Poster: LLM-PQ: Serving LLM on Heterogeneous Clusters with Phase-Aware Partition and Adaptive Quantization

Juntao Zhao[1], Borui Wan[1], Yanghua Peng[2], Haibin Lin[2], Chuan Wu[1]
[1]The University of Hong Kong, China, [2] ByteDance Inc., China
{juntaozh,wanborui,cwu}@cs.hku.hk,{pengyanghua.yanghua,haibin.lin}@bytedance.com

Abstract

The immense sizes of Large-scale language models (LLMs) have led to high resource demand and cost for running the models. Though the models are largely served using uniform high-caliber GPUs nowadays, utilizing a heterogeneous cluster with a mix of available high- and low-capacity GPUs can potentially substantially reduce the serving cost. This paper proposes LLM-PQ, a system that advocates adaptive model quantization and phase-aware partition to improve LLM serving efficiency on heterogeneous GPU clusters. Extensive experiments on production inference workloads demonstrate throughput improvement in inference, showing great advantages over state-of-the-art works.

CCS Concepts: • **Computing methodologies** → **Distributed artificial intelligence**.

Keywords: LM Serving; Heterogenous Cluster; Quantization

1 Introduction

Large-scale language models (LLMs) like GPT3, LLaMA, OPT, and BLOOM [9, 11, 13] have achieved remarkable performance in various AI tasks due to their large model sizes. To cope with the massive size, existing serving frameworks [1, 4] integrate model parallelism techniques, such as tensor-parallelism (TP) and pipeline parallelism (PP), with memory footprint reduction schemes, e.g., quantization or offloading, to lower the resource demands of model serving in a distributed manner. However, practical AI cloud or machine learning (ML) clusters often contain heterogeneous devices, e.g., GPUs of different models purchased at different times. In our production cluster, the utilization rate of other GPUs is much lower than that of A100, which are used intensively for both training and inference of LLMs for the best performance. The commonly adopted TP and PP paradigms partition model operations/layers evenly among the GPUs, resulting in either low utilization of high-capacity GPUs or out-of-memory (OOM) errors on low-memory GPUs for heterogeneous GPUs. The limited studies of models serving on heterogeneous clusters [6] focus on the partition of encoder-based transformer models. However, mainstream LLMs with decoder-only structures contain two phases during inference: prompt processing (prefill) and token generation (decode). The execution time required for each phase, depending on the prompt length and token generation number, varies significantly and this difference is further amplified in a heterogeneous cluster, making the previous partition solutions not suitable. At the same time, when the model is partitioned among heterogeneous GPUs, adopting a single quantization precision across is always suboptimal. Uniform single-precision can select a precision, e.g., INT4, that is suitable for GPUs with lower memory to avoid OOM (Out Of Memory) problems but causes a notable portion of resource waste for those with abundant GPU memory.

LLM-PQ jointly determines the quantization precisions, model layer partition, and hybrid micro-batch sizing strategies, given the LLM, available resources of the heterogeneous cluster, and user-specified model quality targets. We implement LLM-PQ in PyTorch [8] and the code is publicly available at https://github.com/tonyzhao-jt/LLM-PQ.

2 LLM-PQ Design

LLM-PQ consists of an offline assigner and a distributed model inference runtime. The assigner includes a Cost Model that predicts memory and latency consumption based on compute characteristics, and an Indicator Generator that quantifies model perturbation for a specific bit. The optimizer uses these outputs to derive the final execution plan. The distributed runtime incorporates an on-the-fly quantizer to reduce the DRAM requirement of the CPU.

2.1 Phase-Aware Cost Model

Memory Cost Model. The peak memory usage of LLM serving is composed of model weights, the KV cache for all requests, and the peak temporary memory. We determine the former based on the meta-information and the latter using a user-assisted profiling method. **Latency Cost Model.** Computation intensity varies across the prefill and decode

PPoPP '24, March 2–6, 2024, Edinburgh, United Kingdom

ACM ISBN 979-8-4007-0435-2/24/03.
https://doi.org/10.1145/3627535.3638480

phases. We model the execution time of the prefill phase as a function of FLOPs, based on v(batch size), s(prompt length), vs, and vs^2. The decoding phase is dominated by memory access; we hence use the total number of bytes accessed(also called MOPs) to model decoding time, based on parameters $v, v(t+s)$(t is the current generated token number), and $(t+s)$. We profile the execution time of each phase on one decoder layer under different precisions with common prompt lengths and batch sizes. We then use interpolation among the sample points to obtain a linear regression model.

2.2 Variance-based Quantization Indicator

State-of-the-art weight-only quantization of LLMs focuses on linear operators and [2, 5, 7] typically target the minimization of minimum square error of the layer's output. Previous research [3] has used the eigenvalues of the Hessian matrix $\mathbf{H}$ to measure a layer's sensitivity (error term), but it requires computation of Hessian and quantization error ($\|Q(\mathbf{W}) - \mathbf{W}\|_2^2$) concerning different precisions, incurring large computation overhead. We adopt a different approach to describe a layer's sensitivity upon quantization. We consider the round variance of quantization for two widely applied rounding methods, i.e., deterministic and stochastic [12], and derive an upper bound of the output variance introduced by quantization.

Theorem 1. *The variance of a linear operator's output after weight-only quantization using stochastic or deterministic rounding is:*

$$Var[\tilde{\mathbf{W}}\mathbf{X}] = \begin{cases} Var[\mathbf{W}\mathbf{X}] + D_{\mathbf{W}} S_{\mathbf{W}}^2 \frac{1}{4} Var[\mathbf{X}], & Deterministic \\ Var[\mathbf{W}\mathbf{X}] + D_{\mathbf{W}} S_{\mathbf{W}}^2 \frac{1}{6} (\mathbb{E}[\mathbf{X}]^2 + Var[\mathbf{X}]), & Stochastic \end{cases} \quad (1)$$

where $D_{\mathbf{W}}$ is the dimension of model weights $\mathbf{W}$ and $S_{\mathbf{W}}$ is the scaling factor. $\mathbf{X}$ is the calibration input feature.

2.3 Optimizer

We present an iterative algorithm to decide the quantization bitwidth for each decoder layer, micro-batch sizes, and LLM model partition on each device, to strike the best balance between inference latency and model quality degradation. The algorithm explores potential device topology orderings and micro-batch sizes for prefill and decode phases; given a device topology ordering and micro-batch sizes, we solve an integer linear program (ILP) to determine the most suitable bitwidth assignment and layer partition among the devices. We present bitwidth transfer which exchanges layer pairs from straggler to pioneer (e.g. $2 \times$ 4-bit layers with an 8-bit layer) as a heuristic to ensure the searching scalability.

2.4 On-The-Fly Quantizer

To optimize low-caliber GPUs with smaller DRAM facing precision changes, we implement on-the-fly quantized model loading. We partition model weights into module-level weights and overlap disk-to-CPU weight loading, on-GPU model quantization, and CPU-to-GPU memory copy during runtime. This approach reduces DRAM usage for model loading and speeds up recovery in case of failures.

Table 1. Serving performance comparison in clusters. The best results (throughput, PPL) are marked in bold.

Model	Cluster	Scheme	PPL	Latency (s)	Throughput (Token/s)
OPT-30b	1	PipeEdge	10.70	146.40	21.86
		Uniform	10.78	948.90	3.37(0.15×)
		FlexGen	10.70	820.72	3.90(0.18×)
		FlexGen-int8	10.70	309.95	10.32(0.47×)
		LLM-PQ	**10.70**	80.60	**39.70(1.82×)**
OPT-66b	2	PipeEdge	10.34	115.03	27.82
		Uniform	10.50	431.92	7.41(0.27×)
		FlexGen	10.33	279.05	11.47(0.41×)
		FlexGen-int8	11.23	31.11	102.87(0.99×)
		LLM-PQ	**10.31(-0.03)**	68.67	**46.60(1.68×)**
BLOOM-176b	3	PipeEdge	10.97	848.98	3.77
		LLM-PQ	**10.90(-0.07)**	**294.68**	**10.86 (2.88×)**

Table 2. Effectiveness of LLM-PQ's variance indicator. PPL is compared with Random, while × is compared with Hessian.

Model	Cluster	Method	PPL	Overhead (s)
OPT-66b	3	Random	10.33	0
		Hessian	10.33	25625.44
		LLM-PQ	**10.31(-0.02)**	**434.78(58.15×)**

3 Evaluation

We run BLOOM [9] and OPT [13] model families. We evaluate candidate precisions: $BITs = \{3, 4, 8, 16\}$. We compare LLM-PQ with three baselines, (1) *PipeEdge*, where we apply uniform quantization and use PipeEdge [6] for heterogeneous layer partition. (2) *Uniform*, which uses uniform quantization, evenly partitions the model layers among devices, (3) *Offloading*, where we adopt CPU and disk swapping in FlexGen [10]. For (1)(2), we keep lowering the quantization bitwidth from the maximum (i.e., FP16) until the model can fit into the devices or has no more feasible bitwidth. For (1)(3), we use the same micro-batch size for prefill and decode phases by partitioning the global batch size by the number of pipeline stages. We make cluster 1: 3xT4-16G + 1xV100-32G, OPT-30b; cluster 2: 2xV100-32G + 2xA100-40G, OPT-66b; cluster 3: 4xV100-32G + 2xA800-80G, BLOOM-176b. GPUs of the same type are located on the same node, intra-connected with NV-LINK; Nodes in the Clusters are interconnected with 800Gbps Ethernet; All GPUs are equipped with TBs disk, with GB/s SSD. We run batched processing with prompt length 512, and token generation number 100.

Experiments in Table 1 show that LLM-PQ achieves the highest throughputs and the best model accuracy. Table 2 demonstrates the superior performance of our indicator compared to Hessian and random methods while maintaining a lower overhead in statistics data collection.

Acknowledgments

This work was supported in part by Hong Kong RGC grants (17208920, 17204423, and C7004-22G).

References

[1] Reza Yazdani Aminabadi, Samyam Rajbhandari, Minjia Zhang, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Jeff Rasley, Shaden Smith, Olatunji Ruwase, and Yuxiong He. 2022. DeepSpeed- Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale. *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis* (2022), 1–15.

[2] Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. 2023. SpQR: A Sparse-Quantized Representation for Near-Lossless LLM Weight Compression. *ArXiv* abs/2306.03078 (2023).

[3] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2019. HAWQ: Hessian AWare Quantization of Neural Networks With Mixed-Precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

[4] Hugging Face. n.d.. Text Generation Inference. https://github.com/huggingface/text-generation-inference. Accessed on: July 24, 2023.

[5] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. *ArXiv* abs/2210.17323 (2022).

[6] Yang Hu, Connor Imes, Xuanang Zhao, Souvik Kundu, Peter A. Beerel, Stephen P. Crago, and John Paul Walters. 2021. Pipeline Parallelism for Inference on Heterogeneous Edge Computing. *ArXiv* abs/2110.14895 (2021).

[7] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. 2023. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. *ArXiv* abs/2306.00978 (2023).

[8] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Neural Information Processing Systems*.

[9] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. 2022. Bloom: A 176b-parameter open-access multilingual language model. *ArXiv* abs/2211.05100 (2022).

[10] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU. In *Proceedings of the 40th International Conference on Machine Learning*.

[11] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aur'elien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. *ArXiv* abs/2302.13971 (2023).

[12] Borui Wan, Jun Zhao, and Chuan Wu. 2023. Adaptive Message Quantization and Parallelization for Distributed Full-graph GNN Training. *ArXiv* abs/2306.01381 (2023).

[13] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: Open Pre-trained Transformer Language Models. *ArXiv* abs/2205.01068 (2022).