Ablation study of Flash.it

Obozov M.A.

March 2024

1 Abstract

This document provides "Ablation study" and full information about methods, benchmarks that were used to built Flash.it project, e.g. flash-formers framework which is main backbone of Flash.it.

2 Introduction

In flash-formers framework we do optimizations on 5 levels:

1. Lowest level. Resources choice.
2. Lower level. Kernel/Inference platform choice.
3. Middle(Model) level. Model choice, model parts choice. Quantization, pruning, distillation.
4. Higher level. Infrastructure level.
5. Highest(MLOps) level.

We will describe each level further. Also we have several non-optimization features, read about them in Appendix.

3 Lowest level

The main feature of lowest level are special LPU(Language Processor Unit) nits. This units were designed specially to run LLMs on them. Flash.it containers can run several popular model families on them:

1. Llama
2. Mixtral
3. Mistral
4. Gemma
5. Qwen[1]

[1]Might be unstable. In beta.

The main focus on LPUs is made when MoE models like Mixtral are used. By our benchmark we found out that it is in **9x times** more efficient then standard inference. For benchmark we took standard Nvidia A100 and LPUv1 unit. Model that was tested is Mixtral 8x7B. To measure acceleration we used Latency(time to provide first chunk) to Tokens/s. Results are still sustainable of more then 8 Experts.

4 Lower level

The one of the most important level of optimizations is kernel level. On this level main optimizations are related to:

1. L2 Cache
2. Shared Memory
3. GPU Utilization
4. Optimal CUDA calls

And more. Flash.it combines all SOTA ways to optimize your model and provide it's own solution - **FAInference**. Here as benchmark we created LLama container and we used same envirement that was used to LPU test. We compared:

1. PowerInfer
2. FAInference
3. llama.cpp

As result both FAInference and PowerInfer got $\tilde{1}$1.0x speedup over llama.cpp. Here is important to note that if we include more factors, comparison of PowerInfer and llama.cpp will be not fair as llama.cpp provide vast more models supported. Here we can note that similar result are consequence of similar optimizations, but we need to notice that FAInference provide other optimizations that can be useful on many cases.

5 Model level

On the model level we provide several optimizations that are touching the model in different ways:

1. Quantization: AQLM, QuIP, SmoothQuant
2. Pruning: Online/Offline
3. Distillation

And extra level:
4. Re-finetune

Points 1,2 and 3 are obviously SOTA by many open-sourced benchmarks. The only exception is online pruning and point 4. We will discuss them in this section.

The Lottery Ticket Hypothesis. *A randomly-initialized, dense neural network contains a subnet- work that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.*

Easiest case: More formally, consider a dense feed-forward $f(x, \theta)$ with initial parameters $\theta \sim D_\theta$. When optimizing with stochastic gradient descent (SGD) on a training set, f reaches minimum validation loss l at iteration j with test accuracy a. In addition, consider training $f(x; m_\theta)$ with a mask $m \in [0, 1]^{|\theta|}$on its parameters such that its initialization is m 0. When optimizing with SGD on the same training set (with m fixed), f reaches minimum validation loss l at iteration j with test accuracy a. The lottery ticket hypothesis predicts that $\exists m$ for which $j \leq j$ (commensurate training time), aa (commensurate accuracy), and $||m||_0 << |\theta|$ (fewer parameters). We find that a standard pruning technique automatically uncovers such trainable subnetworks from fully-connected and convolutional feed-forward networks. We designate these trainable subnetworks, $f(x; m\theta_0^{'})$, winning tickets, since those that we find have won the initialization lottery with a combination of weights and connections capable of learning. When their parameters are randomly reinitialized $f(x; m\theta)$ where θ' $\theta \sim D_\theta$, our winning tickets no longer match the performance of the original network, offering evidence that these smaller networks do not train effectively unless they are appropriately initialized.

We benchmarked online pruning with maximum 2 % accuracy loss. As the result online pruning got 1.5x speedup over no-pruning model. Here is important to mention that this method can be used in combinations with others, so it is optimal in several cases.

Now we need to discuss point 4 related to "Re-finetuning". Here we need brief introduction to SSMs and Mamba model. By definition SSM is literally:

$\mathrm{h}_t = Ah_{t-1} + Bx_t$
$y_t = Ch_t$

Where x - is input, h - hidden state, y - output and A,B and C are trainable matrices of parameters. Instead of standard attention we calculate SSM on GPU like standard convolution. Subsequently we get O(n) complexity instead of O(n^2).

Our idea was to use Mamba instead of transformer backbone in previous transformer models. For generality we tested this trick on several models and we

averaged the results. Overall we got 2.8x speedup.