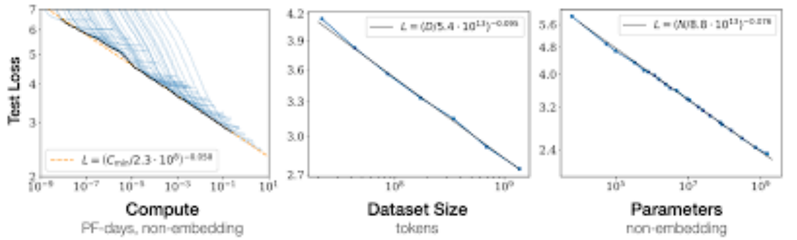
国内における大規模言語モデルの開発状況について

スーパーコンピューティング・ジャパン
タワーホール船堀
2024/03/12

東京工業大学
学術国際情報センター
横田 理央

ChatGPT公開以前の流れ

2019/06/22 MicrosoftがOpenAIに10億ドル(当時 約1100億円)を投資
2020/01/23 OpenAIが言語生成モデルに関するScaling Lawの論文を発表


2020/03/28 OpenAIが言語生成モデルGPT3に関する論文を発表
2020/06/11 OpenAIがGPT3のAPIを公開
2020/09/22 OpenAIがMicrosoftに対してGPT3のソースコードを公開
2021/01/05 OpenAIが画像+言語モデルCLIPと画像生成モデルDALL-Eを発表
2021/06/29 GithubがGPT3にコードを追加学習したCodexを利用したCopilotを発表
2021/07/28 Free Software FoundationがGithub Copilotに対するライセンス上の懸念を表明
2021/09/10 Naverが言語生成モデルHyperClovaを発表
2022/02/22 DeepMindがコード生成モデルAlphaCodeを発表
2022/03/29 DeepMindが言語生成モデルChinchillaを発表
2022/05/23 Googleが画像生成モデルImagenを発表

Teddy bears swimming at the Olympics 400m Butterfly event.
A cute corgi lives in a house made out of sushi.
A cute sloth holding a small treasure chest. A bright golden glow is coming from the chest.
2022/07/20 OpenAIが画像生成モデルDALL-E2を発表
2022/07/22 BigScience (HuggingFace,CNRS,GENCI)が多言語モデルBloomを発表
2022/08/04 精華大学が言語生成モデルGLM-130Bを発表
2022/08/22 Stability AIが画像生成モデルStable Diffusionを発表
2022/11/10 DeepMindが汎用モデルGatoを発表


2022/11/15 DeepMindが画像+言語モデルFlamingoを発表

ChatGPT公開以降の時系列

2022/11/30 OpenAIがChatGPTを発表
2022/12/04 公開後5日で100万ユーザを突破
2022/12/05 Stack OverflowがChatGPTで生成された投稿を禁止
2022/12/21 GoogleがChatGPTの自社への脅威に対してCode Red(非常事態)を宣言
2023/01/23 MicrosoftがOpenAIに100億ドル(当時 約1.3兆円)を投資

2023/02/01 OpenAIが月額$20の有料サービスChatGPT Plusを開始
2023/02/02 公開後約２ヶ月で1億ユーザを突破

2023/02/03 自民党AIの進化と実装に関するプロジェクトチーム (第１回)
2023/02/06 GoogleがChatGPTに対抗して対話型AIのBardを限定公開
2023/02/07 Microsoftが検索エンジンBingにChatGPTを導入
2023/02/09 ChatGPTが米国の医師国家試験に合格できるレベルとの論文が発表される
2023/02/24 Metaが言語生成モデルLLaMAのソースコードを非商用ライセンスで公開
2023/03/01 OpenAIがChatGPTとWhisper(音声認識)のAPIを公開 (ユーザデータは学習に使わないことを宣言)
2023/03/09 MicrosoftがAzure OpenAI ServiceでChatGPTを提供開始
2023/03/10 Googleが言語生成モデルPaLM2を発表
2023/03/14 ChatGPT PlusでGPT-4が利用可能に

2023/03/14 GoogleがPaLM2のAPIを公開

ChatGPT公開以降の時系列

2023/03/16 Microsoft 365 CopilotでWord,Outlook,TeamsなどからAIアシスタントが利用可能に
2023/03/20 ChatGPTから氏名,email,住所,カード番号などの個人情報が漏えい
2023/03/20 GitHubがGPT-4を搭載したCopilot Xを公開 →
2023/03/30 ChatGPTのAPIを利用したAI agentであるAuto-GPTが公開 →
2023/03/30 UC Berkeley,CMU,StanfordなどがGPT-4の会話データで微調整したVicunaを発表
2023/03/31 イタリアがChatGPTの利用を禁止 (4/28に撤回)
2023/05/19 G7広島サミットにて「広島AIプロセス」立ち上げ
2023/05/21 LLM勉強会 (第１回)
2023/06/01 「富岳」政策対応枠 利用開始
2023/06/05 TTI (Abu Dhabi)がFalcon-40Bのソースコードを商用ライセンスで公開 (09/06に180Bも)
2023/07/11 Anthropicが言語生成モデルClaude2を発表
2023/07/18 Metaが言語生成モデルLLaMA2のソースコードを限定的な商用ライセンスで公開 →
2023/09/07 Turingが画像+言語モデルHeronを公開
2023/09/25 AlibabaがQwen-14Bのソースコードを限定的な商用ライセンスで公開
2023/10/02 AmazonがAnthropicに12億ドル(約1800億円)を投資
2023/10/03 産総研の生成AI開発支援プログラムに国立情報学研究所(NII)とELYZA社が採択

ChatGPT公開以降の時系列

2023/10/26 Google, Microsoft, Anthropic, Open AI が AI Safety Fundに10億ドル(1500億円)を投資
2023/10/30 米国がAI規制に関する大統領令を発令
2023/11/17 Sam AltmanがOpenAIから解任→Microsoftに移る→OpenAIに戻る
2023/11/30 AlibabaがQwen-72Bのソースコードを限定的な商用ライセンスで公開
2023/12/06 AI Alliance立ち上げ：日本からは東大,慶應,SB Institute, sakana.ai, Sonyなどが参加
2023/12/07 GoogleがGemini, Alphacode2を発表

2023/12/08 Elon Muskが立ち上げたAIベンチャーXがGrokを発表
2023/12/21 RinnaがNekomataのソースコードを限定的な商用ライセンスで公開
2024/01/02 Mistral AIがMixtral 8x7Bのソースコードを商用ライセンスで公開
2024/02/15 GoogleがGemini Pro 1.5を発表
2024/02/16 OpenAIが動画生成モデルSoraを発表
2024/02/21 GoogleがGemmaのソースコードを商用ライセンスで公開
2024/02/26 Mistral AIがMistral Largeを発表
2024/03/04 AnthropicがClaude3を発表
2024/03/10 Sam AltmanがOpenAI CEO復帰
2024/03/11 東工大・産総研が Swallow-MS, Swallow-MXを商用ライセンスで公開

2024/03/12 Elon MuskがGrokをオープンソースにするとX上で宣言
2024/03/12 ELYZAがELYZA-70Bのソースコードを商用ライセンスで公開

GENIAC

時期：2024年2月開始

資源：Google Cloud, Microsoft Azure

株式会社ABEJA
代表取締役CEO
岡田 陽介

Sakana AI株式会社
Research Scientist
秋葉 拓哉

**大学共同利用機関法人
情報・システム研究機構**
大学共同利用機関法人　情報・システム研究機構　国立情報学研究所所長
黒橋 禎夫

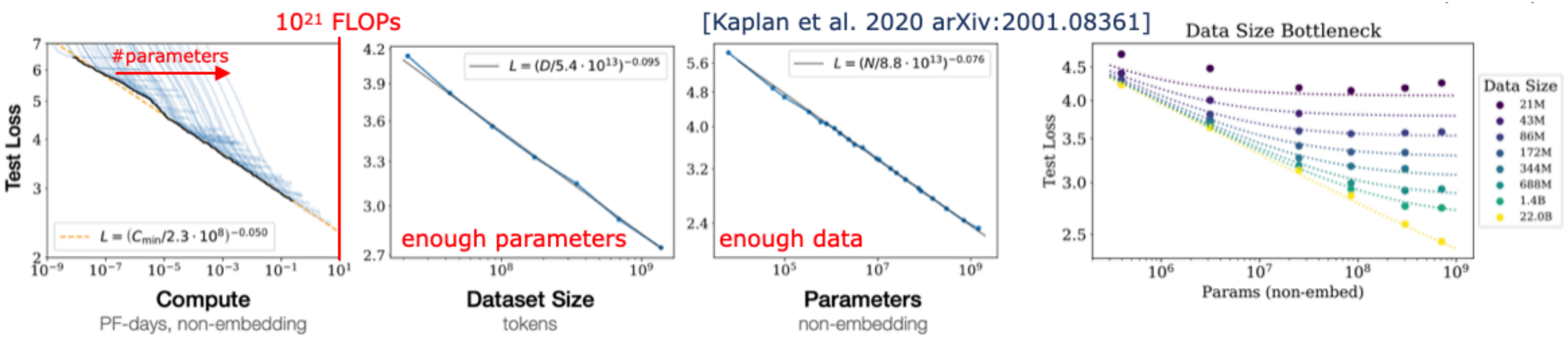
ストックマーク株式会社
取締役CTO
有馬 幸介

Turing株式会社
CTO
青木 俊介

国立大学法人 東京大学
東京大学大学院工学系研究科人工物工学研究センター／技術経営戦略学専攻 教授
松尾 豊

株式会社Preferred Elements
代表取締役社長
岡野原 大輔

Transformerのスケール則 (Scaling Law)



・パラメータ数が十分な場合、データ量に対してTest Lossはべき乗則に従って減少する

・データ量が十分な場合、バラメータ数に対してTest Lossはべき乗則に従って減少する

・必要な計算資源 [FLOPs] = パラメータ数 x テータ量 [tokens] x 5.7

・どのくらいの計算資源(予算)を投入すればどのくらいの性能のものができるかが予想できる

・データ量はパラメータ数に比例させる必要がある （データ量：パラメータ数=20:1）

・GPT-3は データ量：パラメータ数が2:1 になっており圧倒的にデータが不足している

・最近は推論コストを抑えるためにパラメータ数に対してデータ量を増やす傾向が顕著に

Model	Size (# Parameters)	Training Tokens
LaMDA (Thoppilan et al., 2022)	137 Billion	168 Billion
GPT-3 (Brown et al., 2020)	175 Billion	300 Billion
Jurassic (Lieber et al., 2021)	178 Billion	300 Billion
Gopher (Rae et al., 2021)	280 Billion	300 Billion
MT-NLG 530B (Smith et al., 2022)	530 Billion	270 Billion
Chinchilla	70 Billion	1.4 Trillion

[Hoffmann et al. 2022 arXiv:2203.15556]

GPT-4(3x10^{24} FLOPs)をスパコンで学習するには？

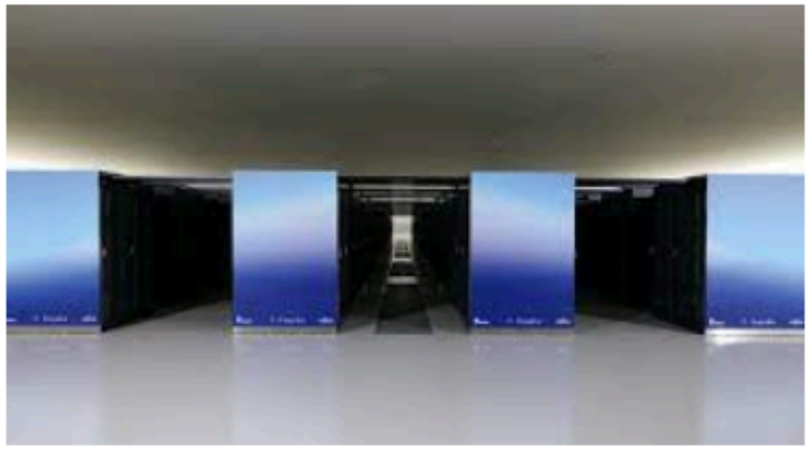
OpenAI (A100x25,000) 90日?

Frontier (MI250Xx37,888) 75日

LUMI (MI250Xx20,480) 140日

Aurora (PVCx63,744) 30日?

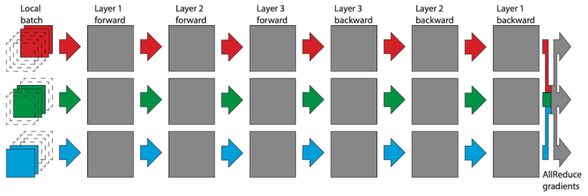
Fugaku (A64FXx158,976) 2100日



ABCI (A100x960+V100x4352) 770日

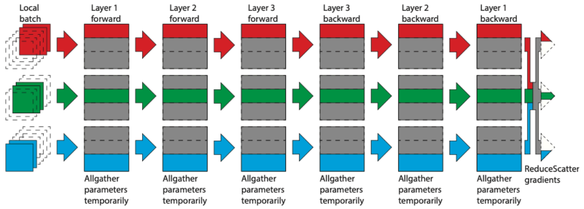
分散並列学習

データ並列 (DP)



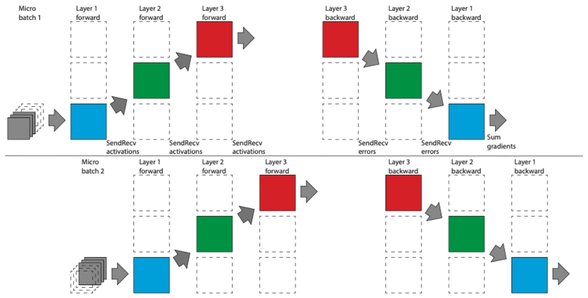
データ：分散
モデル：冗長
通信内容：勾配
通信形式：AllReduce
通信頻度：ステップ毎
長所：実装が簡単
短所：ラージバッチ問題
　　メモリ消費量

ZeRO (FSDP)



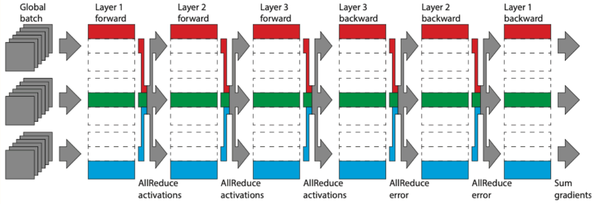
データ：分散
モデル：一時的に分散
通信内容：勾配＋重み
通信形式：ReduceScatter
　　＋AllGather
通信頻度：層毎
長所：実装が簡単
　　省メモリ
短所：ラージバッチ問題

パイプライン並列 (PP)



データ：冗長
モデル：分散
通信内容：活性
通信形式：SendRecv
通信頻度：層毎
長所：省メモリ
　　演算量低減
短所：パイプラインバブル

テンソル並列 (TP)



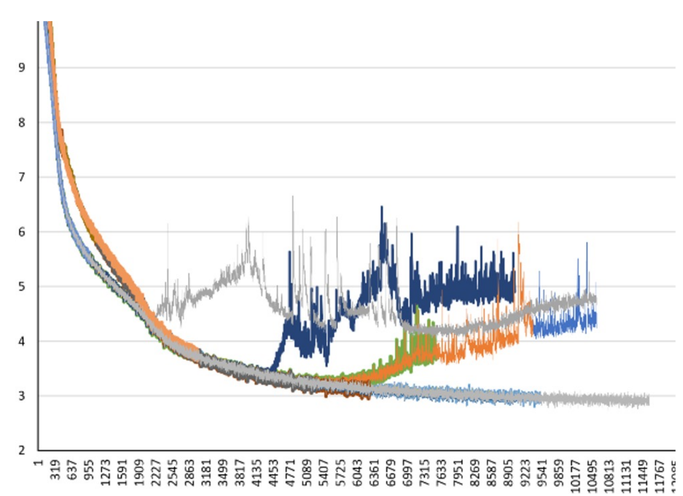
データ：冗長
モデル：分散
通信内容：活性
通信形式：AllReduce
通信頻度：層毎
長所：省メモリ
　　演算量低減
短所：通信オーバーヘッド
　　オーバーラップ不可
　　実装が複雑

深層学習フレームワークの「富岳」への移植

「富岳」版がGPU版と乖離

・テンソル並列を用いるとGPU版と結果が乖離

→乱数Seedの設定と反映タイミングを調整することで解決



Lossが途中でNaNになる

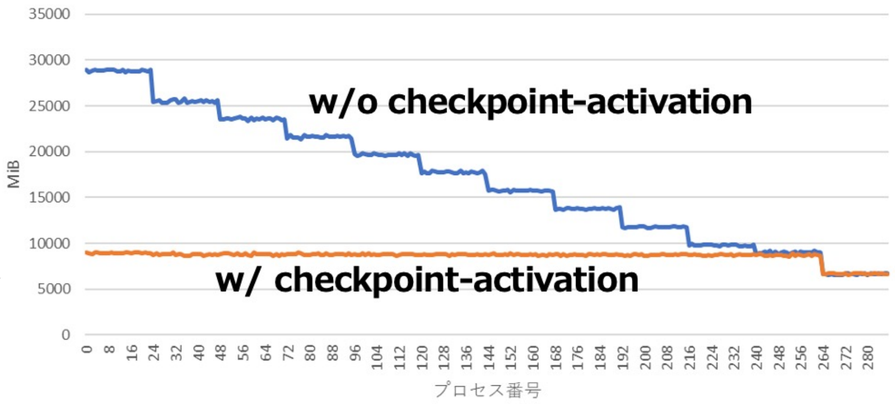
・学習途中でNaNが発生

→高速化したtanhにNaNガードが入っていないことが原因

メモリ消費量の異常

・メモリ消費量が異常に増加し、階段状に減少

→checkpoint-activationを適用(後段の層は適用外)することで解決



A64FX上での行列積(GEMM)とtanh関数の高速化

行列積(GEMM)

・GEMMを実行する際の配列形状などのパラメータ調整
・スレッド並列化の工夫

高速化前

1693389241.318550480.fcc.pytorch.y.r1.13_for_a64fx.tar のプロファイル結果

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	# of Calls
aten::bmm	18.07%	110.819s	18.08%	110.845s	24.055ms	4608
aten::bmm	18.17%	108.802s	18.17%	108.832s	23.618ms	4608
aten::bmm	18.53%	110.858s	18.53%	110.890s	24.065ms	4608
aten::bmm	19.15%	110.594s	19.16%	110.625s	24.007ms	4608
aten::bmm	18.33%	108.646s	18.34%	108.679s	23.585ms	4608

↓ 6倍高速化

高速化後

1701935794.711074240.fcc.pytorch.y.r1.13_for_a64fx.tar.gz のプロファイル結果

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	# of Calls
aten::bmm	3.56%	18.273s	3.57%	18.302s	3.972ms	4608
aten::bmm	3.64%	18.394s	3.64%	18.423s	3.998ms	4608
aten::bmm	3.57%	18.154s	3.57%	18.185s	3.946ms	4608
aten::bmm	3.58%	17.959s	3.59%	17.990s	3.904ms	4608
aten::bmm	3.61%	18.341s	3.62%	18.373s	3.987ms	4608

tanh関数

・exp関数を使用
・インライン展開
・SIMD化

高速化前

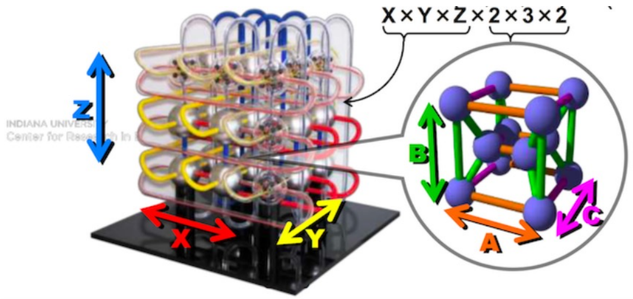
Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	# of Calls
aten::tanh	1.44%	6.964s	1.44%	6.964s	3.023ms	2304
aten::tanh	1.46%	6.997s	1.46%	6.997s	3.037ms	2304
aten::tanh	1.45%	6.945s	1.45%	6.945s	3.014ms	2304
aten::tanh	1.47%	6.886s	1.47%	6.886s	2.989ms	2304
aten::tanh	1.42%	6.813s	1.42%	6.813s	2.957ms	2304

↓ 2.5倍高速化

高速化後

Name	Self CPU %	Self CPU	CPU total %	CPU total	CPU time avg	# of Calls
aten::tanh	0.58%	2.800s	0.58%	2.800s	1.215ms	2304
aten::tanh	0.60%	2.836s	0.60%	2.836s	1.231ms	2304
aten::tanh	0.60%	2.824s	0.60%	2.824s	1.226ms	2304
aten::tanh	0.61%	2.844s	0.61%	2.844s	1.234ms	2304
aten::tanh	0.60%	2.831s	0.60%	2.831s	1.229ms	2304

集団通信(AllReduce)の高速化



Trinaryアルゴリズムの独自実装

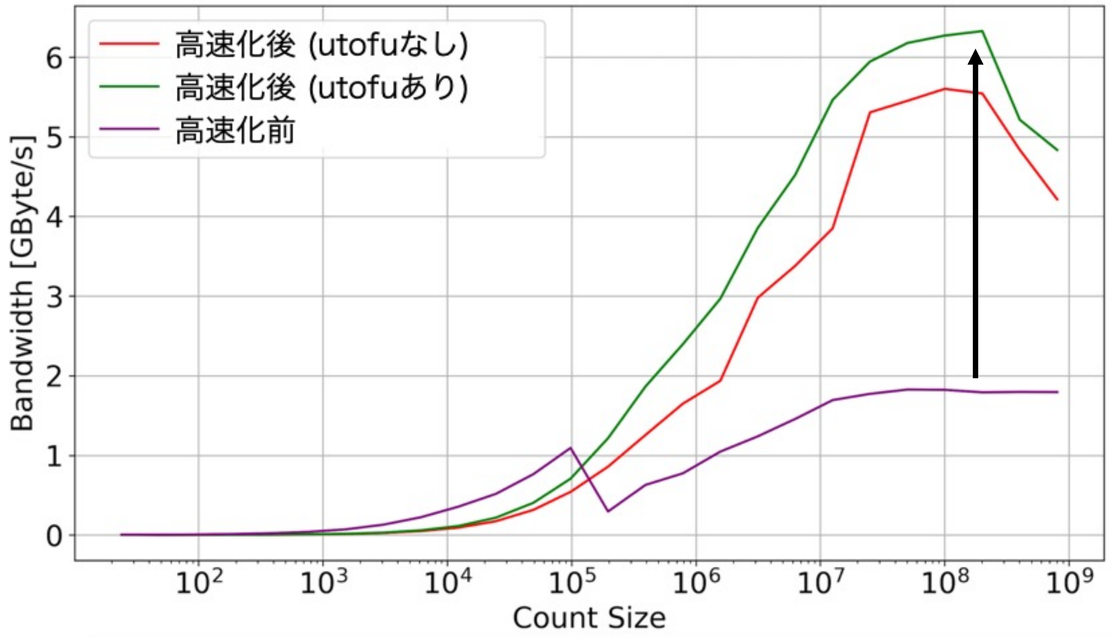
・Trinaryは富士通が開発したTofuネットワークを活かしたアルゴリズム
・ただし、サブコミュニケータでは利用できない
・データ並列とテンソル並列を併用する場合はサブコミュニケータを使わざるを得ない

→Trinaryアルゴリズムの独自実装を行う必要がある
・isend,irecvによる1対1通信を使って手動でAllReduce
・6方向のring AllReduceを実装 (3軸に対して双方向BCastと双方向Reduce)
・Rank mapを用いてノードの配置を最適化
・uTofuを用いたRDMA実装

足しこみの高速化

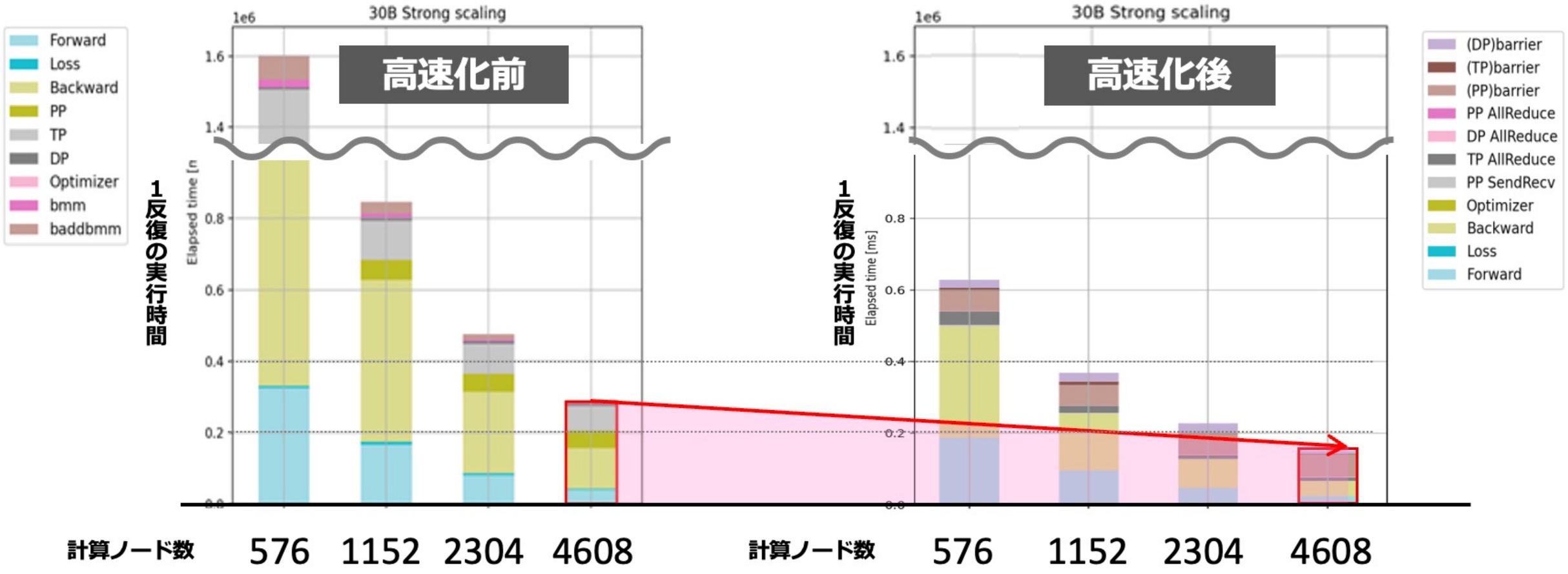
・Inline関数化
・OpenMPのスレッド競合の解決
・アシスタントコアを用いた通信と演算のオーバーラップ

3倍高速化



「富岳」4608ノードにおける実行時間の内訳

30Bパラメータ，4608ノードの設定では：高速化前0.8TFLOP/s → 高速化後1.6TFLOP/s
13Bパラメータ，13,824ノードの設定では： 高速化前0.6TFLOP/s → 高速化後0.9TFLOP/s
データ並列数を増やすと汎化性能が下がるため、テンソル並列・パイプライン並列が必要
→13,824ノードではパイプラインバブルによる性能低下



1.3B and 13B models

Data size: 270B tokens
Model size: 1.3B, 13B
Resource: 16 nodes x 8 A100s

	1.3B	13B
lr (min_lr)	2.0e-4 (1.0e-6)	1.0e-4 (1.0e-6)
init_std	0.013	0.008
num_layers	24	40
hidden_size	2048	5120
num_attn_heads	16	40
lr_warmup	2000	2000
lr_decay	cosine	cosine
lr_decay_tokens	1080B	1080B
Optimizer (decay)	Adam (0.1)	Adam (0.1)
Adam beta1, beta2	0.9, 0.95	0.9, 0.95
batch_size	1536	1536
clip_grads	1.0	1.0
use_flassh_attn	yes	yes
zero stage	1	1
Seq_len	2048	2048

Fix bug in bfloat16 optimizer related to checkpointing #4434

Merged tjruwase merged 10 commits into microsoft:master from okoge-kaz:fix/bf16-optimizer-state on Oct 8

Conversation 6 Commits 10 Checks 15 Files changed 2

okoge-kaz commented on Oct 2 • edited Contributor

Overview

Fixed conditional branching where errors occurred.

```
-       elif self.bfloat16_enabled() and not self.zero_optimization():
+       elif self.bfloat16_enabled() and hasattr(self.optimizer, "bf16_groups"):
```

Error Detail

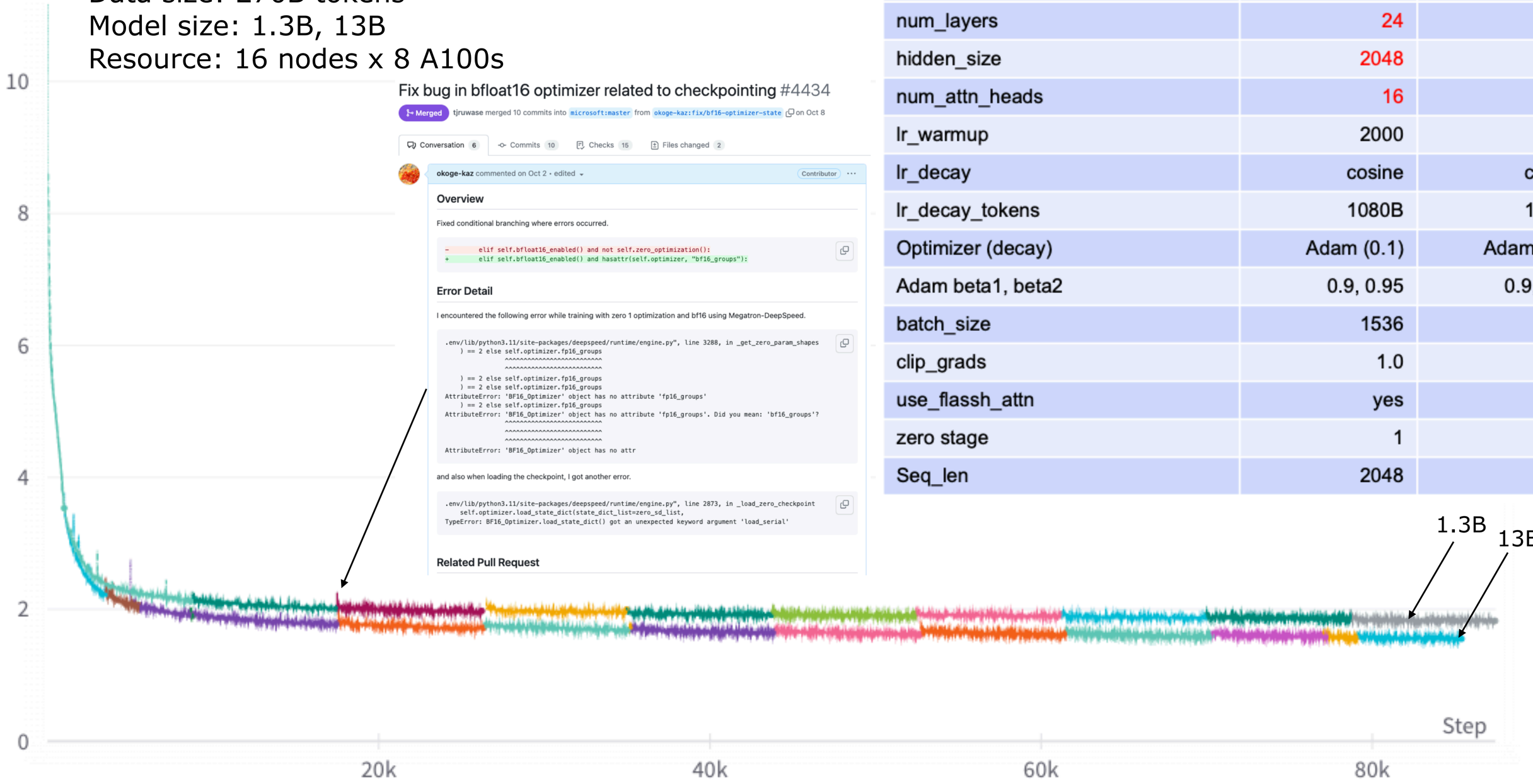
I encountered the following error while training with zero 1 optimization and bf16 using Megatron-DeepSpeed.

```
.env/lib/python3.11/site-packages/deepspeed/runtime/engine.py", line 3288, in _get_zero_param_shapes
    ) == 2 else self.optimizer.fp16_groups
                ^^^^^^^^^^^^^^^^^^^^^^^^^^
                ^^^^^^^^^^^^^^^^^^^^^^^^^^
    ) == 2 else self.optimizer.fp16_groups
    ) == 2 else self.optimizer.fp16_groups
AttributeError: 'BF16_Optimizer' object has no attribute 'fp16_groups'
    ) == 2 else self.optimizer.fp16_groups
AttributeError: 'BF16_Optimizer' object has no attribute 'fp16_groups'. Did you mean: 'bf16_groups'?
                ^^^^^^^^^^^^^^^^^^^^^^^^^^
                ^^^^^^^^^^^^^^^^^^^^^^^^^^
                ^^^^^^^^^^^^^^^^^^^^^^^^^^
AttributeError: 'BF16_Optimizer' object has no attr
```

and also when loading the checkpoint, I got another error.

```
.env/lib/python3.11/site-packages/deepspeed/runtime/engine.py", line 2873, in _load_zero_checkpoint
    self.optimizer.load_state_dict(state_dict_list=zero_sd_list,
TypeError: BF16_Optimizer.load_state_dict() got an unexpected keyword argument 'load_serial'
```

Related Pull Request

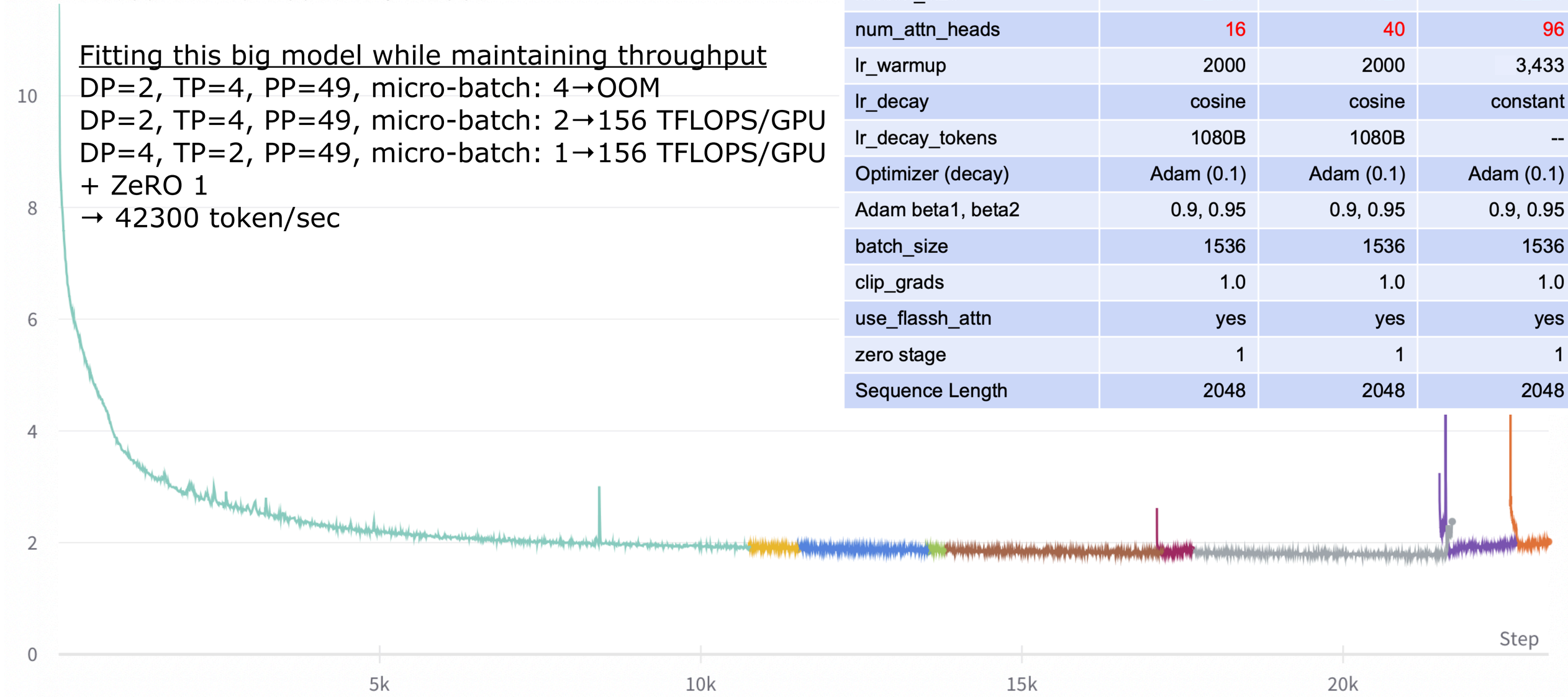


175B model

Data size: Stopped at 70B tokens
Model size: 175B
Resource: 49 nodes x 8 A100s

<u>Fitting this big model while maintaining throughput</u>
DP=2, TP=4, PP=49, micro-batch: 4→OOM
DP=2, TP=4, PP=49, micro-batch: 2→156 TFLOPS/GPU
DP=4, TP=2, PP=49, micro-batch: 1→156 TFLOPS/GPU
+ ZeRO 1
→ 42300 token/sec

	1.3B	13B	175B
lr (min_lr)	2.0e-4 (1.0e-6)	1.0e-4 (1.0e-6)	0.6e-4 (1.0e-6)
init_std	0.013	0.008	0.005
num_layers	24	40	96
hidden_size	2048	5120	12288
num_attn_heads	16	40	96
lr_warmup	2000	2000	3,433
lr_decay	cosine	cosine	constant
lr_decay_tokens	1080B	1080B	--
Optimizer (decay)	Adam (0.1)	Adam (0.1)	Adam (0.1)
Adam beta1, beta2	0.9, 0.95	0.9, 0.95	0.9, 0.95
batch_size	1536	1536	1536
clip_grads	1.0	1.0	1.0
use_flassh_attn	yes	yes	yes
zero stage	1	1	1
Sequence Length	2048	2048	2048



LLM @ Tokyo Tech. (model name: Swallow)

The logo of our university is a Swallow

Our supercomputer “Tsubame” means “Swallow” in Japanese

Continual pre-training from LLaMA-2 7B, 13B, 70B

Framework: Megatron-LM

3D Parallelism + Sequence Parallel + Distributed Optimizer(ZeRO 1)

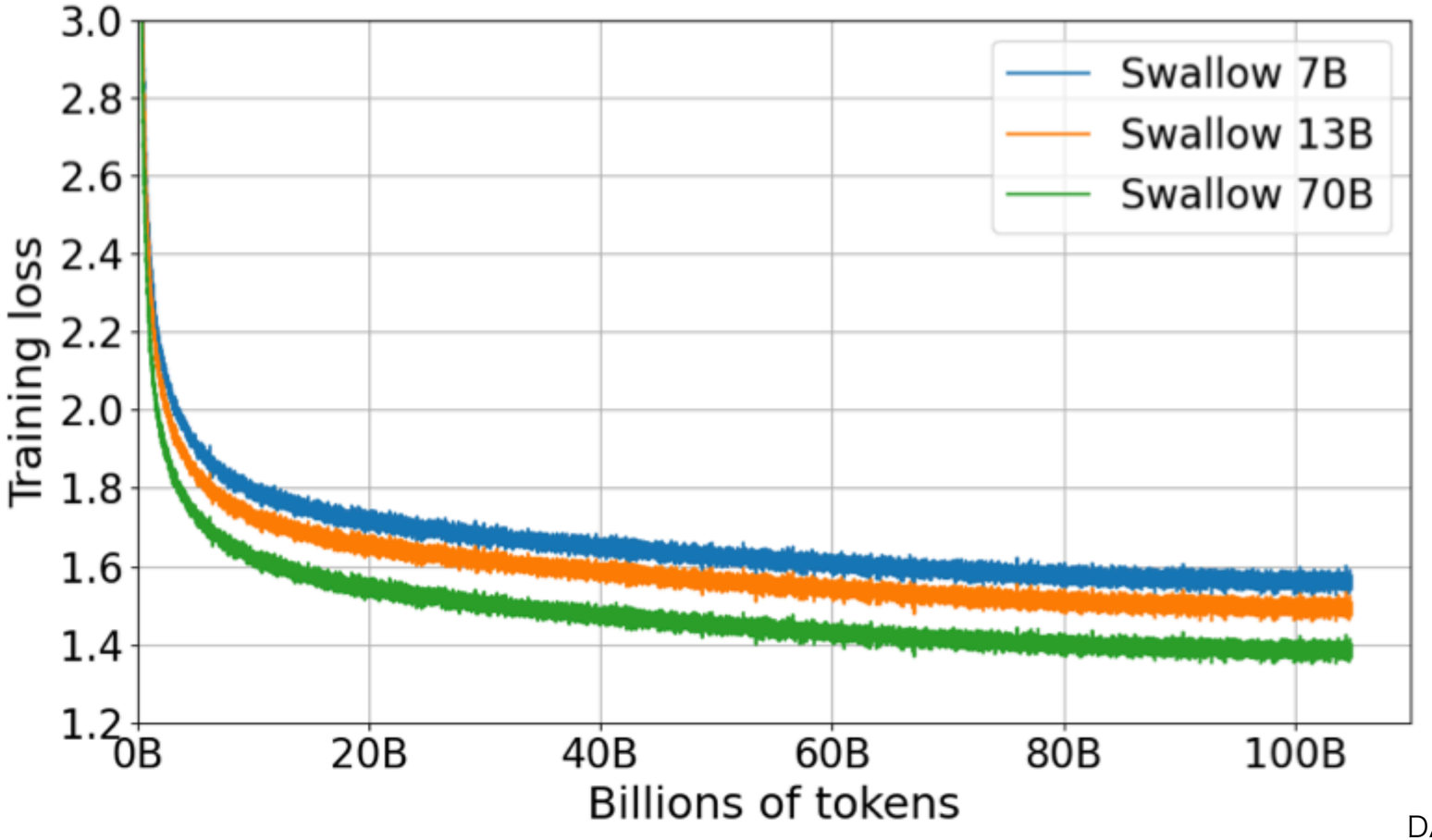
Research questions:

- Effect of extending the vocabulary
- Ratio of Japanese : English = 9 : 1 or 5 : 5
- Quality of the Japanese corpora

DALL-E generated image of swallows playing with a llama in Tokyo

LLM @ Tokyo Tech. (model name: Swallow)

パラメータ数	分散表現のサイズ	注意ヘッド数	層数	文脈長	GQA	トークン数	バッチ数	学習率
7B	4096	32	32	4096	無	約 100B	1024	1.0×10^{-4}
13B	5120	40	40	4096	無	約 100B	1024	1.0×10^{-4}
70B	8192	64	80	4096	有	約 100B	1024	5.0×10^{-5}



DALL-E generated image of swallows playing with a llama in Tokyo

日本語 LLM ベンチマーク

	run_name	Overall average	AVG	EL	FA	MC	MR	NLI	QA	RC	chabsa_	jamp_	janli_ɛ	jcomm	jemho
5	gpt-4-0125-preview	0.7722	0.6463	0.3066	0.2547	0.96	0.97	0.762	0.4693	0.8013	0.3066	0.58	0.98	0.96	
36	gpt-4-0613	0.7622	0.6463	0.3004	0.2199	0.96	0.97	0.768	0.415	0.891	0.3004	0.63	0.89	0.96	
40	gpt-4-1106-preview	0.7479	0.6295	0.317	0.2388	0.93	0.96	0.74	0.4002	0.8206	0.317	0.55	0.97	0.93	
33	gpt-3.5-turbo	0.6701	0.5161	0.2913	0.1886	0.79	0.67	0.56	0.2723	0.8406	0.2913	0.42	0.65	0.79	
15	anthropic.claude-v2:1	0.6682	0.5188	0.201	0.129	0.84	0.84	0.676	0.2882	0.6575	0.201	0.56	0.81	0.84	
1	mistral-large-2402	0.6548	0.5484	0.2792	0.1858	0.84	0.85	0.726	0.1265	0.8317	0.2792	0.62	0.88	0.84	
13	gemini-pro	0.6402	0.5636	0.2188	0.1838	0.91	0.79	0.676	0.4196	0.7471	0.2188	0.44	0.83	0.91	
14	anthropic.claude-v1	0.6387	0.4911	0.2244	0.1613	0.78	0.75	0.694	0.3326	0.4951	0.2244	0.53	0.84	0.78	
7	mistral-medium	0.6354	0.5039	0.3619	0.1382	0.82	0.51	0.722	0.1536	0.8218	0.3619	0.56	0.81	0.82	
16	anthropic.claude-v2	0.6345	0.4834	0.2	0.1454	0.79	0.82	0.642	0.2888	0.4977	0.2	0.56	0.74	0.79	
24	stabilityai/StableBeluga2	0.5284	0.4111	0.2915	0.1158	0	0.72	0.654	0.2123	0.8839	0.2915	0.61	0.85	0	
28	mistralai/Mixtral-8x7B-	0.5006	0.2774	0.189	0.0793	0	0.13	0.672	0.1155	0.7563	0.189	0.54	0.81	0	
27	tokyotech-llm/Swallow-	0.4712	0.5036	0.0927	0.1797	0.58	0.7	0.642	0.4803	0.8506	0.0927	0.49	0.85	0.58	
6	nitky/Superswallow-70b-	0.4314	0.354	0.0323	0.18	0.01	0.67	0.394	0.3085	0.8829	0.0323	0.13	0.49	0.01	
8	lightblue/qarasu-14B-chat-	0.3847	0.1437	0	0.0431	0	0	0.432	0.1157	0.4149	0	0.36	0	0	
31	stabilityai/japanese-	0.3732	0.2432	0.0117	0.0558	0.02	0.55	0.062	0.2975	0.7055	0.0117	0	0.29	0.02	
38	tokyotech-llm/Swallow-	0.373	0.3716	0.049	0.1623	0.48	0.28	0.454	0.3946	0.7811	0.049	0.29	0.73	0.48	
22	rinna/nekomata-14b-	0.3644	0.4375	0.0067	0.1651	0.77	0.42	0.494	0.3402	0.8663	0.0067	0.4	0.64	0.77	
42	stabilityai/StableBeluga-	0.3626	0.2965	0.0029	0.06	0	0.44	0.572	0.1893	0.8114	0.0029	0.44	0.77	0	
10	mistralai/Mistral-7B-	0.3505	0.1835	0.048	0.0641	0	0.11	0.474	0.082	0.5065	0.048	0.36	0.47	0	
18	elyza/ELYZA-japanese-	0.3278	0.1506	0	0.0668	0	0.15	0.128	0.1741	0.5352	0	0	0	0	
23	meta-llama/Llama-2-70b-	0.3004	0.0783	0	0.0117	0	0	0.152	0.0471	0.3373	0	0	0.76	0	
32	llm-jp/llm-jp-13b-instruct-	0.2947	0.4687	0	0.0059	0.89	0.01	0.928	0.5239	0.9229	0	0.99	1	0.89	
39	llm-jp/llm-jp-13b-instruct-	0.2927	0.4698	0	0.0016	0.91	0	0.93	0.5371	0.91	0	0.98	1	0.91	
17	elyza/ELYZA-japanese-	0.2908	0.1191	0	0.0748	0	0.08	0.014	0.1967	0.4681	0	0	0	0	

東工大・産総研で開発

Llama2ベース

Swallow-7B
Swallow-13B
Swallow-70B

Mistralから継続
Swallow-MS-7B

Mixtralから継続
Swallow-MX-8x7B

https://wandb.ai/wandb-japan/llm-leaderboard/reports/Nejumi-Leaderboard-Neo--Vmlldzo2MTkyMTU0

まとめ

大規模言語モデルの事前学習は、最大級のスパコンの全系を使っても何ヶ月もかかる

全系までスケーリングさせるためには、４種類の並列化手法を組み合わせる必要がある
・データ並列
・ZeRO
・パイプライン並列
・テンソル並列

「富岳」では12.5PFLOP/sが限界であった
・ノードあたりのメモリが32GB → ローカルバッチサイズが増やせない
・ラージバッチ問題 → データ並列数が増やせない (Weak Scalingできない)
・パイプライン並列とテンソル並列(Strong Scaling)に頼らざるを得ない

LLM-jp → スクラッチから日英で事前学習
Swallow → 最新のオープンソースモデルから日本語で継続事前学習