

IntelliPaper: Virtual Assistant for Research Papers

Select Element

2307.08691v1.pdf

Drag and drop PDF here or click to upload

Generate Summary

0:00 / 0:00

Chat History

Conversation 1

New Chat Delete Chat

Type a message...

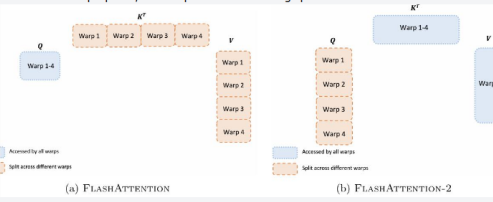
Send

Tell me about FlashAttention

FlashAttention and FlashAttention-2 are algorithms that operate by blocks, which allows for the understanding of long books, reports, high resolution images, audio and video. These algorithms will also accelerate training, fine-tuning, and inference of existing models. Despite context length increasing even more, FlashAttention is still not as efficient as other models. For any blocks where all data can be processed together, the performance may vary depending on specific use cases or implementations.

In terms of correctness, runtime, and memory requirement, Algorithm 1 returns with FlashAttention-2's forward pass described in Algorithm 1. As an immediate next step to making FlashAttention widely applicable in different kinds of devices (e.g., H1 cuDA GPUs, AMD GPUs), as well as new data types such as FP8, the plan is to collaborate with researchers and engineers.

For visualization purposes, an example of a related image path can be:



249dc39e-f0b2-49...

1 / 14

68%

+

-

249dc39e-f0b2-49...

17 Jul 2023

FLASHATTENTION-2: Faster Attention with Better Parallelism and Work Partitioning

Trn Dao^{1,2}

¹Department of Computer Science, Princeton University

²Department of Computer Science, Stanford University

trn@cs.stanford.edu

July 18, 2023

Abstract

Scaling Transformers to longer sequence lengths has been a major problem in the last several years, prompting us to improve performance in language modeling and high-resolution image understanding, as well as to make new applications in code, audio, and video generation. The attention layer is the most bottleneck in scaling to longer sequences, as its runtime and memory requirement grow quadratically in the sequence length. FlashAttention [1] exploits the approximate GPU memory hierarchy to bring significant memory saving, these natural of parallelism and runtime saving is 10x compared to optimized baselines, with no approximation. However, FlashAttention is still not nearly as fast as optimized matrix-multiply (GEMM) operations, reaching only 15-40% of the theoretical maximum FLOPs. We observe that the inefficiency is due to sub-optimal work partitioning between different thread blocks and warps on the GPU, causing either low-occupancy or unnecessary shared memory reads/writes. We propose FlashAttention-2, with better work partitioning to address these issues. In particular, we (1) break the attention to reduce the number of non-optimal FLOPs, (2) parallelize the attention computation, (3) distribute the work between warps to reduce communication through shared memory. These yield around 2x speedup compared to FlashAttention, reaching 50-70% of the theoretical maximum FLOPs, on an A100 and getting close to the efficiency of GEMM operations. We empirically validate that when used end-to-end in text GPT-like models, FlashAttention-2 makes training speed up to 10x (100% vs 10% FLOPs) per A100 GPU (75% model FLOPs utilization).¹

1 Introduction

Scaling up the context length of Transformers [18] is a challenge, since the attention layer at their heart has runtime and memory requirements quadratic in the input sequence length. Ideally, we would like to go beyond the standard 1K sequence length limit to train models to understand books, high-resolution images, and long-form videos. Just within the last year, there have been several language models with much longer context than before: GPT-4 [11] with context length 128k, Gemini-1.5 [10] with context length 1M, and Anthropic's Claude with context length 100k. Emerging use cases such as long document querying and story writing have demonstrated a need for models with such long context.

To reduce the computational requirement of attention on such long context, there have been numerous methods proposed to approximate attention [2, 3, 4, 5, 6, 14, 15, 20]. Though these methods have seen some use cases, as far as we know, most long-scale training runs still use standard attention. Motivated by this, Dao et al. [1] proposed to reorder the attention computation and leverage classical techniques (tiling, recomputation) to significantly speed it up and reduce memory usage from quadratic to linear in sequence length. This yields 2-4x wall-clock time speedup over optimized baselines, up to 10-20x memory saving.

¹FlashAttention-2 is available at <https://github.com/Dao-AILab/flash-attention>.

1

with an approximation, and as a result FlashAttention has seen wide adoption in large-scale training and inference of Transformers.

However, context length increases even more. FlashAttention is still not nearly as efficient as other primitives such as matrix-multiply (GEMM). In particular, while FlashAttention is already 2-4x faster

Quickly gain deep understanding on any subject. Manage thousands of research papers effortlessly with our advanced Q&A

IntelliPaper streamlines the process of exploring academic papers, allowing you to quickly grasp essential insights and gain a deep understanding of any subject. Manage and access thousands of papers effortlessly with our advanced voice enabled Q&A chatbot interface. It integrates seamlessly advanced LLMs on-premise, leveraging the power of AMD Radeon W7900 GPUs to deliver fast, accurate, and insightful responses. The tool also generates an audio podcast of the summary of the current research paper in the display. Revolutionize your research experience with IntelliPaper where efficiency meets depth.

Installation

To install IntelliPaper we require a machine with minimum of 128GB of RAM and an AMD GPU Radeon PRO W7900. This machine need to be installed with many Generative AI libraries and a version of flash-attention package suitable for the AMD GPU Radeon PRO W7900. The following sub-sections presents the steps to install the necessary python packages:

Install PyTorch for 6.2

Create a fresh conda environment with minimal libraries: Python3.10(or higher Python3.11), matplotlib, JupeterLab

After the conda enironment is created the next step is to install the PyTorch Library for ROCm from the [PyTorch portal](#). The pip command to install the latest PyTorch library for ROCm6.2.4 is:

```
pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/rocm6.2
```

Details of pytorch installation can be found from the command:

```
python3 -m torch.utils.collect_env
```

Setup up the following environment variables:

```
export AMDGPU_TARGETS="gfx1100;gfx1101;gfx1102"  
export PYTORCH_ROCM_ARCH=gfx1100  
export GFX_ARCH=gfx1100  
export ROCM_VERSION=6.2  
export GPU_TARGETS=gfx11  
export HIP_VISIBLE_DEVICES=0  
export TORCH_ROCM_AOTRITON_ENABLE_EXPERIMENTAL=1  
export ROCM_PATH=/opt/rocm-6.2.1  
export PATH=$PATH:/opt/rocm-6.2.1/bin
```

Install Optimum-AMD for accessing Huggingface Libraries

For installing advanced LLM models from HuggingFace portal the package Optimum-AMD package is required:

[Find out more about these integrations in the documentation!](#)

```
pip install --upgrade-strategy eager optimum[amd]
```

After installing PyTorch, and Optimum-AMD packages, the system will be ready to deploy advanced LLM models through HuggingFace and the package Ollama. The latest nightly build of pytorch also include Flash Attention 2 through ROCM version of Triton.

Installing Ollama:

Ollama interface can deploy many LLMs to use in systems with AMD GPUs. The installation of Ollama can be done by running a single command:

```
curl -fsSL https://ollama.com/install.sh | sh
```

The details of Ollama installation and the list of LLMs that can be deployed are given in the link [Ollama](#)

For our work, we deployed the following models: Llama3.1 70B, Azure Phi3 medium(14B),
The model llama3.1 (70B) is deployed in the system using the following command:

```
ollama run llama3.1:70b
```

However, user can deploy various other LLMs which are available through ollama for example: internLM-20b or Phi3.5.

Installing Whisper (medium size)

For installing the Speech-to-Text model Whisper model we followed the instructions [ROCm Whisper](#)

Deploying IntelliPaper

Installing in local directory

1. Clone this repository

```
git clone https://github.com:syed-hamza/PdfReader.git
cd PdfReader
```

2. Install the required Python packages

```
cd app
pip install -r requirements.txt
```

3. Run the application

```
python ./app.py
```

4. Launch the IntelliPaper dashboard

<http://127.1.1.0:5000>

PROF

Installing through Docker

IntelliPaper build can be done through Docker using the command

```
docker build -t pdfr .
```

The docker build command will create a container with PyTorch, Optimum-AMD, and IntelliApp code.

The command to run the container is

```
docker run -p 5000:5000 --network="host" -it pdfr
```

IntelliPaper User guide

IntelliPaper dashboard has three panels:

1. Left side panel is uploading a research paper,
2. Middle panel is for viewing the paper summary,

3. Right panel is for browsing the pdf version of the research paper.

Generate Summary button generates the summary of the entire research paper using the Llama 3.1 (70B) model deployed in Ollama framework. The generated Summary is presented in the middle panel. The Audio button presents the narrative of the research paper summary. The yellow marker moves to show the location of the text in the speech.

The middle panel also has a QnA chatbot interface. Here users can ask questions regarding the uploaded research paper. For example, user can query "What is Flash Attention 2?". This chat interface is also can be operated using voice comand.

Contributing

If you have any suggestions or find any issues, please create an issue or submit a pull request.

License

This project is licensed under the MIT License. See the [LICENSE](#) file for details.

Acknowledgements

Feel free to reach out if you have any questions or need further assistance!