work is further divided among warps (groups of 32 threads). The strategy for this division has a major impact on the need for communication via shared memory, which is a key performance limiter.

• **FlashAttention ("split-K" scheme):** The `K` and `V` matrices were split across warps, while the `Q` matrix was accessible to all. This required warps to write their intermediate results to shared memory, synchronize, and then combine them, creating a bottleneck.

(a) FlashAttention's "split-K" scheme

• **FlashAttention-2:** The `Q` matrix is split across warps, while `K` and `V` are accessible to all. Each warp can compute its slice of the output independently without needing to communicate with other warps. This reduction in shared memory traffic yields a significant speedup.

(b) FlashAttention-2's improved scheme

3. Empirical Validation and Performance

FlashAttention-2 demonstrates substantial performance gains over all existing exact attention implementations across a range of benchmarks on NVIDIA A100 and H100 GPUs.

3.1. Attention Layer Benchmarks

When measuring the runtime of just the attention layer (forward and backward passes), FlashAttention-2 is consistently faster.

• **Speedup vs. FlashAttention:** Approximately **2x faster** on average.
• **Speedup vs. Standard PyTorch:** Up to **10x faster**.
• **A100 GPU Performance:** Reaches up to **230 TFLOPs/s**, achieving **73% of the theoretical maximum throughput** in the forward pass and up to **63%** in the backward pass.
• **H100 GPU Performance:** Without H100-specific optimizations, FlashAttention-2 already achieves up to **335 TFLOPs/s**.

3.2. End-to-End Model Training

The benefits translate directly to faster end-to-end training of large language models. The following table shows training speed on 8xA100 80GB GPUs.

Model	Sequence Context	Without FlashAttention	FlashAttention	FlashAttention-2	Speedup (FA2 vs Baseline)
GPT3-1.3B	2k	142 TFLOPs/s	189 TFLOPs/s	**196 TFLOPs/s**	1.38x
GPT3-1.3B	8k	72 TFLOPs/s	170 TFLOPs/s	**220 TFLOPs/s**	3.05x
GPT3-2.7B	2k	149 TFLOPs/s	189 TFLOPs/s	**205 TFLOPs/s**	1.38x
GPT3-2.7B	8k	80 TFLOPs/s	175 TFLOPs/s	**225 TFLOPs/s**	2.81x

As shown, FlashAttention-2 achieves up to **225 TFLOPs/s** (72% model FLOPs utilization), providing up to a **1.3x speedup** over the original FlashAttention and a **2.8x speedup** over a standard implementation, with the largest gains seen at longer sequence lengths.

4. Implications and Future Directions

The efficiency gains from FlashAttention-2 have profound implications for the AI field. The 2x speedup means that training a model with a 16k context length can be done for the same cost and time as previously training an 8k context model. This unlocks new capabilities for models designed to process long-form content such as books, reports, high-resolution images, audio, and video.

Future work planned for FlashAttention-2 includes:

• **Hardware Support:** Expanding applicability to new devices like H100 and AMD GPUs.
• **New Data Types:** Implementing support for FP8 precision.

• **H100 Optimizations:** Leveraging new hardware features like Tensor Memory Accelerator (TMA) and 4th-gen Tensor Cores for further speedups.
• **Algorithmic Integration:** Combining low-level optimizations with high-level algorithmic changes like block-sparse attention to enable models with even longer contexts.

3 GPU Tricks That Doubled Transformer Speed: A FlashAttention-2 Deep Dive

Introduction: The Attention Bottleneck

The attention mechanism is the heart of the Transformer architecture, but it has a fundamental problem: its computational and memory costs grow quadratically with the length of the input sequence. For years, this has been the main bottleneck preventing AI models from efficiently processing very long texts, high-resolution images, or videos.

Despite this limitation, the demand for longer context windows has exploded, with models like GPT-4 handling 32k tokens and Claude reaching 100k. The first major breakthrough to address this was FlashAttention, an algorithm that reordered the attention computation to reduce memory usage from quadratic to linear. While revolutionary, FlashAttention still wasn't as fast as it could be, reaching only 30-50% of a GPU's theoretical max performance in its forward pass.
This article explores FlashAttention-2, a successor that achieves a roughly 2x speedup over the original. The secret isn't new hardware, but a series of clever, hardware-aware algorithmic tricks that squeeze maximum performance out of existing GPUs. Let's dive into the three key optimizations that made this possible.

1. Not All Math is Created Equal: Minimizing "Expensive" Operations

The first lesson in hardware-aware programming is that not all math is created equal. Modern GPUs like the NVIDIA A100 are highly specialized; they contain dedicated hardware, known as Tensor Cores, that are incredibly fast at matrix multiplication (matmul). However, other mathematical operations—referred to as "non-matmul FLOPs"—can be up to 16 times slower to execute on the same hardware.

The original FlashAttention algorithm, for all its memory efficiency, spent too much time on these "expensive" non-matmul operations. FlashAttention-2 tackles this hardware limitation head-on with two specific algorithmic tweaks:

1. **Smarter Scaling:** The original algorithm repeatedly rescaled intermediate results during its online softmax calculation. The new version avoids these costly division operations by maintaining an "un-scaled" version of the output throughout its inner loop, only applying the final scaling operation once at the very end.
2. **Optimized Backward Pass:** To calculate gradients in the backward pass, the original algorithm needed to save two statistics from the forward pass: the `max` and the `sum of exponentials` of the attention matrix rows. FlashAttention-2 fuses these into a single value—the `logsumexp`—reducing both the number of values to store and the number of non-matmul operations needed during recomputation.
These subtle changes significantly reduce the time the GPU spends on slow operations, ensuring it spends as much time as possible doing the fast matrix math it was designed for.

2. Leave No Core Behind: Parallelizing Across the Sequence Itself

A key hardware limitation of any GPU is achieving high "occupancy"—that is, keeping all of its powerful processing units busy. An NVIDIA A100 GPU, for example, has 108 streaming multiprocessors (SMs). If your workload isn't parallel enough, many of these SMs will sit idle, wasting potential performance.

The original FlashAttention parallelized its workload across the batch size and the number of attention heads. This worked well for large batches. However, when processing very long sequences, the batch size must often be kept small to fit in memory. This led to low occupancy, leaving many of the GPU's 108 SMs with nothing to do.

FlashAttention-2 solves this by introducing a new dimension of parallelism. The work is now also split up *along the sequence length dimension*, ensuring the GPU is saturated with work even with a small batch size. Crucially, the strategy is different for the forward and backward passes to match the data flow of each computation:

• In the **forward pass**, different workers (thread blocks) are assigned to compute different *rows* of the final attention output matrix, which can be done completely independently.

• In the **backward pass**, the work is instead split by *columns*, as this aligns with how gradients are calculated.

This clever division of labor dramatically improves GPU occupancy, ensuring that more of the chip is actively working, especially in the common scenario of long sequences and small batch sizes.

3. Reduce the Chatter: Smarter Work-Sharing Inside the GPU

Within a single worker unit (a thread block) on a GPU, the work is further divided among smaller teams of threads called "warps." A major hardware bottleneck can arise from how these warps collaborate: the on-chip shared memory they use to communicate is fast, but it's not infinite, and excessive traffic creates slowdowns.

The original FlashAttention used a work-sharing strategy called the **"split-K" scheme**. Here, the Key (K) and Value (V) matrices were split among the warps. To compute their part of the final output, the warps had to constantly write partial results to shared memory, wait for each other to finish (synchronize), and then read the results back to combine them. This constant communication created a bottleneck, like a team of workers who have to stop and talk after every small task.

FlashAttention-2 eliminates this chatter by changing the work partitioning. Instead of splitting the K and V matrices, it splits the Query (Q) matrix among the warps. With this new arrangement, each warp computes a complete, independent slice of the final output. There is no need for an intermediate reduction step across warps, which eliminates the shared memory communication bottleneck entirely. This reduction in shared memory reads and writes is a key source of the algorithm's speedup.

The Real-World Impact: What a 2x Speedup Actually Means

The combination of these three techniques results in substantial performance gains. According to benchmarks on an NVIDIA A100 GPU, FlashAttention-2 is:

• Around **2x faster** than the original FlashAttention.

• **3 to 10x faster** than a standard PyTorch implementation of attention.

This allows it to reach up to **73% of the theoretical maximum FLOPs/s** in the forward pass. When used for end-to-end model training, it achieves speeds of up to **225 TFLOPs/s per A100 GPU**, which translates to an impressive 72% model FLOPs utilization.

The practical value of this speedup is best summarized by the paper's authors:

"FlashAttention-2 is 2× faster than FlashAttention, which means that we can train models with 16k longer context for the same price as previously training a 8k context model, for the same number of tokens."

Conclusion: Unlocking the Future of Long-Context AI

FlashAttention-2 is a powerful reminder that groundbreaking performance improvements don't always require new hardware. By developing smarter, hardware-aware algorithms, we can unlock the untapped potential of the tools we already have.

These optimizations will directly speed up the training, fine-tuning, and inference of countless Transformer models. As the computational barriers to processing vast amounts of information continue to fall, we are left to wonder: Now that the computational barriers to extremely long context windows are falling, what entirely new applications for AI might become possible?

Study Guide: FlashAttention-2

Quiz

Answer the following questions in 2-3 sentences each, based on the provided source context.

1. What is the fundamental problem with the standard attention mechanism in Transformers that FlashAttention-2 aims to solve?
2. How did the original FlashAttention reduce the memory requirement of the attention layer from quadratic to linear?
3. What were the primary sources of inefficiency in the original FlashAttention that limited its performance relative to theoretical maximums?
4. Explain one of the algorithmic tweaks in FlashAttention-2 designed to reduce the number of non-matmul FLOPs.
5. How does FlashAttention-2 improve parallelism, especially in scenarios involving long sequences and small batch sizes?
6. Describe the "split-K" work partitioning scheme used in the original FlashAttention's forward pass and explain why it was suboptimal.
7. How did FlashAttention-2 change the work partitioning between warps in the forward pass to improve efficiency?
8. According to the provided benchmarks, what is the approximate speedup of FlashAttention-2 compared to the original FlashAttention?
9. What is the role of causal masking, and how does FlashAttention-2 efficiently implement it to gain a speed advantage?
10. Describe the difference between HBM and SRAM in the GPU memory hierarchy and explain why this is critical to the design of FlashAttention.

Answer Key

1. The standard attention mechanism's runtime and memory requirements increase quadratically with the input sequence length. This creates a major bottleneck for

scaling Transformers to longer contexts, such as those needed for understanding books or high-resolution images.
2. The original FlashAttention used a technique called tiling, which loads blocks of inputs from HBM to SRAM and computes attention on these blocks. By reordering the computation and using online softmax, it avoids materializing and writing the large intermediate attention matrices (S and P) to HBM, thus reducing memory usage to be linear in sequence length.
3. The inefficiency in the original FlashAttention stemmed from suboptimal work partitioning between different thread blocks and warps on the GPU. This resulted in either low occupancy (underutilization of GPU resources) or unnecessary shared memory reads and writes between warps.
4. FlashAttention-2 modified the online softmax calculation so that the output update is maintained in an "un-scaled" state until the very end of the loop. This avoids repeated scaling operations, which are non-matmul FLOPs and are significantly slower to execute on modern GPUs than matmul FLOPs. Another tweak was storing only the logsumexp for the backward pass, rather than both the max and the sum of exponentials.
5. In addition to parallelizing over the batch size and number of heads, FlashAttention-2 parallelizes the computation along the sequence length dimension. This increases GPU occupancy, ensuring more of the streaming multiprocessors are utilized, which provides a significant speedup for long sequences where batch sizes are often small.
6. The "split-K" scheme in FlashAttention's forward pass split the K (key) and V (value) matrices across different warps while Q (query) was accessible by all. This was suboptimal because all warps needed to write their intermediate results to shared memory, synchronize, and then read from shared memory to sum the results, causing slowdowns.
7. FlashAttention-2 inverts the partitioning scheme for the forward pass: it splits the Q matrix across warps while K and V are accessible by all. This eliminates the need for communication between warps, as each warp can compute its slice of the output independently, thereby reducing shared memory reads/writes.
8. The benchmarks show that FlashAttention-2 achieves around a 2x speedup compared to the original FlashAttention. End-to-end training of GPT-style models showed speedups of up to 1.3x over FlashAttention.
9. Causal masking is used in auto-regressive language modeling to prevent a token from attending to future tokens (i.e., setting S_{ij} to $-\infty$ for $j > i$). FlashAttention-2 efficiently skips computation for entire blocks where all column indices are greater than the row indices, leading to a speedup of around 1.7-1.8x compared to non-causal attention.
10. High Bandwidth Memory (HBM) is the large, main GPU memory with high capacity but lower bandwidth, while on-chip SRAM (shared memory) is much smaller

but has significantly higher bandwidth. FlashAttention's design is critical on this distinction; it leverages fast SRAM to perform computations on "tiles" of data, minimizing the slow and expensive reads/writes to HBM.

Essay Questions

Answer the following questions in a detailed essay format, synthesizing information from across the source document.

1. Trace the evolution from the standard attention implementation to FlashAttention and finally to FlashAttention-2. For each stage, describe the primary performance limitations and explain how the subsequent version's algorithmic and architectural innovations addressed them.
2. Analyze the deep interplay between GPU hardware characteristics and the algorithmic design of FlashAttention-2. Discuss how the GPU's memory hierarchy (HBM vs. SRAM), execution model (thread blocks, warps), and specialized compute units (Tensor Cores) directly influenced the development of its core techniques.
3. Explain the concepts of parallelism (across thread blocks) and work partitioning (within a thread block) as implemented in FlashAttention-2. Contrast the strategies used for the forward and backward passes, detailing why they differ and how each is optimized for its specific computational dependencies.
4. Using the empirical data presented in Figures 4, 5, 6, and Table 1, construct a comprehensive performance evaluation of FlashAttention-2. Compare its speed in TFLOPs/s against PyTorch, the original FlashAttention, and other implementations across different sequence lengths, head dimensions, and with/without causal masking.
5. Discuss the significance of FlashAttention-2 in the context of scaling Transformer models. Elaborate on the new applications it enables and the future research directions outlined in the paper, including optimizations for new hardware (like H100 GPUs) and integration with other algorithmic changes.

Glossary of Key Terms

Term	Definition
Attention	The core mechanism in Transformer models that computes the output O by taking a softmax of the dot product of queries (Q) and keys (K), and multiplying the result by values (V). The standard implementation has quadratic runtime and memory complexity with sequence length.
Backpropagation	The method used to calculate the gradients of a loss function with respect to the weights of a neural network, flowing backwards from the output. The backward pass of attention involves calculating gradients dQ, dK, and dV.
Causal Masking	A technique used in auto-regressive models where entries in the attention matrix S are masked (set to $-\infty$) to prevent a position from attending to subsequent positions. This ensures that the prediction for a token only depends on previous tokens.

FLOPs/s	Floating Point Operations Per Second. A measure of computational performance. TFLOPs/s refers to TeraFLOPs/s, or trillions of floating-point operations per second.
GEMM	General Matrix Multiply. A highly optimized routine for matrix multiplication. Optimized GEMM operations can reach 80-90% of a GPU's theoretical maximum throughput.
GPU	Graphics Processing Unit. A specialized processor with a massively parallel architecture, consisting of compute elements and a memory hierarchy, designed for high-throughput computation.
HBM (High Bandwidth Memory)	The large (e.g., 40-80GB on an A100) main memory of a GPU. It has high capacity but significantly lower bandwidth compared to on-chip SRAM.
Kernel	An operation or function executed on a GPU by a massive number of threads.
Model FLOPs Utilization (MFU)	A percentage that indicates how close the achieved training speed (in TFLOPs/s) is to the theoretical maximum computational throughput of the hardware for a given model.
Multi-Head Attention (MHA)	A variant of attention where the computation is performed in parallel across multiple "heads," each with its own Q, K, and V matrices.
Multi-Query Attention (MQA)	A variant of attention where multiple query heads attend to the same single head of key and value, designed to reduce the size of the KV cache during inference.
Occupancy	The fraction of a GPU's computational resources (specifically, its streaming multiprocessors) that are being actively used at a given time. Low occupancy indicates underutilization.
Online Softmax	A technique that allows splitting a softmax computation into blocks. It computes a "local" softmax for each block and then rescales the output of each block to produce the correct final result without approximation.
Recomputation	A technique used in FlashAttention's backward pass where intermediate values (like the attention matrices S and P) are re-calculated on-the-fly from inputs stored in fast SRAM, rather than being stored in and re-read from slow HBM.
SRAM (Static RAM)	Fast, on-chip memory available to each streaming multiprocessor on a GPU. It has much lower capacity (e.g., 192KB per SM on A100) but dramatically higher bandwidth than HBM. Also referred to as shared memory.
Softmax	A mathematical function that converts a vector of real numbers into a probability distribution. In attention, it is applied row-wise to the S matrix to create the P matrix.
Streaming Multiprocessor (SM)	The core processing units of an Nvidia GPU. A GPU contains many SMs (e.g., 108 on an A100), and thread blocks are scheduled to run on them.
Tensor Cores	Specialized hardware units on modern Nvidia GPUs designed to accelerate matrix multiply operations, particularly in low-precision formats like FP16/BF16.

Thread Block	A group of threads that execute a kernel. Threads within a block can communicate via fast shared memory (SRAM).
Tiling	A classical technique of breaking down a large computation into smaller blocks or "tiles." FlashAttention uses tiling to load blocks of the Q, K, and V matrices into SRAM to reduce HBM I/O.
Transformer	A neural network architecture based on the self-attention mechanism, which has become the standard for large-scale language models and other sequence-to-sequence tasks.
Warp	A group of 32 threads within a thread block. Threads within a warp execute in lockstep and can communicate using very fast shuffle instructions.