THEME ARTICLE: HOT CHIPS 2023

AI and Memory Wall

Amir Gholami, *UC Berkeley, Berkeley, CA, 94720, USA*

Zhewei Yao, *Snowflake, Bellevue, WA, 98004, USA*

Sehoon Kim, Coleman Hooper, Michael W. Mahoney, and Kurt Keutzer, *UC Berkeley, Berkeley, CA, 94720, USA*

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 large language models. However, the main performance bottleneck is increasingly shifting to memory bandwidth. Over the past 20 years, peak server hardware floating-point operations per second have been scaling at 3.0× per two years, outpacing the growth of dynamic random-access memory and interconnect bandwidth, which have only scaled at 1.6 and 1.4 times every two 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.

The amount of compute needed to train large language models (LLMs) has been growing at a rate of 750× per two years. 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 the memory hierarchy.

However, these trends miss an emerging challenge with training and serving these models: memory and communication bottlenecks. In fact, several AI applications are becoming bottlenecked by intra-/interchip and communication across/to AI accelerators rather than compute. These challenges are commonly referred to as the memory wall problem, a term originally coined by William Wulf and Sally McKee in 1995[11]:

> "Each is improving exponentially, but the exponent for microprocessors is substantially larger than that for DRAMs [dynamic random-access memories]. The difference between diverging exponentials also grows exponentially."

Interestingly, despite many innovations in memory technology, this trend has remained the same until today. This can be seen from Figure 1, where we show how the peak compute of server-grade AI hardware has increased by 60,000× over the past 20 years, as opposed to 100× for DRAM or 30× for interconnect bandwidth. Unfortunately, it has been very difficult to overcome the fundamental challenges of increasing DRAM/interconnect bandwidth.[8]

The memory wall problem involves both the limited capacity and the bandwidth of memory transfer. This entails different levels of memory data transfer—for example, data transfer between compute logic and on-chip memory, between compute logic and DRAM memory, or across different processors on different sockets. For all of these cases, the capacity and the speed of data transfer have been significantly lagging behind hardware compute capabilities.

If we study the trend of recent AI models and, in particular LLMs, we notice that practitioners, motivated by neural scaling law,[5] have been scaling the amount of data, model size, and compute needed to train recent models at unprecedented levels. Even though the compute/floating-point operations per second (FLOPS) needed to train these recent models have increased by a factor of 750× per two years in the 2018–2022 timeframe (Figure 2), compute is not necessarily the bottleneck. First, the LLM sizes have scaled at a rate of 410× per two years in that timeframe as well, exceeding memory available on a single chip. One might hope that we can use distributed-memory parallelism by scaling

0272-1732 © 2024 IEEE
Digital Object Identifier 10.1109/MM.2024.3373763
Date of publication 25 March 2024; date of current version 6 June 2024.

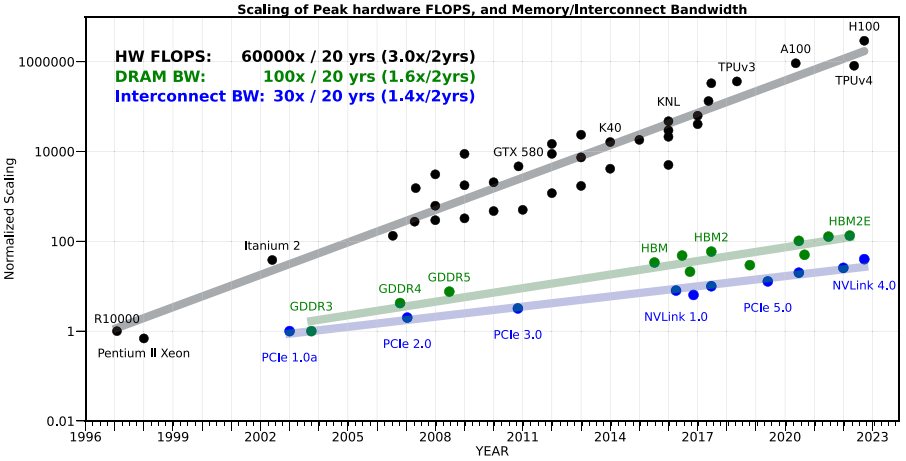
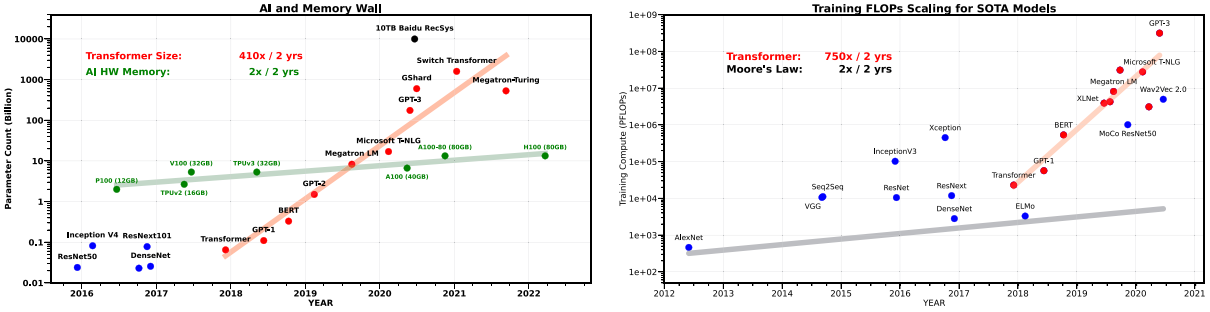


FIGURE 1. The scaling of the bandwidth of different generations of interconnections and memory as well as the peak floating-point operations per second (FLOPS). As can be seen, the bandwidth is increasing very slowly. We are normalizing hardware peak FLOPS with the R10000 system, as it was used to report the cost of training LeNet-5. DRAM: dynamic random-access memory; FLOPS: floating-point operations per second; HW: hardware; BW: bandwidth.



(a) (b)

FIGURE 2. (a) The evolution of the number of parameters of 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 two years. The growth rate for the transformer models is calculated by only considering the nonrecommendation system models (red circles), and the GPU memory is plotted by dividing the corresponding memory size by six as an approximate upper bound for the largest model that can be trained with the corresponding capacity. (b) The amount of compute, measured in petaFLOPs, needed to train SOTA models for different CV, natural language processing (NLP), and speech models along with the different scaling of transformer models (750× per two years).[a] PFLOPs: petaFLOPs; SOTA: state-of-the-art; CV: computer vision.

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 also faces 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, intrachip memory still transfers from/to registers, Level 2 cache, global memory, etc. are increasingly

[a]We are specifically not including the cost of training reinforcement learning models in this graph, 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 of the rates in this document have been computed by solving a linear regression to fit the data shown in each graph.

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 have been scaling at 3.0× per two years, outpacing the growth of DRAM and interconnect bandwidth, which have only scaled at 1.6 and 1.4 times every two 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.

In the following section, we perform a detailed case study for transformers that helps showcase the interplay the between FLOPs, memory operations (MOPs), and end-to-end runtime for different generative and encoder models.

CASE STUDY

In this section, we first outline the runtime characteristics and the performance bottleneck associated with transformer inference. We examine two different variations of the transformer architecture: the encoder architecture (e.g., BERT[2]), which concurrently processes all tokens, and the decoder architecture (e.g., GPT[1]), which runs autoregressively to process and generate one token at each iteration.

Arithmetic Intensity

A popular method for measuring the performance bottleneck is to compute the total number of floating-point operations (FLOPs) required to compute the transformer encoder-only and decoder-only models. However, this metric in isolation can be very misleading. Importantly, one needs to study the arithmetic intensity of the operations involved. Arithmetic intensity is the number of floating-point operations 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)[10,b]:

$$\text{arithmetic intensity} = \frac{\text{number of FLOPs}}{\text{number of MOPs}}. \quad (1)$$

[b]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 multiply-accumulate operation separately when computing FLOPs.

To illustrate the importance of considering arithmetic intensity, we studied BERT-Base and BERT-Large along with GPT-2. The first two are encoder models that involve matrix–matrix operations for their inference, and the last is a decoder/autoregressive model, where its inference involves repeated matrix–vector multiplications.

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 final latency of these models for different sequence lengths.[c] 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 was depicted in Figure 3(a)]. This is due to the higher MOPs and lower arithmetic intensity of matrix–vector operations inherent in the autoregressive inference of GPT [see Figure 3(c)]. 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.[d]

PROMISING SOLUTIONS FOR BREAKING THE WALL

"No exponential can continue forever," and delaying an exponential scaling at the rate of 410× per year 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

[c]We assumed that all model parameters and activations are stored in 8-bit precision and a batch size of one. 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.

[d]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.

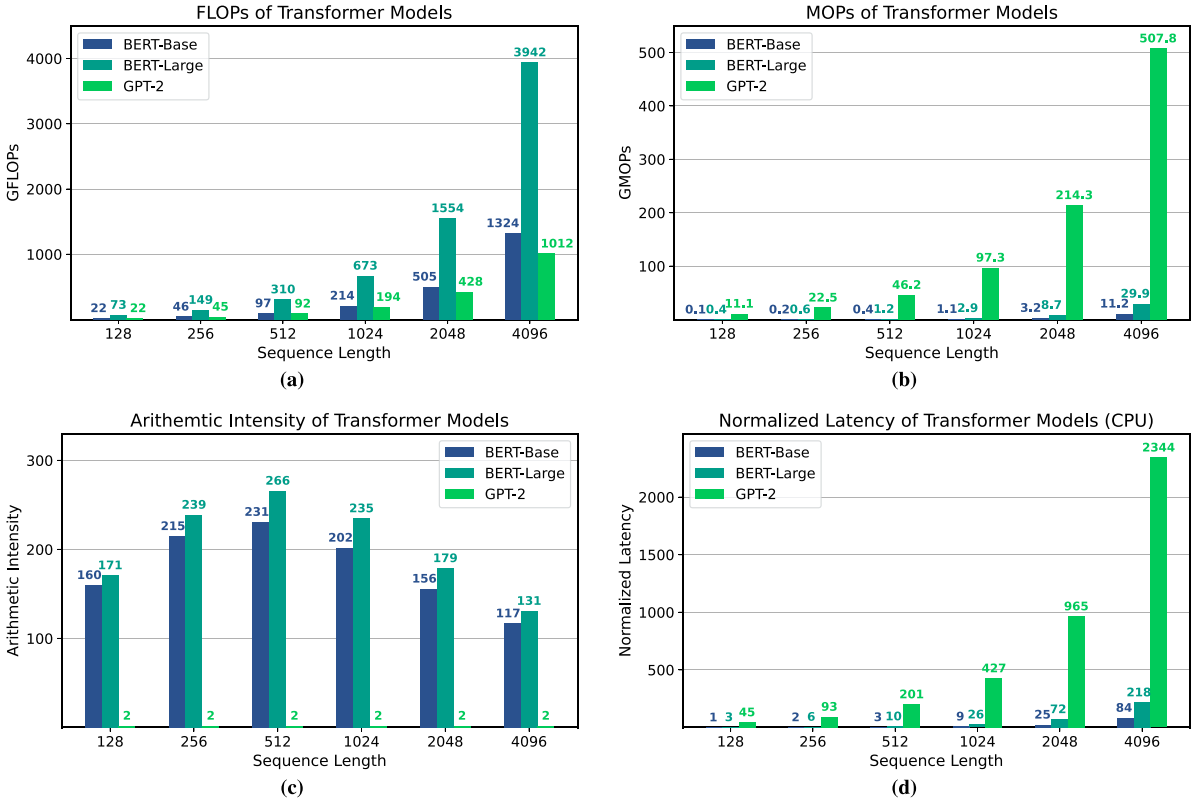


FIGURE 3. Profiling results for BERT-Base, BERT-Large, and GPT-2 models for processing/generating different sequence lengths with a batch size of one. (a) Total inference FLOPs: notice how encoder models have higher FLOPs. (b) Total inference MOPs: notice how the decoder GPT model has orders of magnitude 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 HW'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 Kim et al.[7] for more details. MOPs: memory operations.

transformer models are mostly just a scaled version of almost the same base architecture proposed in the original BERT model.[2] 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 the learning rate, momentum, etc.), which often results in hundreds of trial-and-error sweeps to find the right 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 and 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.

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 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. These methods are typically more robust to hyperparameter tuning, and they can achieve SOTA. However, current methods have 3–4× higher memory footprint, which needs to be addressed.[12] 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.[9] 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. One simple example is to only store/checkpoint a subset of activations during the forward pass, instead of saving all activations, to reduce the feature map's memory footprint, shown in Figure 4. The rest of the activations could then be recomputed when needed. Even though this will increase compute, one can significantly reduce the memory footprint by up to 5× with just 20% more compute.[6] 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 nonexpert developers. In fact, traditional trends have shown that new models have been found by researchers to have a direct trend of the amount of single-chip memory available (see Figure 4).

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. This has enabled more than a 10× increase in hardware compute capability. However, it has been challenging to further reduce the precision from half precision to INT8 without accuracy degradation with current optimization methods.

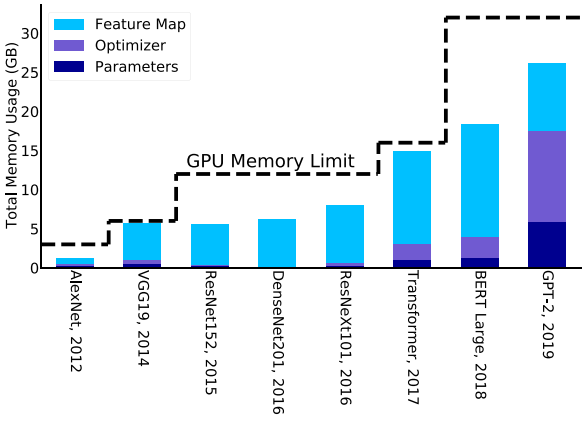


FIGURE 4. The amount of memory required to train different NN models. Here, the optimizer used for CV models is SGD+Momentum, and for NLP models it is ADAM. There is an interesting trend in discovering/designing new models, based on the available GPU memory size. Every time the GPU memory capacity is increased, data scientists have designed newer models. As such, breaking this so-called GPU memory wall could further allow new innovations. See Jain et al.[6] for more details on checkpointing.

Efficient Deployment

Deploying recent SOTA models, such as GPT-3 or large recommendation systems, is quite challenging, as they require distributed-memory deployment for inference. One promising solution to address this is to compress these models for inference by reducing the precision (i.e., quantization) or removing (i.e., pruning) their redundant parameters.

The first approach is quantization, a method that 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 ultralow precision for inference. With current methods, it is relatively easy to quantize inference down to INT4 precision, with minimal impact on accuracy. This results in up to 8× reduction in model footprint and latency.[3] However, inference with sub-INT4 precision is more challenging and is currently a very active area of research.

Another possibility is to completely remove/prune 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.[4] Pushing beyond this limit, however, is very challenging, and it often results in fatal accuracy degradation. Resolving this is an open problem.

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.[8] However, it is possible to sacrifice peak compute to achieve better compute/bandwidth tradeoffs. This is not an impossible task, and, 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. 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 that of AI accelerators, such as GPUs or tensor processing units. 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.

CONCLUSION

The computational cost of training recent SOTA transformer models in natural language processing (NLP) has been scaling at a rate of 750× per two years, and the model parameter size has been scaling at 410× per two years. In contrast, the peak hardware FLOPS is scaling at a rate of 3.0× per two years, while both the DRAM and interconnect bandwidth have been increasingly falling behind, with a scaling rate of 1.6× per two years and 1.4× per two years, 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-/interchip memory transfer—will soon become the main limiting factor 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.

REFERENCES

1. T. B. Brown et al., "Language models are few-shot learners," 2020, *arXiv:2005.14165*.
2. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.
3. 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*, G. K. Thiruvathukal, Y.-H. Lu, J. Kim, Y. Chen, and B. Chen, Eds., London, U.K.: Chapman and Hall, 2022, pp. 291–326.
4. 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," *J. Mach. Learn. Res.*, vol. 22, no. 1, pp. 10,882–11,005, 2021.
5. J. Hoffmann et al., "Training compute-optimal large language models," 2022, *arXiv:2203.15556*.
6. P. Jain et al., "Checkmate: Breaking the memory wall with optimal tensor rematerialization," in *Proc. Mach. Learn. Syst.*, 2020, vol. 2, pp. 497–511.
7. S. Kim et al., "Full stack optimization of transformer inference: a survey," 2023, *arXiv:2302.14017*.
8. D. A. Patterson, "Latency lags bandwith," *Commun. ACM*, vol. 47, no. 10, pp. 71–75, 2004, doi: 10.1145/1022594.1022596.
9. S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: Memory optimizations toward training trillion parameter models," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC)*, Piscataway, NJ, USA: IEEE, 2020, pp. 1–16, doi: 10.1109/SC41405.2020.00024.
10. S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, 2009, doi: 10.1145/1498765.1498785.
11. W. A. Wulf, and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *ACM SIGARCH Comput. Archit. News*, vol. 23, no. 1, pp. 20–24, 1995, doi: 10.1145/216585.216588.
12. Z. Yao, A. Gholami, S. Shen, M. Mustafa, K. Keutzer, and M. Mahoney, "ADAHESSIAN: An adaptive second order optimizer for machine learning," in *Proc. AAAI Conf. Artif. Intell.*, 2021, vol. 35, no. 12, pp. 10,665–10,673, doi: 10.1609/aaai.v35i12.17275.

AMIR GHOLAMI is an assistant research scientist with UC Berkeley, Berkeley, CA, 94720, USA. His research interests include AI systems, machine learning, and high performance computing. Gholami received his Ph.D. degree in computational science from the University of Texas at Austin. Contact him at amirgh@berkeley.edu.

ZHEWEI YAO is a scientist at Snowflake, Bellevue, WA, 98004, USA. His research interests include machine learning, large-scale training and inference, as well as distributed training system. Yao received his Ph.D. degree from UC Berkley. Contact him at zheweiy@berkeley.edu.

SEHOON KIM is a Ph.D. candidate at UC Berkeley, Berkeley, CA, 94720, USA. His research interests include AI systems, efficient AI, and machine learning. Kim received his B.S. degree in electrical and computer engineering from Seoul National University. Contact him at sehoonkim@berkeley.edu.

COLEMAN HOOPER is a Ph.D. student at UC Berkeley, Berkeley, CA, 94720, USA. His research interests include systems and hardware-software co-design for machine learning. Hooper received his B.S. degree in electrical engineering from Harvard University. Contact him at chooper@berkeley.edu.

MICHAEL W. MAHONEY is with the Department of Statistics, UC Berkeley, Berkeley, CA, 94720, USA, and with the International Computer Science Institute (ICSI), Berkeley, CA, 94704, USA. He is an Amazon Scholar as well as head of the Machine Learning and Analytics Group at the Lawrence Berkeley National Laboratory (LBNL), Berkeley, CA, 94720, USA. His research interests include algorithmic and statistical aspects of modern large-scale data analysis. Mahoney received his Ph.D. in physics from Yale University. Contact him at mmahoney@stat.berkeley.edu or https://www.stat.berkeley.edu/~mmahoney/.

KURT KEUTZER is a professor of the Graduate School in the Berkeley AI Research Lab, Department of Electrical Engineering and Computer Science, UC Berkeley, Berkeley, CA, 94720. His research interests include all aspects of making artificial intelligence computationally efficient, from client to cloud, and making systems of large language models efficient. Keutzer received his Ph.D. in computer science from Indiana University. He is a Life Fellow of IEEE. Contact him at keutzer@berkeley.edu.