AI and Memory Wall

Amir Gholami[1,2] Zhewei Yao[1] Sehoon Kim[1] Coleman Hooper[1] Michael W. Mahoney[1,2,3] Kurt Keutzer[1]

[1]University of California, Berkeley [2]ICSI [3]LBNL

***Abstract*—The availability of unprecedented unsupervised training data, along with neural scaling laws, has resulted in an unprecedented surge in model size and compute requirements for serving/training LLMs. However, the main performance bottleneck is increasingly shifting to memory bandwidth. Over the past 20 years, peak server hardware FLOPS has been scaling at 3.0×/2yrs, outpacing the growth of DRAM and interconnect bandwidth, which have only scaled at 1.6 and 1.4 times every 2 years, respectively. This disparity has made memory, rather than compute, the primary bottleneck in AI applications, particularly in serving. Here, we analyze encoder and decoder Transformer models and show how memory bandwidth can become the dominant bottleneck for decoder models. We argue for a redesign in model architecture, training, and deployment strategies to overcome this memory limitation.**

I. Introduction

The amount of compute needed to train Large Language Models (LLMs) has recently been growing at a rate of 750×/2yrs. This exponential trend has been the main driver for AI accelerators that focus on increasing the peak compute power of hardware, often at the expense of simplifying other parts such as memory hierarchy.

However, these trends miss an emerging challenge with training and serving AI models: memory and communication bottlenecks. In fact, several AI applications are becoming bottlenecked by intra/inter-chip and communication across/to AI accelerators, rather than compute. This is not a new phenomena, and several works in the past observed and warned about this issue. One of the earliest observations of this dates back to 1990 when Ousterhout concluded the following after analyzing the factors impacting operating system's performance [30]:

> "The first hardware-related issue is memory bandwidth: the benchmarks suggest that it is not keeping up with CPU speed ... If memory bandwidth does not improve dramatically in future machines, some classes of applications may be limited by memory performance."

Later in 1995, William Wulf and Sally Mckee further echoed this prediction and coined the term "memory wall". Their argument for this followed a simple but elegant reasoning. The time to complete an operation is dependent on how fast we can perform the arithmetic as well as how fast we can feed data to the arithmetic units of hardware.[1] In the simplest case, the data is either available in the cache, or needs to be fetched from DRAM. With this assumption, even if 80% of data is readily available in cache, and only 20% needs to be fetched from DRAM, the time to complete the operation will be completely limited by DRAM if it takes more than 5 cycles to fetch the 20% cache-miss data from it. This means that no matter how fast the hardware could perform arithmetics per second, the problem will be entirely limited by DRAM bandwidth. They predicted that the diverging speed of improvement of how fast computations can be performed versus how fast data can be fetched is going to create a "memory wall" issue [25, 42]. Based on this they concluded:

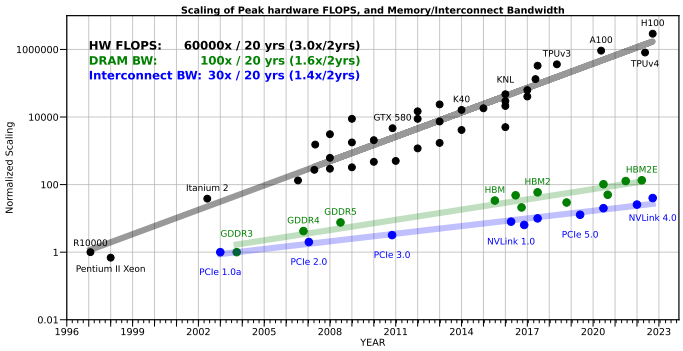


Fig. 1: *The scaling of the bandwidth of different generations of interconnections and memory, as well as the Peak FLOPS. As can be seen, the bandwidth is increasing very slowly. We are normalizing hardware peak FLOPS with the R10000 system, as it was used to report the cost of training LeNet-5 [22].*

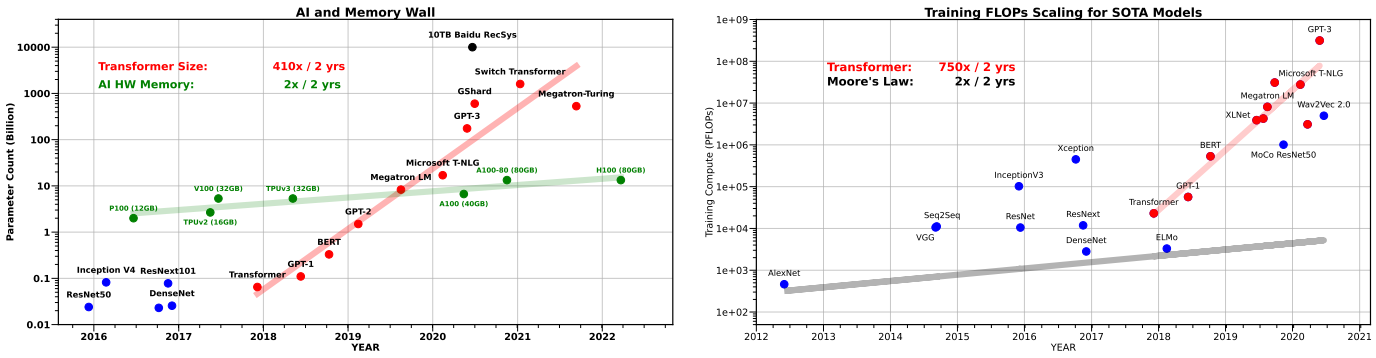
> "Each is improving exponentially, but the exponent for microprocessors is substantially larger than that for DRAMs. The difference between diverging exponentials also grows exponentially."

Several later works also reported a similar observation [12, 24, 25, 31, 36, 40].

In this work, we re-examined this trend by studying more recent data, with a particular focus on hardware used to train AI models, as well as the characteristics of the computations used for training/serving them. 30 years after, the above observations and predictions could not be further correct. Despite many innovations in memory technology, the trend shows that the "memory wall" is increasingly becoming the dominant bottleneck for a range of AI tasks.

We first start by analyzing how peak compute of server-

[1]Just for reference a better way to analyze this is through arithmetic intensity proposed by Sammuel Williams [41] which will be discussed in Sec.II-A.



(a) **(b)**

Fig. 2: *(a) The evolution of the number of parameters of state-of-the-art (SOTA) models over the years, along with the AI accelerator memory capacity (green dots). The number of parameters in large Transformer models has been exponentially increasing with a factor of 410× every two years, while the single GPU memory has only been scaled at a rate of 2× every 2 years. The growth rate for the Transformer models is calculated by only considering the non-recommendation system models (red circles), and the GPU memory is plotted by dividing the corresponding memory size by 6 as an approximate upper bound for the largest model that can be trained with the corresponding capacity. (b) The amount of compute, measured in Peta FLOPs, needed to train SOTA models, for different computer vision (CV), natural language processing (NLP), and Speech models, along with the different scaling of Transformer models (750×/2yrs).*[2]

grade AI hardware has changed since 1998 when Yann Lecun trained the famous Lenet-5 model on MNIST data [22]. We can see that the peak compute of the hardware has increased by 60,000× over the past 20 years, as opposed to a 100× increase for DRAM or a 30× increase for interconnect bandwidth.

The memory wall problem involves both the limited capacity, the bandwidth of memory transfer, as well as its latency (which has been even harder to improve [32] than bandwidth). This entails different levels of memory data transfer. For example, data transfer between compute logic and on-chip memory, or between compute logic and DRAM memory, or across different processors on different sockets. In all these cases, the capacity and the speed of data transfer has been significantly lagging behind hardware compute capabilities.

Now, if we study the trend of recent AI models, and in particular LLMs, we notice that practitioners, motivated by neural scaling law [14], have been scaling the amount of data, model size, and compute needed to train recent models at unprecedented levels. Even though compute / floating-point operations (FLOPs)[3] needed to train these recent models has increased by a factor of 750×/2yrs in the 2018-2022 time frame (see Figure 2), compute is not necessarily the bottleneck, especially for model serving.

First, the LLM sizes have scaled at a rate of 410×/2yrs in that time frame, exceeding memory available on single chip. One might hope that we can use distributed-memory parallelism by scaling-out the training/serving to multiple accelerators to avoid the single hardware's limited memory capacity and bandwidth. However, distributing the work over multiple processes can also face the memory wall problem: the communication bottleneck of moving data between neural network (NN) accelerators, which is even slower and less efficient than on-chip data movement. Similar to the single system memory case, we have not been able to overcome the technological challenges to scale the network bandwidth.

Second, even when the model fits within a single chip, intra-chip memory transfers from/to registers, L2 cache, global memory, etc. are still increasingly becoming the bottleneck. Thanks to the recent advancements in specialized compute units, such as Tensor cores, the arithmetic operations for a large set of computations can finish in a few cycles. Therefore, to keep these arithmetic units utilized at all times one needs to rapidly feed them large amounts of data, and that is where the chip memory bandwidth becomes the bottleneck.

As one can see in Figure 1, over the past 20 years, peak server hardware FLOPS has been scaling at 3.0×/2yrs, outpacing the growth of DRAM and interconnect bandwidth, which have only scaled at 1.6 and 1.4 times every 2 years, respectively. This disparity has made memory, rather than compute, increasingly become a bottleneck, even for cases when the model can fit within a single chip.

Next, we perform a detailed case study for Transformers which helps better showcase the interplay between FLOPs, Memory Operations (MOPs), and end-to-end runtime by considering common models used today.

[3]Please note that we use FLOPs with lowercase s to denote the number of floating point operations needed to perform a task, and FLOPS with capital S to denote the rate at which a given hardware can perform floating point operations per second.

[3]We are specifically not including the cost of training reinforcement learning models in this figure, as the training cost is mostly related to the simulation environment, and there is currently no consensus on a standard simulation environment. Also note that we report the PFLOPs required to train each model to avoid using any approximation for hardware deployment utilization, as the latter depends on the specific library and the hardware used. Finally, all the rates in this document have been computed by solving a linear regression to fit the data shown in each graph.

II. Case Study

In this section, we first outline the run-time characteristics and the performance bottleneck associated with Transformer inference. We examine two different variations of the Transformer architecture: the encoder architecture (e.g., BERT [7]), which concurrently processes all tokens, and the decoder architecture (e.g., GPT [3, 33]), which runs auto-regressively to process and generate one token at each iteration.

A. *Arithmetic Intensity*

A popular method for measuring performance bottlenecks is to compute the total number of FLOPs required to compute the Transformer encoder-only and decoder-only models. However, this metric in isolation can be very misleading. Instead, one needs to study the arithmetic intensity of the operations involved. Arithmetic intensity is the number of FLOPs that can be performed per byte loaded from memory. It can be computed by dividing the total number of FLOPs by the total number of bytes accessed (also referred to as MOPs, or memory operations) [41][4]:

$$\text{Arithmetic Intensity} = \frac{\text{\# FLOPs}}{\text{\# MOPs}}. \quad (1)$$

To illustrate the importance of considering Arithmetic Intensity, we studied BERT-Base and BERT-Large [7], along with GPT-2 [33]. The first two are encoder models, which involve matrix-matrix operations for their inference, and the latter is a decoder/auto-regressive model, where its inference involves repeated matrix-vector multiplications.

B. *Profiling*

To analyze the bottlenecks in Transformer workloads on commodity hardware, we profiled Transformer inference on an Intel Gold 6242 CPU. Figure 3 shows the total FLOPs, MOPs, Arithmetic Intensity, and the final latency of these models for different sequence lengths.[5] It is evident that the GPT-2 latency is significantly longer than the latency for either BERT-Base or BERT-Large for each sequence length, even though BERT-Base and GPT-2 have largely the same model configuration and end-to-end FLOPs (as is depicted in Figure 3a). This is due to the higher memory operations and lower arithmetic intensity of matrix-vector operations inherent in the auto-regressive inference of GPT (see Figure 3c). A model with higher arithmetic intensity can run faster with the same or possibly even more FLOPs than a model with lower arithmetic intensity. This clearly shows how the memory wall can become a major bottleneck for decoder models (at low batch sizes) and not compute.[6]

[4]Here, we are assuming that the local memories are large enough to hold both matrices entirely in memory for a given operation, and that the computed arithmetic intensity values therefore serve as an upper bound for the achievable data reuse. We are also counting the multiplication and addition from a MAC operation separately when computing FLOPs.

[5]We assumed that all model parameters and activations are stored in 8-bit precision, and batch size of 1. In the case of the decoder model, we measured the total amount of the FLOPs and MOPs needed to iteratively generate the full sequence of the given length.

III. Promising Solutions for Breaking the Wall

"No exponential can continue forever," [28] and delaying an exponential scaling at the rate of 410×/2yrs is not going to be feasible for long, even for large hyperscalar companies. This, coupled with the increasing gap between compute and bandwidth capability, will soon make it very challenging to train larger models, as the cost will be exponentially higher.

To continue the innovations and break the memory wall, we need to rethink the design of AI models. There are several issues here. First, the current methods for designing AI models are mostly ad-hoc, and/or involve very simple scaling rules. For instance, recent large Transformer models [3, 16, 37] are mostly just a scaled version of almost the same base architecture proposed in the original BERT model [7]. Second, we need to design more data-efficient methods for training AI models. Current NNs require a huge amount of training data and hundreds of thousands of iterations to learn, which is very inefficient. Some might note that it is also different from how human brains learn, which often only requires very few examples per concept/class. Third, the current optimization and training methods need a lot of hyperparameter tuning (such as learning rate, momentum, etc.), which often results in hundreds of trial and error sweeps to find the right hyperparameter setting to train a model successfully. As such, the training cost reported in Figure 2 (b) is only a lower bound of the actual overhead, and the true cost is typically much higher. Fourth, the prohibitive size of the state-of-the-art models makes their deployment for inference very challenging. This is not just restricted to models such as GPT-3. In fact, deploying large recommendation systems that are used by hyperscalar companies is a major challenge. Finally, the design of hardware accelerators has been mainly focused on increasing peak compute with relatively less attention on improving memory-bound workloads. This has made it difficult both to train large models, as well as to explore alternative models, such as Graph NNs which are often bandwidth-bound and cannot efficiently utilize current accelerators.

All of these issues are fundamental problems in machine learning. Here, we briefly discuss recent research (including some of our own) that has targeted the last three items.

A. *Efficient Training Algorithms*

One of the main challenges with training NN models is the need for brute-force hyperparameter tuning. This includes finding the learning rate, its annealing schedule, the number of iterations needed to converge, etc. This adds (much) more overhead for training SOTA models. Many of these problems arise from the first-order SGD methods used for training. While SGD variants are easy to implement, they are not robust

[6]Note that this may not apply to all kinds of applications of decoder models such as in summarizing long documents where the inherent operations for processing the input prompt are matrix-matrix operations. Other cases include large batch size inference which effectively includes matrix-matrix operations.

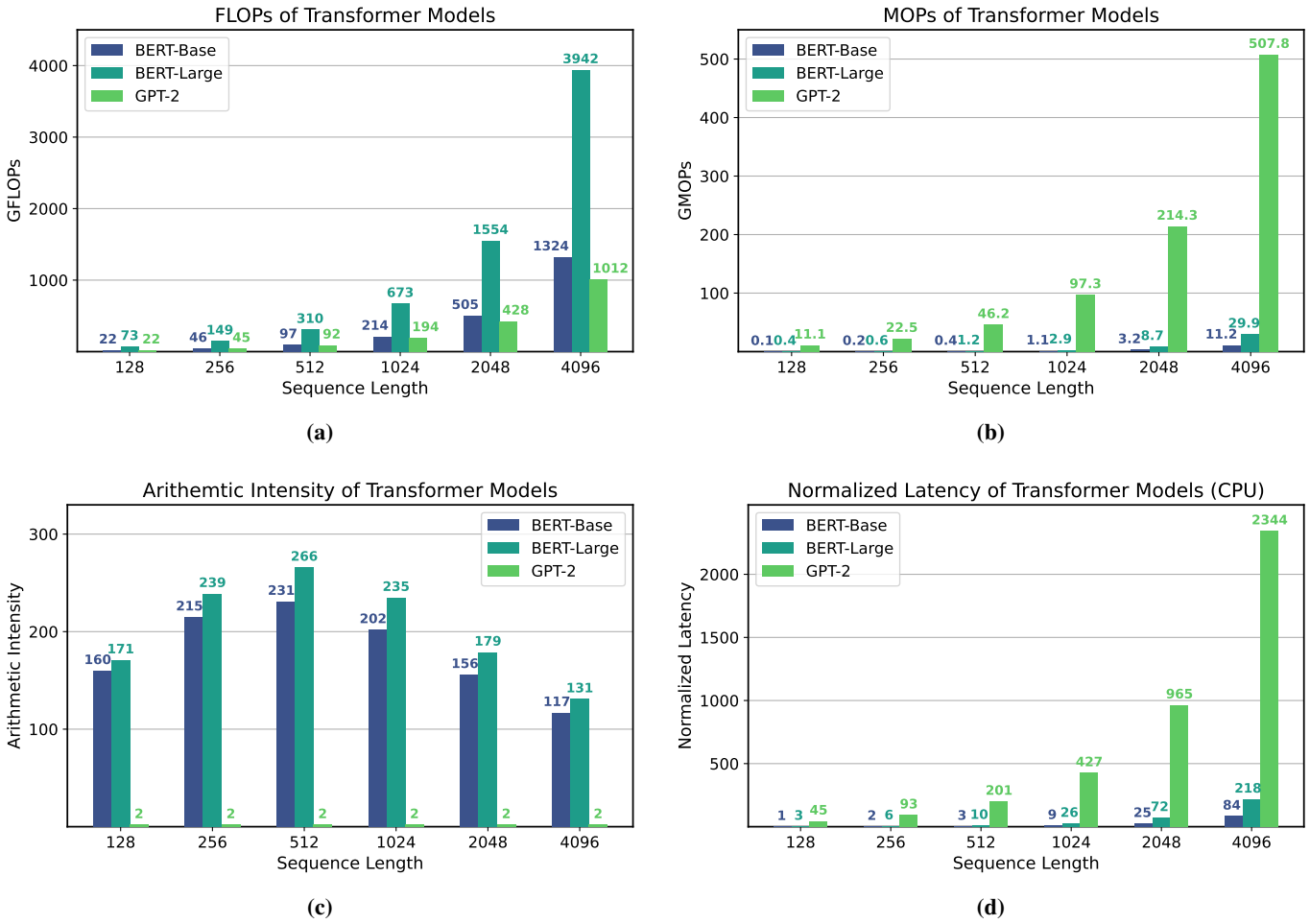


Fig. 3: *Profiling results for BERT-Base, BERT-Large, and GPT-2 models for processing/generating different sequence lengths with batch size 1. (a) total inference FLOPs: notice how encoder models have higher FLOPs; (b) total inference Memory Operations (MOPs): notice how the decoder GPT model has orders of magnitude more MOPs due to its matrix-vector type operations vs matrix-matrix operations in encoder models; (c) arithmetic intensity: notice how GPT-2 has orders of magnitude smaller arithmetic intensity, which makes it very challenging to effectively utilize a given hardware's compute units; (d) end-to-end latency of the different models normalized to the BERT-Base model for processing an input sequence length of 128: notice how the decoder model's runtime is the slowest, despite having smaller FLOPs. See [18] for more details.*

to hyperparameter tuning, and are very hard to tune for new models, for which the right set of hyperparameters is unknown. One promising approach to address this is to use second-order stochastic optimization methods [43]. These methods are typically more robust to hyperparameter tuning, and they can achieve SOTA [43]. However, current methods have 3–4× higher memory footprint, which needs to be addressed [43]. A promising line of work for that is the Zero framework from Microsoft, which showed how one can train 8× bigger models with the same memory capacity by removing/sharding redundant optimization state variables [2, 34]. If the overhead of these higher-order methods could be addressed, then they can significantly reduce the total cost of training large models.

Another promising approach includes reducing the memory footprint and increasing the data locality of optimization algorithms, at the expense of performing more computations. A notable example of this in numerical linear algebra is the family of communication avoiding algorithms [1]. One example of optimizing for memory for NN training is re-materialization, where we only store/checkpoint a subset of activations during the forward pass, instead of saving all activations. This reduces the feature map's memory footprint, as shown in Figure 4. The rest of the activations could then be recomputed when needed [15, 19]. Even though this will increase compute, one can significantly reduce the memory footprint by up to 5× [15], with just 20% more compute. This can also allow practitioners to train large models on single-chip memory rather than utilize distributed training, which is often difficult to set up (outside of major hyperscaler companies) and is hard to debug (for non-expert developers). Interestingly, traditional trends show that new NN model architectures have been developed based on what researchers have access to

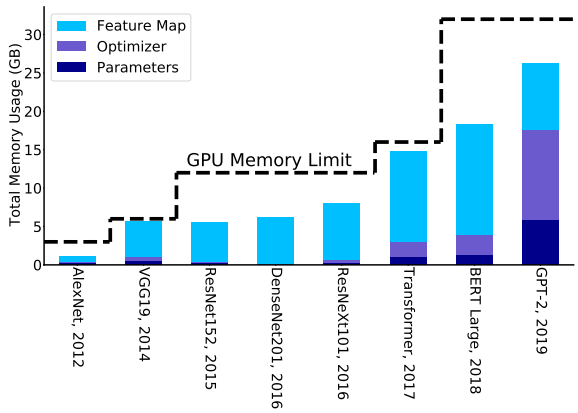


Fig. 4: *The amount of memory required to train different NN models. Here, the optimizer used for CV models is SGD+Momentum, and for NLP models it is ADAM. There is an interesting trend in discovering/designing new models, based on the available GPU memory size. Every time the GPU memory capacity is increased, data scientists have designed newer models. As such, breaking this so-called GPU memory wall could further allow new innovations. See [15] for more details on checkpointing.*

within a single chip rather than using complicated distributed memory methods see Figure 4 Of course there are many counter examples for this coming from big hyperscalars that have dedicated teams to support researchers to deploy large models, but such examples are limited when we consider the entire community. In fact, even with recent LLMs there is often large efforts in compressing the models so that they fit within on system to make the model accessible to a larger community of researchers.

Another important solution is to design optimization algorithms that are robust to low-precision training. In fact, one of the major breakthroughs in AI accelerators has been the use of half-precision (FP16) arithmetic, instead of single precision [11, 26]. This has enabled more than a 10× increase in hardware compute capability. However, it has been challenging to reduce the precision further, from half-precision to INT8, without accuracy degradation with current optimization methods. A recent promising trend is to use a mixture of FP8 and FP16 (and even most recently FP4) [27]. Algorithmic innovations in this area will certainly enable us to more efficiently utilize the hardware, and could allow more areas of the chip to be used to improve memory (which is commonly referred to as memory-gap penalty [31]).

B. Efficient Deployment

Deploying recent SOTA models [4, 5, 16, 37] such as GPT-3 [3] or large recommendation systems [29] is quite challenging, as they require distributed-memory deployment for inference. One promising solution to address this is to compress these models for inference, either by reducing the precision (i.e., quantization), removing (i.e., pruning) their redundant parameters, or design small language models [35].

The first approach, quantization, can be applied at the training and/or inference steps. While it has been very challenging to reduce the training precision much below FP16, it is possible to use ultra-low precision for inference. With current methods, it is relatively easy to quantize inference down to INT4 precision, with minimal impact on accuracy [6, 9, 17, 23, 44]. This results in up to 8× reduction in model footprint and latency [10]. However, inference with sub-INT4 precision is more challenging, and it is currently a very active area of research.

The second approach, pruning, completely removes/prunes redundant parameters in the model. With current methods, it is possible to prune up to 30% of neurons with structured sparsity, and up to 80% with unstructured sparsity, with minimal impact on accuracy [8, 13, 21]. Pushing beyond this limit, however, is very challenging, and it often results in fatal accuracy degradation. Resolving this is an open problem.

The third approach, small language models, could open up completely new frontiers and enable widespread adoption of AI. Interestingly, the models used for LLMs has not changed since the Transformer model was introduced in 2017 [38]. What has worked so far is to scale the data and size of the models, which has led to the "emergent capabilities" of these models [3, 39]. However, recent work on small language models has shown promising results [35] on their abilities. If a model could fit completely on-chip, then that can result in orders of magnitude speedup and energy savings.

C. Rethinking the Design of AI Accelerators

There are fundamental challenges in increasing both the memory bandwidth and the peak compute capability of a chip at the same time [32]. However, it is possible to sacrifice peak compute to achieve better compute/bandwidth trade-offs. In fact, the CPU architecture already incorporates a well-optimized cache hierarchy. This is why CPUs have much better performance than GPUs for bandwidth-bound problems. Such problems include large recommendation problems [29]. However, the main challenge with today's CPUs is that their peak compute capability (i.e., FLOPS) is about an order of magnitude less than AI accelerators such as GPUs or TPUs. One reason for this is that AI accelerators have mainly been designed to achieve maximum peak compute. This often requires removing components such as cache hierarchy in favor of adding more compute logic. One could imagine an alternative architecture in between these two extremes, preferably with more efficient caching, and importantly with higher capacity DRAM (possibly a hierarchy of DRAMs with different bandwidths). The latter could be very helpful in mitigating the distributed-memory communication bottlenecks [20].

IV. Conclusion

The computational cost of training recent SOTA Transformer models in NLP has been scaling at a rate of 750×/2yrs, and the model parameter size has been scaling at 410×/2yrs. In contrast, the peak hardware FLOPS has been scaling at

a rate of 3.0×/2yrs, while both the DRAM and interconnect bandwidth have been increasingly falling behind, with a scaling rate of 1.6×/2yrs and 1.4×/2yrs, respectively. To put these numbers into perspective, peak hardware FLOPS has increased by 60,000× over the past 20 years, while DRAM/Interconnect bandwidth has only scaled by a factor of 100×/30× over the same time period, respectively. With these trends, memory — in particular, intra/inter-chip memory transfer — will soon become the main limiting factoring in serving large AI models. As such, we need to rethink the training, deployment, and design of AI models as well as how we design AI hardware to deal with this increasingly challenging memory wall.

ACKNOWLEDGEMENTS

We would like to thank Suresh Krishna, and Aniruddha Nrusimha for their valuable feedback. We acknowledge gracious support from Furiosa team. We also appreciate the support from Microsoft through their Accelerating Foundation Model Research, including great support from Sean Kuno. Furthermore, we appreciate support from Google Cloud, the Google TRC team, and specifically Jonathan Caton, and Prof. David Patterson. Prof. Keutzer's lab is sponsored by the Intel corporation, Intel One-API, Intel VLAB team, the Intel One-API center of excellence, Apple, Samsung, Panasonic, as well as funding through BDD and BAIR. We appreciate great feedback and support from Ellick Chan, Saurabh Tangri, Andres Rodriguez, and Kittur Ganesh. Sehoon Kim would like to acknowledge the support from the Korea Foundation for Advanced Studies (KFAS). Amir Gholami was supported through funding from Samsung SAIT. Michael W. Mahoney would also like to acknowledge a J. P. Morgan Chase Faculty Research Award as well as the DOE, NSF, and ONR. Our conclusions do not necessarily reflect the position or the policy of our sponsors, and no official endorsement should be inferred.

REFERENCES

[1] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, "Minimizing communication in numerical linear algebra," *SIAM Journal on Matrix Analysis and Applications*, vol. 32, no. 3, pp. 866–901, 2011.

[2] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Review*, vol. 60, no. 2, pp. 223–311, 2018.

[3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[4] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann *et al.*, "Palm: Scaling language modeling with pathways," *Journal of Machine Learning Research*, vol. 24, no. 240, pp. 1–113, 2023.

[5] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, Y. Li, X. Wang, M. Dehghani, S. Brahma *et al.*, "Scaling instruction-finetuned language models," *arXiv preprint arXiv:2210.11416*, 2022.

[6] T. Dettmers, R. Svirschevski, V. Egiazarian, D. Kuznedelev, E. Frantar, S. Ashkboos, A. Borzunov, T. Hoefler, and D. Alistarh, "Spqr: A sparse-quantized representation for near-lossless llm weight compression," *arXiv preprint arXiv:2306.03078*, 2023.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019, pp. 4171–4186.

[8] E. Frantar and D. Alistarh, "Sparsegpt: Massive language models can be accurately pruned in one-shot," 2023.

[9] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "OPTQ: Accurate quantization for generative pre-trained transformers," in *The Eleventh International Conference on Learning Representations*, 2023.

[10] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," in *Low-Power Computer Vision*. Chapman and Hall/CRC, 2022, pp. 291–326.

[11] B. Ginsburg, S. Nikolaev, A. Kiswani, H. Wu, A. Gholaminejad, S. Kierat, M. Houston, and A. Fit-Florea, "Tensor processing using low precision format," Dec. 28 2017, uS Patent App. 15/624,577.

[12] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[13] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, "Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks," *The Journal of Machine Learning Research*, vol. 22, no. 1, pp. 10 882–11 005, 2021.

[14] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, O. Vinyals, J. Rae, and L. Sifre, "An empirical analysis of compute-optimal large language model training," in *Advances in Neural Information Processing Systems*, vol. 35, 2022, pp. 30 016–30 030.

[15] P. Jain, A. Jain, A. Nrusimha, A. Gholami, P. Abbeel, J. Gonzalez, K. Keutzer, and I. Stoica, "Checkmate: Breaking the memory wall with optimal tensor rematerialization," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 497–511, 2020.

[16] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier *et al.*, "Mistral 7b," *arXiv preprint arXiv:2310.06825*, 2023.

[17] S. Kim, C. Hooper, A. Gholami, Z. Dong, X. Li, S. Shen, M. W. Mahoney, and K. Keutzer, "Squeezellm:

Dense-and-sparse quantization," *arXiv preprint arXiv:2306.07629*, 2023.

[18] S. Kim, C. Hooper, T. Wattanawong, M. Kang, R. Yan, H. Genc, G. Dinh, Q. Huang, K. Keutzer, M. W. Mahoney, Y. S. Shao, and A. Gholami, "Full stack optimization of transformer inference: a survey," *Workshop on Architecture and System Support for Transformer Models (ASSYST) at ISCA*, 2023.

[19] V. A. Korthikanti, J. Casper, S. Lym, L. McAfee, M. Andersch, M. Shoeybi, and B. Catanzaro, "Reducing activation recomputation in large transformer models," *Proceedings of Machine Learning and Systems*, vol. 5, 2023.

[20] S. Krishna and R. Krishna, "Accelerating recommender systems via hardware scale-in," *arXiv preprint arXiv:2009.05230*, 2020.

[21] W. Kwon, S. Kim, M. W. Mahoney, J. Hassoun, K. Keutzer, and A. Gholami, "A fast post-training pruning framework for transformers," in *Advances in Neural Information Processing Systems*, vol. 35. Curran Associates, Inc., 2022, pp. 24 101–24 116.

[22] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[23] J. Lin, J. Tang, H. Tang, S. Yang, X. Dang, C. Gan, and S. Han, "Awq: Activation-aware weight quantization for llm compression and acceleration," 2023.

[24] J. McCalpin, "Stream: Sustainable memory bandwidth in high performance computers," *http://www. cs. virginia. edu/stream/*, 2006.

[25] S. A. McKee, "Reflections on the memory wall," in *Proceedings of the 1st Conference on Computing Frontiers*, ser. CF '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 162.

[26] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed precision training," in *International Conference on Learning Representations*, 2018.

[27] P. Micikevicius, D. Stosic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, S. Ha, A. Heinecke, P. Judd, J. Kamalu *et al.*, "Fp8 formats for deep learning," *arXiv preprint arXiv:2209.05433*, 2022.

[28] G. Moore, "No exponential is forever: but "forever" can be delayed! [semiconductor industry]," in *2003 IEEE International Solid-State Circuits Conference, 2003. Digest of Technical Papers. ISSCC.*, 2003, pp. 20–23 vol.1.

[29] M. Naumov, D. Mudigere, H.-J. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C.-J. Wu, A. G. Azzolini *et al.*, "Deep learning recommendation model for personalization and recommendation systems," *arXiv preprint arXiv:1906.00091*, 2019.

[30] J. Ousterhout, "Why aren't operating systems getting faster as fast as hardware?" in *USENIX Summer Conference, 1990*, 1990.

[31] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick, "A case for intelligent ram," *IEEE micro*, vol. 17, no. 2, pp. 34–44, 1997.

[32] D. A. Patterson, "Latency lags bandwith," *Communications of the ACM*, vol. 47, no. 10, pp. 71–75, 2004.

[33] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[34] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: Memory optimizations toward training trillion parameter models," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–16.

[35] T. Schick and H. Schütze, "It's not just size that matters: Small language models are also few-shot learners," *arXiv preprint arXiv:2009.07118*, 2020.

[36] D. Sites, "It's the memory, stupid!" *Microprocessor Report*, pp. 18–24, 1996.

[37] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[38] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[39] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler *et al.*, "Emergent abilities of large language models," *arXiv preprint arXiv:2206.07682*, 2022.

[40] M. V. Wilkes, "The memory wall and the cmos endpoint," *ACM SIGARCH Computer Architecture News*, vol. 23, no. 4, pp. 4–6, 1995.

[41] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[42] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *ACM SIGARCH computer architecture news*, vol. 23, no. 1, pp. 20–24, 1995.

[43] Z. Yao, A. Gholami, S. Shen, M. Mustafa, K. Keutzer, and M. Mahoney, "Adahessian: An adaptive second order optimizer for machine learning," in *proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 12, 2021, pp. 10 665–10 673.

[44] Z. Yao, R. Yazdani Aminabadi, M. Zhang, X. Wu, C. Li, and Y. He, "Zeroquant: Efficient and affordable post-training quantization for large-scale transformers," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 27 168–27 183.