Fixstars Performance Engineering Series #2

NVIDIA H200: AI Acceleration and Performance Engineering in Practice

August 18, 2025

Naoki Yoshifuji

Fixstars Corporation, Performance Engineering Lab

Abstract

- The NVIDIA H200 has the same computing unit specifications as the NVIDIA H100; the only significant difference is its larger memory capacity. The key to its value lies in how this increased memory is utilized.
- We conducted a performance engineering study, leveraging the H200's increased memory across five key areas: (1) 3D parallelism, (2) micro-batch size, (3) computational precision, (4) activation recomputation, and (5) optimization algorithms. This reduced the time to reach the same model precision by about 60%, resulting in a 2.5× speedup.
- Fixstars AIBooster incorporates Performance Intelligence (PI) to automate these types of optimizations, supporting the entire process from measurement to final tuning.

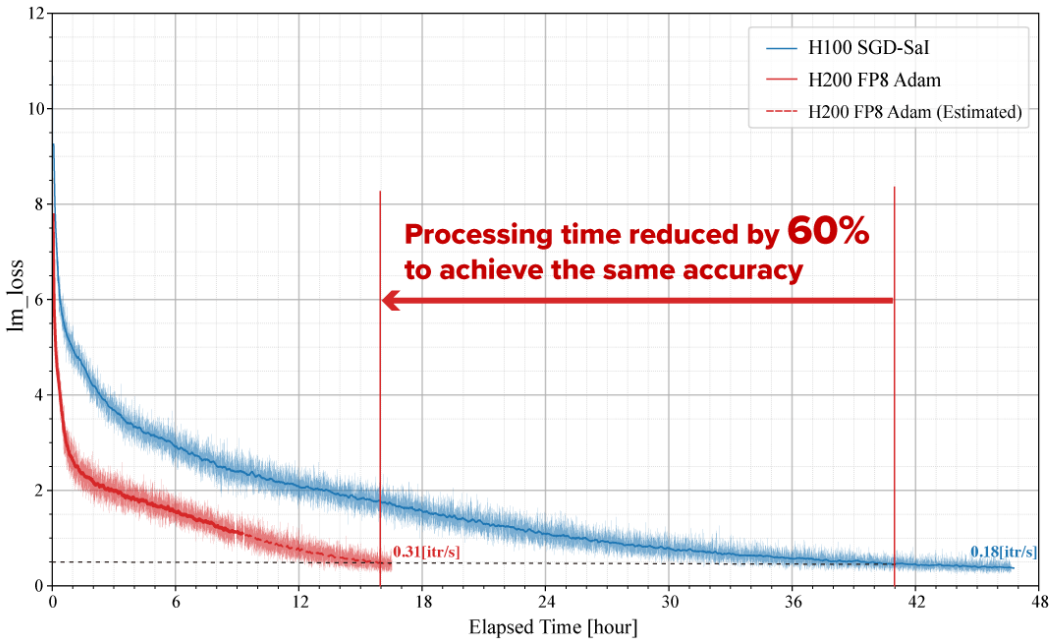

Figure 1: Loss curve after performance engineering on H200 (red). Compared with H100 (blue), the time to reach equivalent accuracy drops by 60%.

1. Background

In recent years, with the rise of generative AI and large language models, AI model sizes have been growing exponentially. In practice, typically we first select a model capacity sufficient to meet task requirements (an accuracy-first approach), confirm its utility, and then optimize. Therefore, GPU memory, which is often the strongest hardware constraint

for AI training and inference tasks, is commonly used to its limit, leaving little room for performance optimization—a situation that tends to lead to inefficiency.

For example, consider running pre-training of "Llama 3.1" with 70B parameters, a medium-scale language model, in a practical minimum configuration. While execution is possible using 2 servers with NVIDIA H100 (80GB memory/GPU) × 8 cards each (total 16 GPUs), prioritizing operation over performance leads to hyperparameter settings that are unfavorable for speed:

- Micro-batch size (MBS) = 1: Only one sample processed at a time during training
- Tensor parallel (TP) = 8, Pipeline parallel (PP) = 2, Data parallel (DP) = 1: 70B model split across 16 GPUs for parallel execution
- Computational precision: BF16: Computation using only 16-bit width
- Activation recomputation: recompute-granularity=full: No intermediate activation storage, recalculate everything
- Optimization algorithm: SGD-Sal[1]: Memory-efficient but slow convergence method

While detailed explanations of each item follow later, all represent results of prioritizing memory constraint satisfaction, significantly suppressing throughput (itr/s, TFLOP/s). This is a typical example of sacrificing processing speed to maintain model accuracy.

This situation where GPU memory shortage prevents adding effective measures for speed improvement has become commonplace in current AI infrastructure. There are many scenarios where AI acceleration/GPU acceleration is physically difficult despite the desire to achieve it.

The key point is recovering memory capacity with NVIDIA H200 and leveraging performance engineering to utilize it effectively.

2. NVIDIA H200 Advent and Its Significance

The NVIDIA H200 (official name: NVIDIA H200 SXM), which appeared in late 2023, is the latest GPU architecture succeeding the NVIDIA H100. H200's greatest feature is its 141GB memory per GPU, approximately 1.8× the H100's 80GB. However, the GPU's computational performance itself (core configuration and clock) remains equivalent to H100, with virtually no fundamental differences beyond memory capacity. Therefore, simply replacing H100 with H200 will unfortunately not make processing faster. H200's value depends on how the increased memory is utilized—performance improvements are only achieved by effectively leveraging the additional memory.

Additionally, the Sakura Internet GPU cloud service "Koukaryoku PHY" H200 model[2] used in our verification has 4× total interconnect (inter-GPU communication) bandwidth compared to equivalent H100 models (Table 1). While inter-GPU communication can become a bottleneck in large-scale distributed training, the H200 environment strengthens this communication aspect, establishing a foundation to fully utilize increased computational resources.

Furthermore, H200 upgrades offer cost benefits. Generally, H200-equipped servers cost approximately 1.1× to 1.3× more than H100-equipped servers, while memory capacity is 1.8×. This means introducing H200 servers is more economical than preparing 1.8× H100 servers to secure the same memory capacity, enabling cost-effective system construction with proper tuning.

For these reasons, the emergence of new hardware H200 presents an excellent opportunity for AI infrastructure managers to explore what can be accomplished with increased memory. Fixstars challenged "making the same AI processing faster" using H200, improving areas compromised due to memory constraints on H100, achieving 2.5× speedup compared to H100. The following chapters detail the 5 performance engineering measures that realized this acceleration.

Table 1: Sakura Internet Koukaryoku PHY Model Specifications

PHY Model Name	H100 Sec.A	H100 Sec.B/C	H200 Sec.D/E
Chassis	SuperServer SYS-821GE-TNHR	SuperServer SYS-821GE-TNHR	SuperServer SYS-421GE-TNHR2-LCC
CPU Type	Xeon Platinum 8480+ (Sapphire Rapids)	Xeon Platinum 8480+ (Sapphire Rapids)	Xeon Platinum 8580 (Emerald Rapids)
CPU Count	2	2	2
Main Memory [TB]	2	2	1.5
GPU Type	H100 SXM(80G)	H100 SXM(80G)	H200 SXM(141G)
GPU Count	8	8	8
OS NVMe Storage [TB]	0.8	0.8	0.8
OS NVMe Storage Count	2	2	2
Non-OS NVMe Storage [TB]	7.68	7.68	7.68
Non-OS NVMe Count	4	4	4
External Network Bandwidth [Gbps]	10	25	25
External Network Ports	2	2	2
Interconnect Bandwidth [Gbps]	200	400	400
Interconnect Ports	4	4	8

3. Important Measurement Metrics

One pitfall to avoid when pursuing AI acceleration/GPU acceleration is starting performance improvement with unclear metrics.

To achieve reproducible speedup results, both speed (throughput) and accuracy (loss/convergence) must be quantified, including distance from hardware theoretical limits. This chapter explains 4 essential metrics for evaluating subsequent performance engineering measures.

3.1 Training Throughput (iteration/sec = itr/s)

Definition: Number of training iterations executable per second. Many frameworks and scripts output it/s or time/iter in logs.

Use: Effective for ETA prediction and immediate comparison after setting changes.

Note: itr/s is a "speed-only" metric that doesn't reflect algorithmic differences affecting computation to reach equivalent accuracy (e.g., optimizer differences). itr/s alone cannot evaluate final results.

3.2 Computational Throughput

Definition: Floating-point operations per second. Operations per iteration for transformers can be approximated as "6 × parameter count × token count"[3]. Dividing by measured iteration time gives computational throughput, measured in FLOP/s (Floating OPerations per Second).

Use: GPUs have hardware specifications for "floating-point operations executable per second," professionally called Rpeak (peak performance). Computational throughput quality is evaluated by ratio to hardware's theoretical peak performance—"performance efficiency."

$$\begin{aligned} &Performance\ Efficiency \\ &= \frac{Computational\ Throughput\ (measured)}{GPU\ Theoretical\ Peak\ Performance\ (Rpeak)} \end{aligned}$$

For example, NVIDIA H100 SXM has Rpeak of 989.4 [TFLOP/s] for 16-bit (FP16/BF16) and 1978.9 [TFLOP/s] for 8-bit (FP8)[4]. As guidelines for LLM pre-training, performance efficiency ranges:

- Initial state: ~10%
- Basic optimization: 20-30%
- Thorough optimization: ~50%

- Maximum tuning: ~80%

These are empirical values from Fixstars' performance engineering experience.

Note: Execution isn't always compute-bound; memory bandwidth, I/O, and interconnect may be limiting factors. However, for scenarios known to be compute-bound like transformer pre-training, this is the clearest indicator of distance from hardware theoretical limits.

Tip: Megatron-LM with **--log-throughput** outputs elapsed time per iteration (ms) and throughput per GPU (TFLOP/s/GPU) in logs, enabling simultaneous tracking of itr/s and TFLOP/s.

3.3 Loss Curve (vs Iteration Count)

Definition: Curve plotting horizontal axis = training iterations, vertical axis = loss (model error).

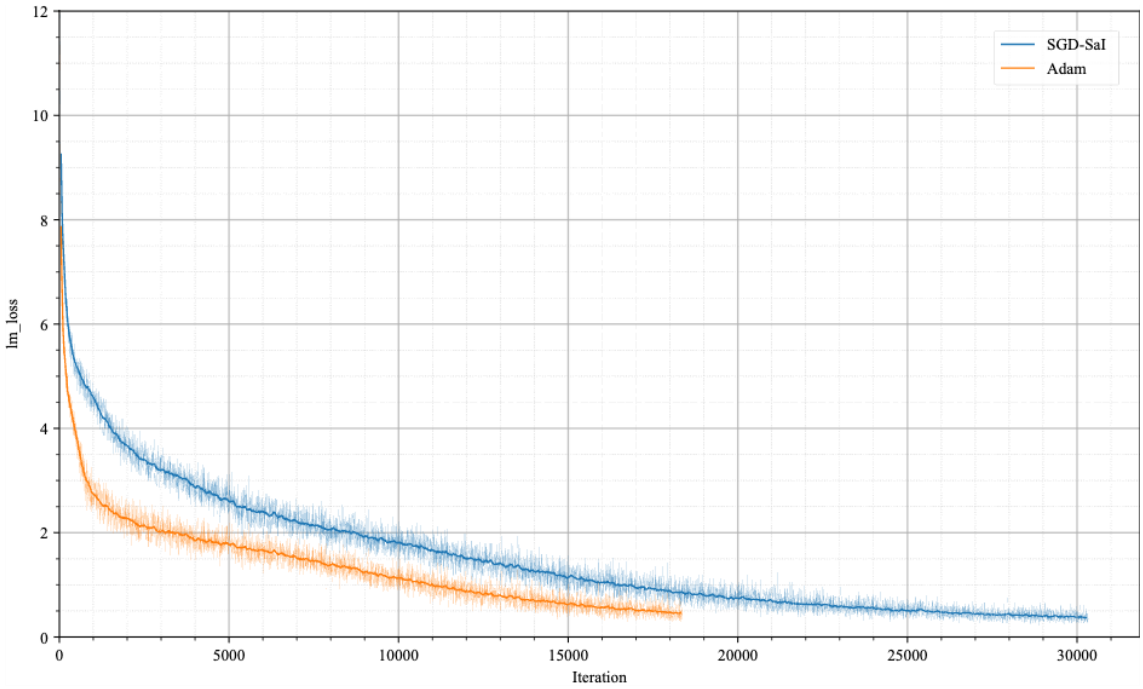

Figure 2: Typical loss curve example (vs iteration count)

Usage: Figure 2 compares "convergence efficiency differences" between two algorithms. It visualizes loss (model error) decreasing as iterations progress. In this example, Adam (orange line) requires fewer iterations to reach the same loss compared to SGD-SaI (blue line).

Note: Speed (itr/s) differences aren't reflected here. Even with fewer iterations, if one iteration is significantly slower, time won't be reduced. From a performance engineering perspective, this metric alone cannot determine which method is better.

3.4 Loss Curve (vs Time)

Definition: Horizontal axis = elapsed time, vertical axis = loss. Since the horizontal axis combines speed (itr/s) and convergence efficiency (iteration count), this is the most important metric for performance engineering.

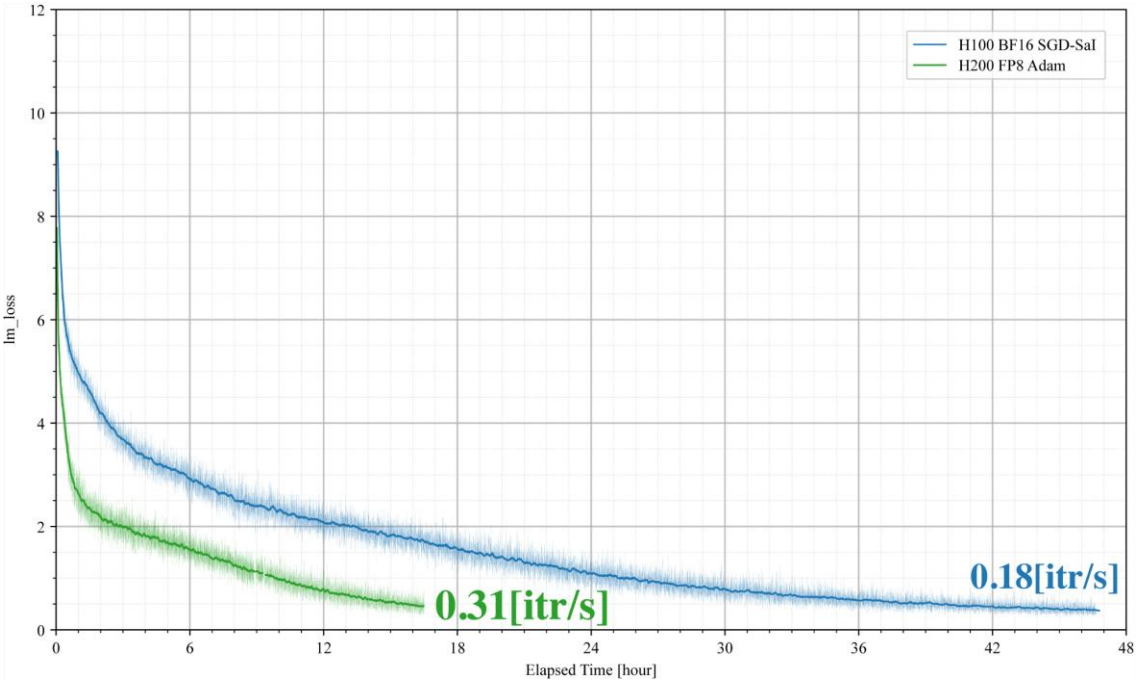

Figure 3: Example loss curve vs elapsed time

Usage: Set a target loss value (e.g., lm_loss = 1.0) and compare the measured time (h) to reach it. Figure 3 shows H100 conventional settings (blue line) taking ~42 hours while H200 optimized (green line) takes ~16 hours to reach the same loss. Time is reduced to ~40%, meaning equivalent models are obtained 2.5× faster. As mentioned earlier, 1.5× faster in iteration count plus training throughput improvement from 0.18→0.31 [itr/s] (approximately 1.7×) results in 1.5×1.7=2.5.

Note: When applying FP8 or changing recompute for speed without affecting accuracy behavior, confirming with iteration-count loss curves increases reliability.

4. Performance Engineering Strategies

Fixstars enabled H100-compared 2.5× faster execution using H200's increased memory capacity for "Llama 3.1" 70B parameter pre-training. The key was the following 5 performance engineering measures. We carefully determined where to allocate increased memory and what setting changes to make, implementing optimization while considering speed improvement and memory consumption trade-offs.

This section explains the five key areas: (1) 3D parallelism, (2) micro-batch size, (3) computational precision, (4) activation recomputation, and (5) optimization algorithms, covering their objectives and effects.

4.1 Three-Dimensional Parallelism (Tensor Parallel, Pipeline Parallel, Data Parallel)

Large model training commonly uses three-dimensional (3-directional) parallel processing, splitting models and data across multiple GPUs. Main parallel methods include:

- TP (Tensor Parallel): Split matrices (tensors) of each layer row- or column-wise for simultaneous computation across multiple GPUs.
    - Characteristic: Heavy inter-node communication typically requires intra-node completion. Excessive TP increase causes All-Gather and Reduce-Scatter collective communication bottlenecks.
- PP (Pipeline Parallel): Split layers front-to-back, placing Stage-0/1... on separate GPUs.
    - Characteristic: Smaller is better. Layer computational dependencies cause later-stage GPUs to idle while earlier stages compute (pipeline bubble).
- DP (Data Parallel): Prepare multiple identical model replicas, each training different batches. Synchronize gradients via All-Reduce.
    - Characteristic: Product of the 3 parallel numbers must match GPU count. DP is the most efficient parallelization, so minimize TP, PP and maximize DP.

Total GPUs = TP × PP × DP. H100 environment used 16 GPUs with TP=8, PP=2, DP=1 configuration (8 units horizontally splitting the model, 2-stage pipeline across 2 nodes, no data parallelism) for 70B models. H200's memory increase expands parallel method options, so we considered other parallel configurations like increasing DP.

After testing several combination patterns, we found that changes in parallelism configuration had relatively small performance impact. While it is generally known that "data parallel is more efficient than pipeline parallel," under the current conditions, maintaining the same TP=8, PP=2 configuration as H100 proved optimal, and we concluded that rather than forcing an increase in data parallelism, the effect from micro-batch expansion described later was greater. This judgment allowed us to achieve

performance improvements while avoiding complex changes arising from parallel algorithm modifications.

4.2 Micro-Batch Size (MBS)

Micro-batch size is the number of samples one GPU processes per iteration. Increasing this number enables more efficient parallel GPU computation, improving throughput (processing per unit time), but requires more memory. H100 environment forced minimum micro-batch size of 1 due to memory constraints. The second measure leveraging H200's additional memory was maximizing this micro-batch size. We first investigated its impact and maximum value by fixing global batch size=2048, Adam, Selective, TP=8.

As shown in Figure 4, increasing MBS consistently improved performance, matching known conventional wisdom. However, FP8 showed greater improvement when increasing MBS compared to BF16, and pipeline parallel 1 (data parallel 2) couldn't run MBS=4 due to memory shortage (Table 2). FP8's greater effectiveness likely stems from larger Rpeak providing more improvement potential compared to BF16.

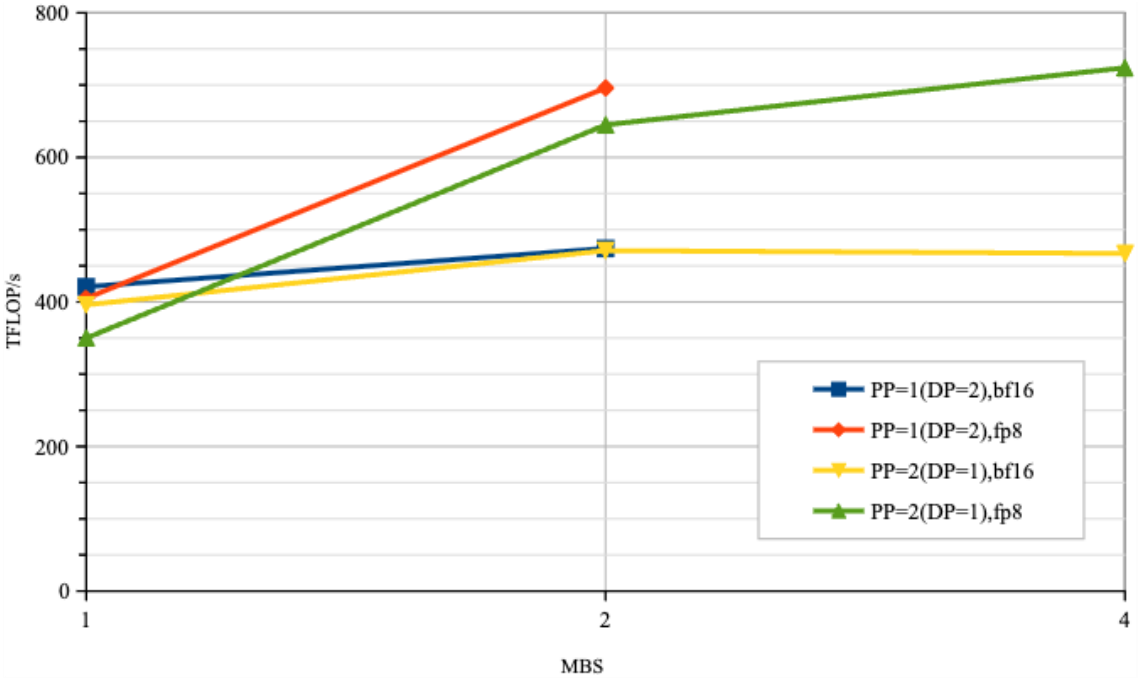

Figure 4: Computational throughput improvement with micro-batch size increase

Table 2: Computational throughput improvement with micro-batch size increase (details)

PP	DP	Precision	MBS	TFLOP/s/
1	2	BF16	1	431
1	2	BF16	2	474
1	2	BF16	4	Out of Memory
1	2	FP8	1	405
1	2	FP8	2	696
1	2	FP8	4	Out of Memory
2	1	BF16	1	396
2	1	BF16	2	471
2	1	BF16	4	467
2	1	FP8	1	350
2	1	FP8	2	645
2	1	FP8	4	724

Setting MBS to 4 alone achieves approximately 1.8× throughput improvement compared to MBS=1, leading to significant speedup. While increasing MBS enables effective computational resource utilization, trade-offs emerge with other measures (e.g., parallelism degree, computational precision). After exploring optimal solutions combined with other parameter adjustments, we determined that keeping H200 environment parallel configuration same as H100 while only increasing micro-batch size to 4 was best. The important achievement is that H200's large memory capacity created headroom, enabling larger batch sizes for performance improvement.

4.3 Computational Precision (FP16 vs FP8)

GPU AI computation can increase operations per second by reducing numerical computation precision (bit width). H100/H200 both support FP16-equivalent half precision (primarily Brain Float 16 = BF16) plus even lower precision 8-bit floating point (FP8) operations. Theoretically, FP8 enables faster processing than FP16, making FP8 utilization attractive for maximizing H200 computational performance.

However, the Megatron-LM training framework used allocates additional buffers internally when using FP8, increasing memory consumption when FP8 is utilized. While reducing precision intuitively seems to reduce memory consumption, implementation reasons can sometimes cause opposite effects. This creates a trade-off: "consume memory for FP8

acceleration or use FP16 slower but allocate memory elsewhere." We compared these options with the same global batch size=2048, Adam, Selective, TP=8.

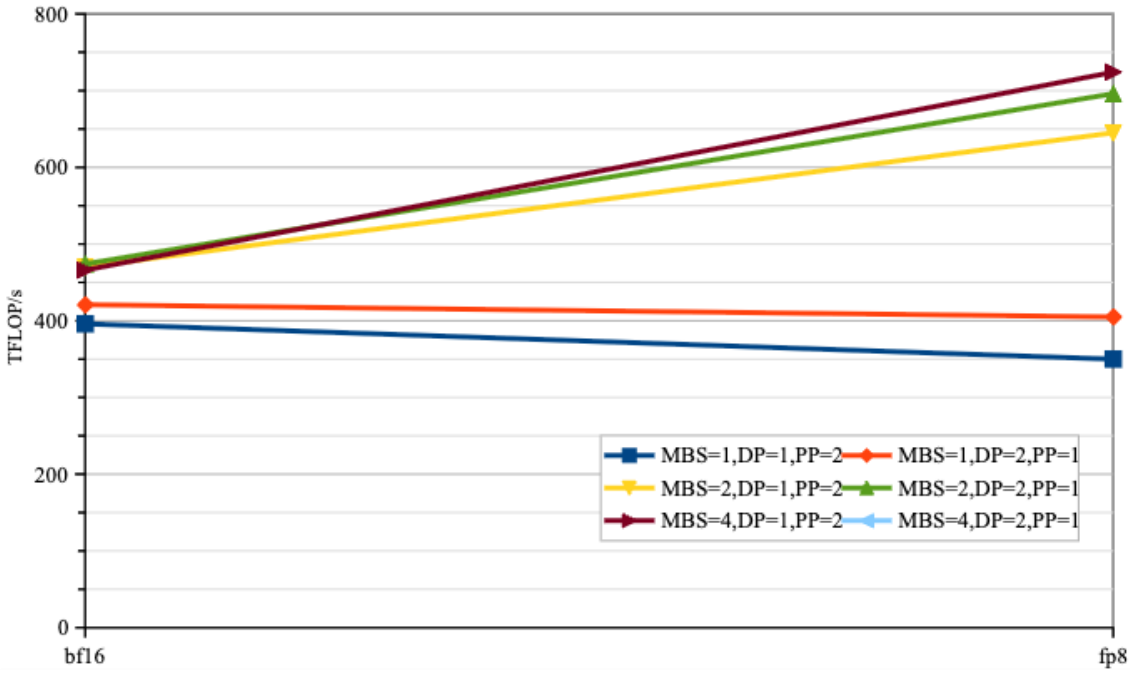

Figure 5: Computational throughput improvement by computational precision

Comparison experiments confirm FP8 is generally faster. Particularly when micro-batch size increases to 2 or higher, FP8 utilization shows significant throughput improvement. For example, under MBS=4, DP=1, PP=2 conditions, FP8 achieved 724 TFLOP/s versus FP16's 466 TFLOP/s—approximately 1.55× throughput improvement. However, MBS=1 surprisingly showed FP8 slightly slower (this point remains unclear, but since MBS=1 deviates from optimal solutions, we didn't pursue further investigation). Regardless, FP8 utilization achieves computational acceleration itself, indicating H200 performance extraction potential.

We also verified precision concerns (impact on training convergence). Results confirmed virtually no difference in loss curves between FP16 (BF16) and FP8. Figure 6 compares loss progression when training with FP16 vs FP8—both curves nearly overlap, with differences barely visible even when magnified (magnification shows slight offsets between orange-green and purple-brown pairs).

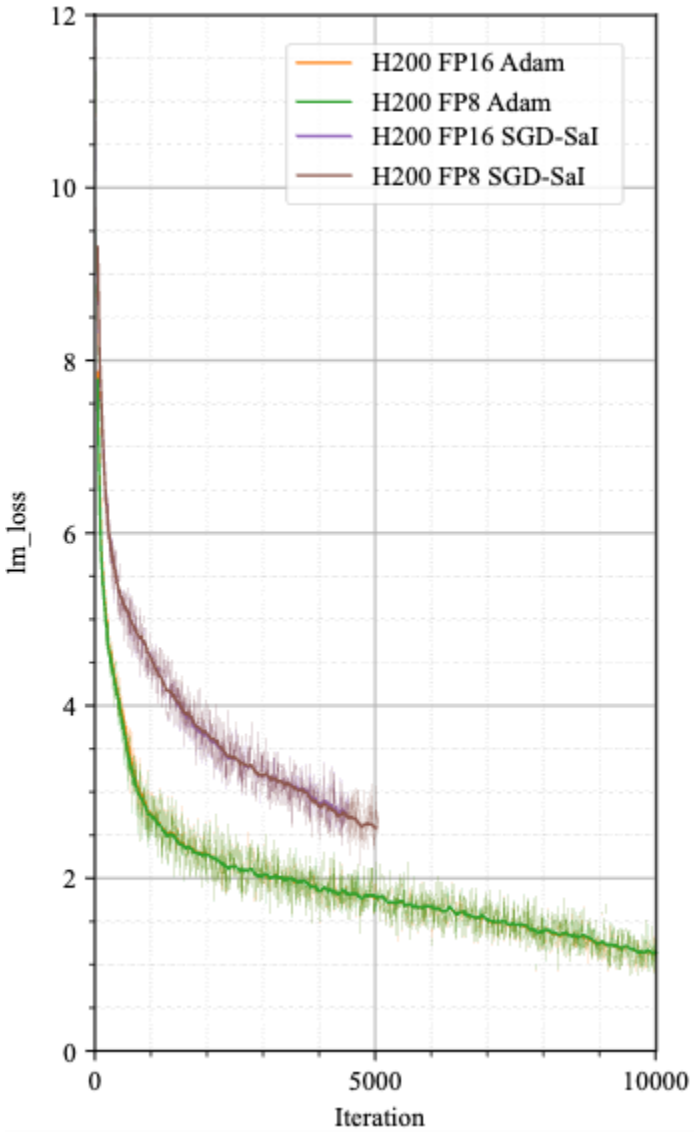

Figure 6: Loss progression comparison between FP16 vs FP8 training

Megatron-LM's FP8 implementation is highly sophisticated, confirming that FP8 enables training progress without accuracy degradation. Therefore, we concluded that H200 environments should actively incorporate FP8 computational acceleration within memory limits.

4.4 Activation Recomputation (Recompute)

Neural network training allows choosing between storing intermediate activations from forward propagation in memory or recalculating during backpropagation. Storage increases memory usage but accelerates computation, while no storage saves memory but increases computational cost through recalculation during backpropagation. Generally, three modes exist:

- **full** : Store no intermediate values, recalculate everything when needed (minimum memory, high computational load)
- **none** : Store all intermediate values, no recalculation (maximum memory, no computational load)
- **selective** : Store only important parts, recalculate others (medium memory, small computational load)

H100 environment used full (full recomputation ON) for memory savings. Full involves much computational waste (repeatedly calculating same activations)—none would be fastest. However, none is nearly impossible for 70B model scale even with H200 (memory shortage). Additionally, paying some recalculation cost to save memory and using saved memory for other acceleration measures might be faster overall. Therefore, we tested selective (recalculate except core layers) to leverage H200's increased memory while reducing computational cost.

Measurements across several combination patterns showed full-to-selective changes clearly improve throughput. Particularly for high memory consumption settings like extending micro-batch size to 4 and applying FP8, selective setting was prerequisite to avoid memory shortage preventing operation. Ultimately, adopting selective mode enabled by H200's additional memory achieved approximately 1.5× throughput improvement combined with other measures. As expected, none (no recomputation) couldn't be measured due to memory shortage. These results demonstrate that recomputation settings appropriately revised according to H200's memory capacity contribute to acceleration.

4.5 Optimization Algorithms

AI training processes conclude with optimization (Optimizer) that updates model parameters using computed gradients. This optimization algorithm selection also involves memory usage and training speed trade-offs. Generally, Adam-family methods (adaptive moment estimation) achieve fast convergence and high accuracy while consuming additional memory by maintaining second-order gradient moments. Conversely, SGD-family methods (stochastic gradient descent) are simple with small memory overhead but

tend toward slower convergence accuracy-wise. H100 environment used a special memory-efficient SGD variant SGD-SaI for memory savings.

H200's memory availability made accuracy-focused Adam usage practical. First, comparing per-iteration throughput for both optimizers showed virtually no difference (under TP=8, PP=2, DP=1, MBS=4, FP16 conditions: Adam: 384 TFLOP/s, SGD-SaI: 391 TFLOP/s). Thus, switching to Adam doesn't reduce GPU computational efficiency.

Next, we compared crucial "final loss convergence speed" by running long-term training with both methods (Figure 7). SGD-SaI required approximately 1.5× iteration count to reach target loss values. Specifically, reaching certain loss values (e.g., lm_loss = 1.0) required 27,000 steps for SGD-SaI versus 18,000 steps for Adam. Conversely, Adam reduces loss (converges training) approximately 1.5× faster than SGD-SaI.

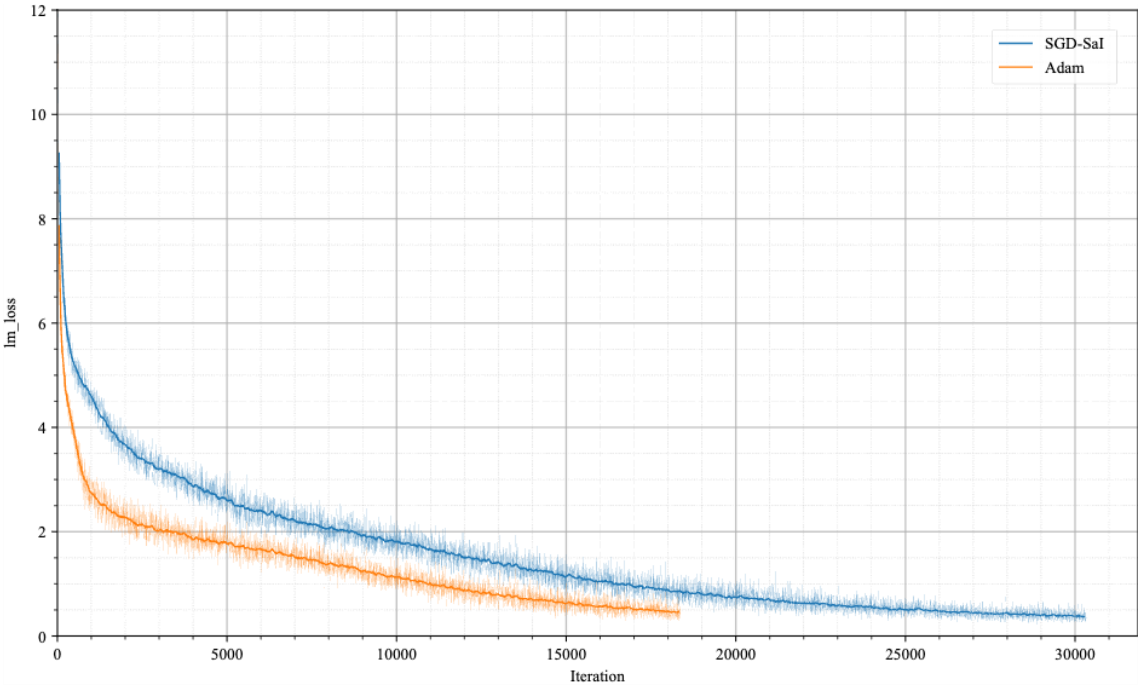

Figure 7: Loss curves by optimization algorithm

Based on these results, we concluded that H200 environments should adopt Adam-family optimizers with accuracy benefits outweighing memory loads. Enabling fast convergence algorithms through increased memory size represents another significant H200 acceleration factor.

4.6 Summary of Various Measure Effects

The 5 measures' setting changes from H100→H200 were:

- Micro-batch size: increased from 1 to 4
- Computational precision: enabled FP8
- Recomputation: reduced amount (selective)
- Optimization algorithm: switched to high-accuracy method (Adam)

These are summarized in the following graph:

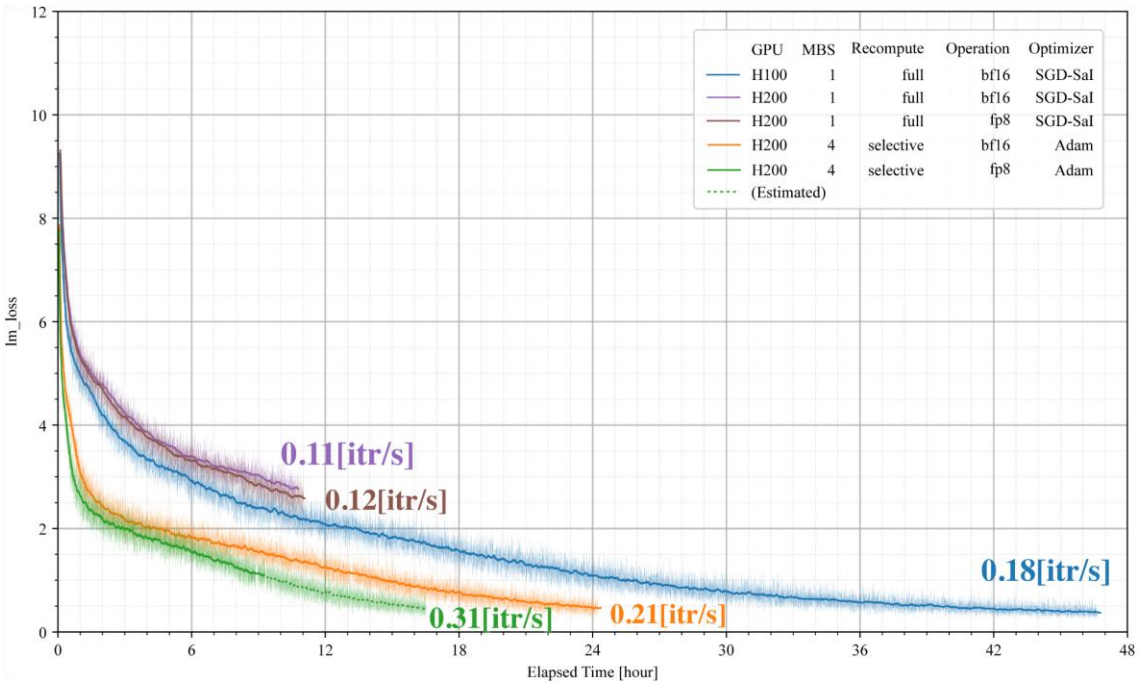

Figure 8: Loss curves by implemented optimization methods

The blue-to-green line represents final changes, with time to reach loss=1 reduced from ~42 hours to 16 hours—40% reduction = 2.5× speedup.

Besides blue and green lines, three additional lines warrant explanation. First, direct H200 migration from H100 initial state (purple line) shows unexpected slowdown. Since iteration-count graphs matched blue lines, this appears caused by pure training throughput [itr/s] degradation. We haven't identified this cause—it remains future work—but likely involved H200 driver or CUDA version issues. The brown line shows changing computational precision from BF16 to FP8 alone, with minimal performance improvement. The orange line differs from final results (green line) only in computational precision changing from FP8 to BF16.

5. AIBooster Optimization Process Efficiency

The above 5 measures were derived through Fixstars' trial-and-error process to extract H200 environment potential. However, manually implementing all these optimizations in actual operational settings is challenging, and doing so repeatedly for every hardware or model change is impractical. This is where Fixstars AIBooster[5] truly shines.

Figure 9: Performance Observability (PO) screen. Hardware resource utilization and software component processing times are visible.

Fixstars AIBooster is software aimed at optimizing computational resource utilization efficiency including GPUs while maintaining consistently high performance. It provides two functions: Performance Observability (PO) and Performance Intelligence (PI). PO function enables detailed measurement and visualization of hardware and application operational status (e.g., GPU utilization, computational efficiency, bottleneck processing among software components), while PI function automatically executes performance improvement tuning.

The H200 large model training acceleration knowledge covered applies broadly beyond generative AI pre-training. For example, middle-to-small range language model fine-

tuning and inference tasks also benefit from memory headroom enabling larger batch sizes or bigger models—expanding options for faster, higher-precision processing. Since optimal settings differ by model and dataset in practice, this section presents general performance engineering procedures using AIBooster as guidance for applying this white paper's methods in your environment.

Step 1: Current Performance Visualization and Issue Extraction
First, use AIBooster's Performance Observability function to measure target AI workload status. Dashboard visualization of GPU utilization, memory usage, computational efficiency, I/O wait time, etc. identifies bottleneck locations.

Step 2: Improvement Strategy Planning
Analyze which resources have capacity and which are constrained based on visualization data. For example, "GPU cores have capacity but memory is at limits" might benefit from batch size expansion or precision changes described in this document. Conversely, "GPU cores constantly at 100% indicates full computational capacity usage—consider model partitioning revision or GPU additions." Formulate hypotheses for appropriate measures.

Step 3: Automatic Tuning via Performance Intelligence
Next, use AIBooster's Performance Intelligence function to automatically explore and apply optimization strategies. The hyperparameter tuning tool, one PI function, can automatically execute searches like "gradually increase micro-batch size while testing recomputation setting (full→selective) and computational precision (FP16→FP8) combinations," measuring throughput and loss for each setting. Efficient comprehensive exploration of multiple parameter simultaneous optimization quickly identifies high-performance configurations.

Step 4: Optimal Configuration Application and Verification
Apply recommended settings from automatic tuning results to production jobs and confirm actual performance improvements. Repeat Steps 1-3 as needed until reaching target performance and accuracy. AIBooster enables consistent measurement-to-tuning workflows, enabling rapid cycle completion.

Step 5: Continuous Monitoring and Expert Support
Post-deployment, monitor systems via the AIBooster dashboard and perform tuning upon model or data changes. Fixstars engineers also provide deployment support and operational assistance. For example, Sakura Internet Koukaryoku PHY service users receive AIBooster installation kits when starting GPU server usage, with Fixstars engineers providing setup and environment adjustment support.

6. Conclusion

This white paper presented specific examples of directly connecting hardware evolution—

"memory capacity expansion"—to performance improvements through software ingenuity, introducing NVIDIA H200 GPU performance engineering methods and effects. From the perspective of reallocating surplus resources gained through H200 migration, we implemented 5 measures: (1) parallelism optimization, (2) batch size expansion, (3) low-precision computation utilization, (4) recomputation reduction, (5) fast convergence optimization algorithm adoption, achieving 2.5× AI acceleration compared to H100. These results demonstrate bottleneck resolution and optimization importance in AI infrastructure while highlighting that "extracting maximum hardware potential requires thorough performance tuning."

However, covering such tuning work manually is difficult. This is where Fixstars AIBooster comes in. AIBooster is a solution that semi-automates sophisticated performance engineering described in this document. It provides comprehensive support for AI processing visualization and acceleration (GPU acceleration, AI acceleration) on GPU servers, realizing integrated hardware-software optimization. If you're interested in AIBooster, please try its effects. Fixstars AIBooster is freely downloadable, and online demos show actual screens.

Contact us or apply for downloads via the link below—Fixstars' specialist team provides full support from deployment to utilization.

▶ Fixstars AIBooster Free Download Application: https://www.fixstars.com/ja/ai/ai-booster/get-started

References

[1] SGD-Sal
[2] NVIDIA H200(141GB) 8GPU Model Sec.D/E (Japanese)
[3] The FLOPs Calculus of Language Model Training
[4] NVIDIA H100 Tensor Core GPU Architecture
[5] Fixstars AIBooster Product Website