



大規模AIモデル訓練ワークロード における集団通信の高速化

NTT ソフトウェアイノベーションセンタ

研究員 立岩齊明

2024/07/29 電子情報通信学会 情報ネットワーク研究会 @ 札幌モエレ沼公園

目次



1. LLMの概要
2. 分散深層学習
3. 分散深層学習高速化
 1. 集団通信の最適化
 2. 光学的ネットワーク
 3. In-Network Aggregation
 4. 我々の取り組み

LLMの概要

LLM (Large Language Model)



多くのテキストデータを学習して、言語を理解し生成することができるシステム
事前学習による知識獲得 + 追加学習によるタスク適応

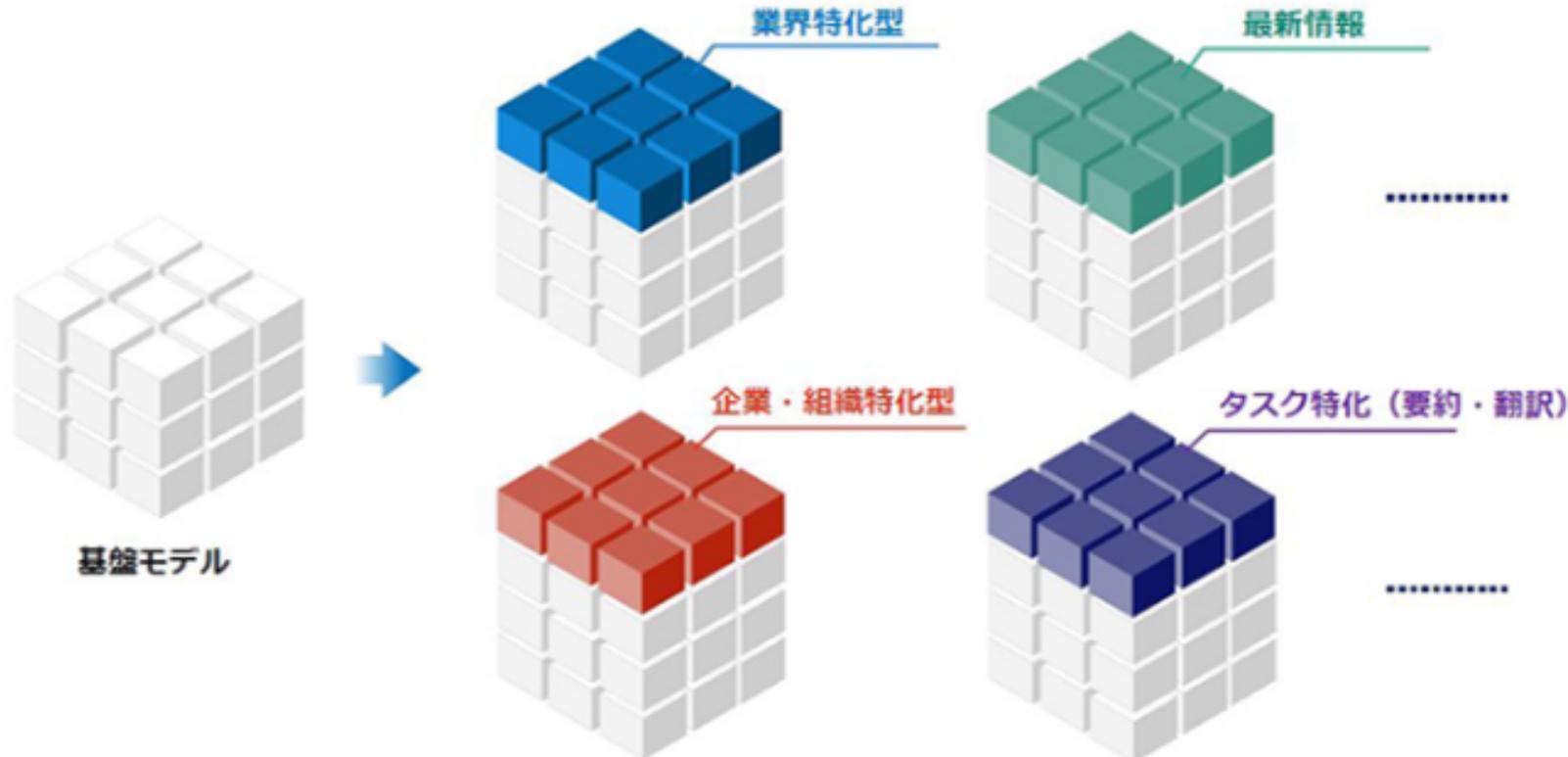


図: "NTT版大規模言語モデル「tsuzumi」 | NTT R&D Website." NTT R&D Website, 25 Apr. 2024,
www.rd.ntt/research/LLM_tsuzumi.html.

基盤モデル

大量のデータと計算資源を用いて事前学習し、**知識と推論能力**を付与したモデル

- 事前学習では与えられた文章から次に出現するトークン(≈ 単語)を予測するタスクを解かせる
- HuggingFace 🤗 プラットフォームを通して学習済みモデルを公開、ダウンロード可能
- 例えばLLaMA2モデルをもとに多くの派生モデルが作成

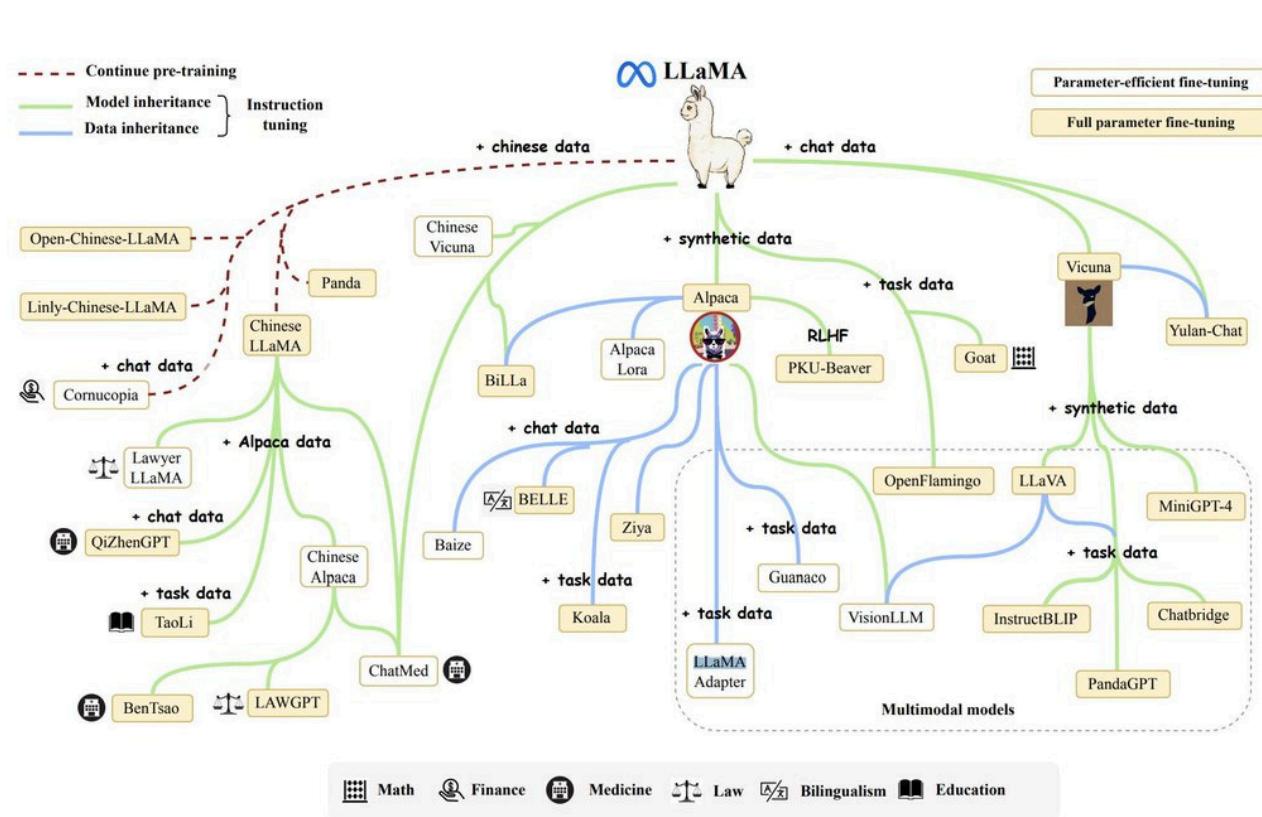


図: Guo, Charlie. "AI Zoology: Why are there so many animal models?" <https://www.ignorance.ai/p/ai-zoology-why-are-there-so-many>.

追加学習



基盤モデルをもとにタスクに適したデータを用いて追加で学習を行い、タスク、ドメイン知識に適応させる

- フルチューニング: モデルの全ての重みを更新
- 部分チューニング: モデルの一部の重みのみを更新
 - (右図) LoRA [Hu+, 2021]: もとのモデルに比べて少量のパラメタを追加しそれのみを学習させることで精度向上させる

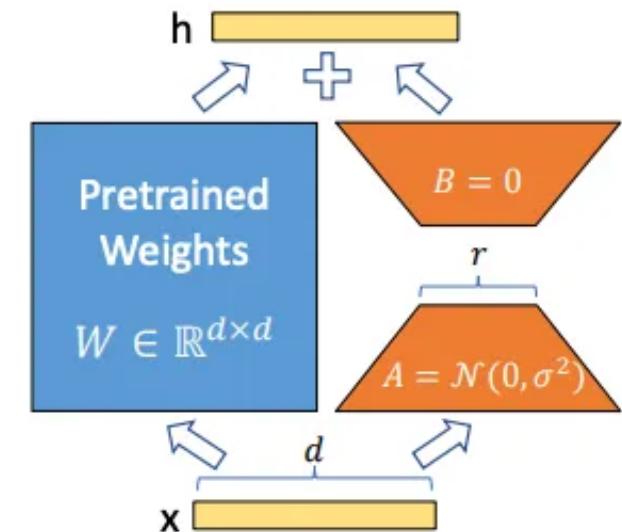
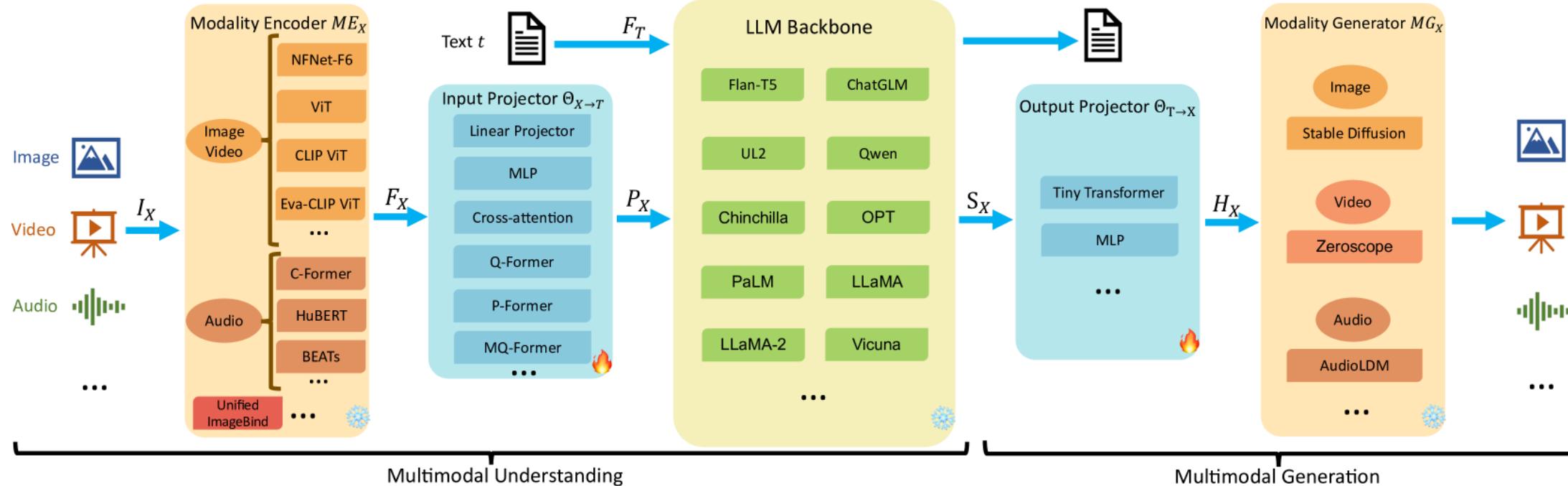


Figure 1: Our reparametrization. We only train A and B .

言語分野以外でのLLMの活用



- 画像、映像、音声などのマルチモーダルモデルの多くのコアにLLM [Zhang+, 2024]
- プロジェクタと呼ばれる機構(青色)がLLM用に入力と出力を変換
- LLMがモデルの大部分を占め、性能にも大きく影響



[Zhang+, 2024] Zhang, Duzhen, et al. "Mm-LLMs: Recent advances in multimodal large language models." arXiv preprint arXiv:2401.13601 (2024). Fig.2

LLM (Large Language Model) の特徴



- 主流の基盤モデルはTransformer[Vaswani+, 2017]をベースにした繰り返し構造であり、**モデルサイズの拡大が容易**
- モデルの細かい形状(FFN層の隠れ層次元など)が性能に与える影響は小さい
- モデルサイズ、データサイズ、計算資源の増加に対し、べき乗で性能向上、その限界は不明。またこれらの情報から、ある程度の**学習後モデルの精度予測ができる**(Scaling Law [Kaplan+, 2020], [Hoffmann+, 2022])
- この法則に伴い、基盤モデルサイズとデータサイズは拡大

[Vaswani+, 2017] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

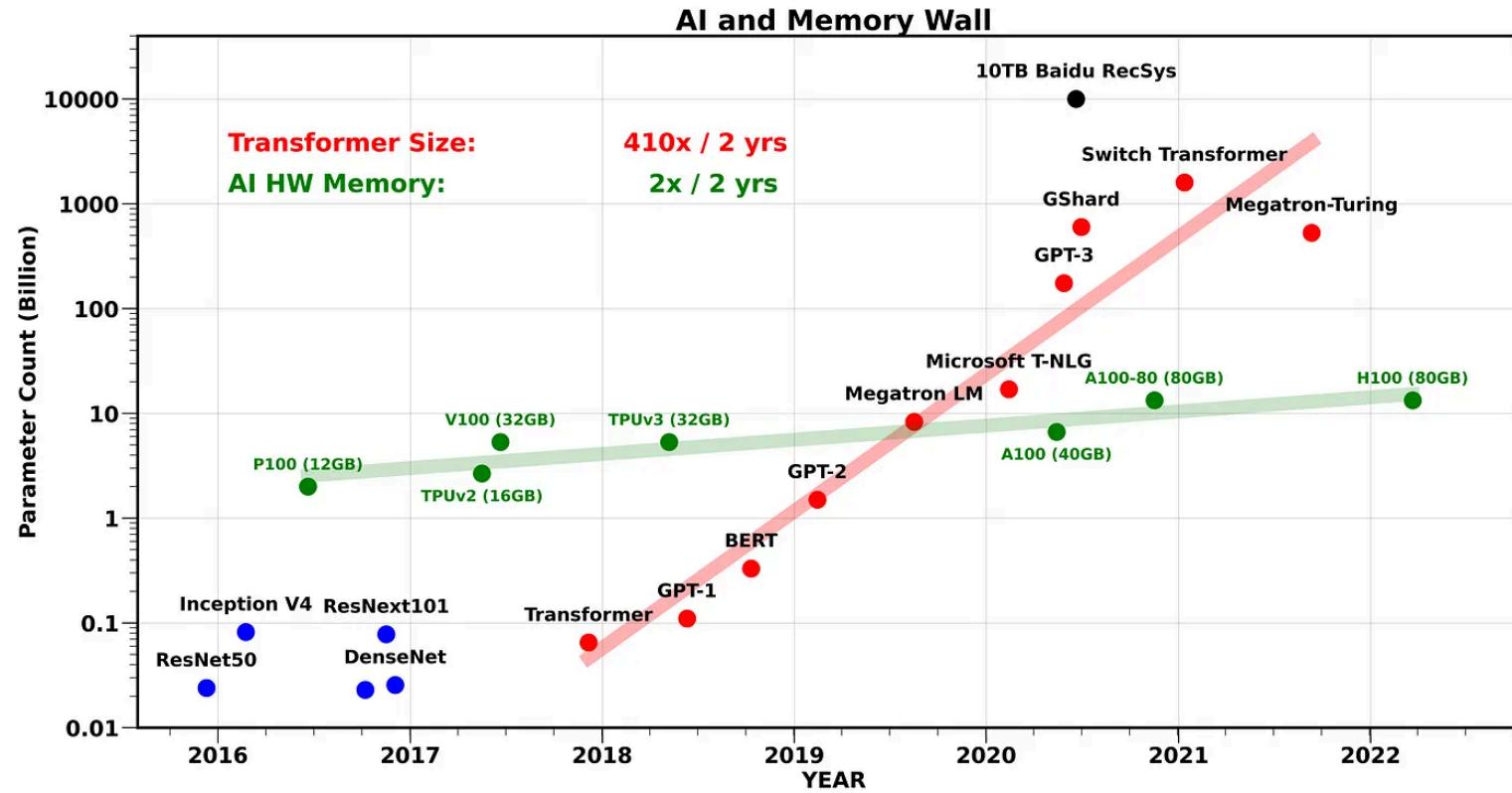
[Kaplan+, 2020] Kaplan, Jared, et al. "Scaling laws for neural language models." arXiv preprint arXiv:2001.08361 (2020). [Hoffmann+, 2022]

Hoffmann, Jordan, et al. "Training compute-optimal large language models." arXiv preprint arXiv:2203.15556 (2022).

Memory Wall [Gholami+, 2024]



- モデルサイズの拡大に対してメモリ容量とメモリ帯域幅の向上は緩やか
- メモリシステムがボトルネックとなり、Scaling Lawのハードウェア面での限界に

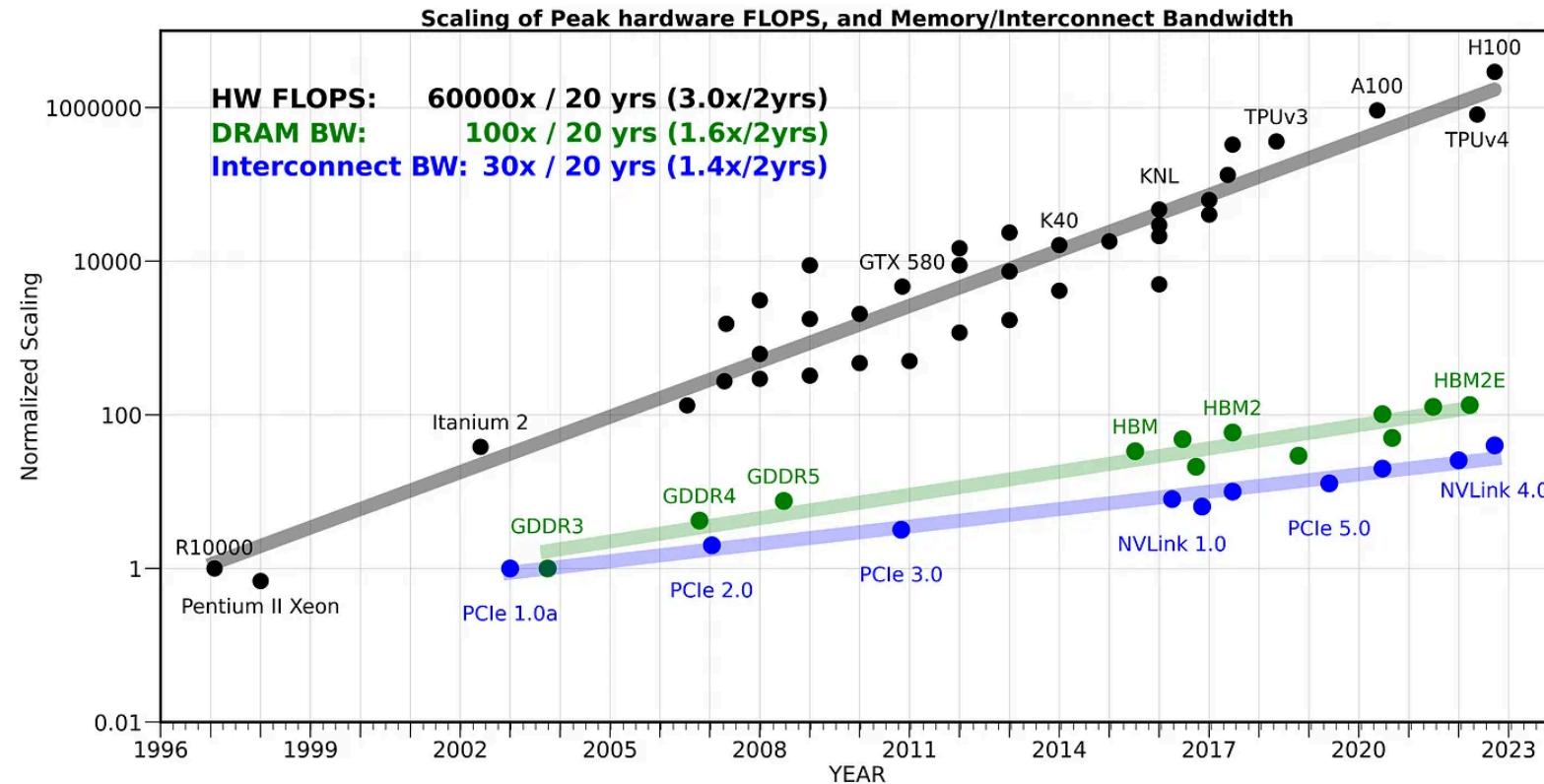


[Gholami+, 2024] A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney and K. Keutzer, "AI and Memory Wall," in IEEE Micro, doi: 10.1109/MM.2024.3373763.

Memory Wall [Gholami+, 2024]



- メモリ容量不足は分散処理によるメモリ分散で補えるが**GPU間通信**が発生
- 演算性能に対しメモリ、通信帯域の向上も乖離傾向であり**通信の効率化**が必要

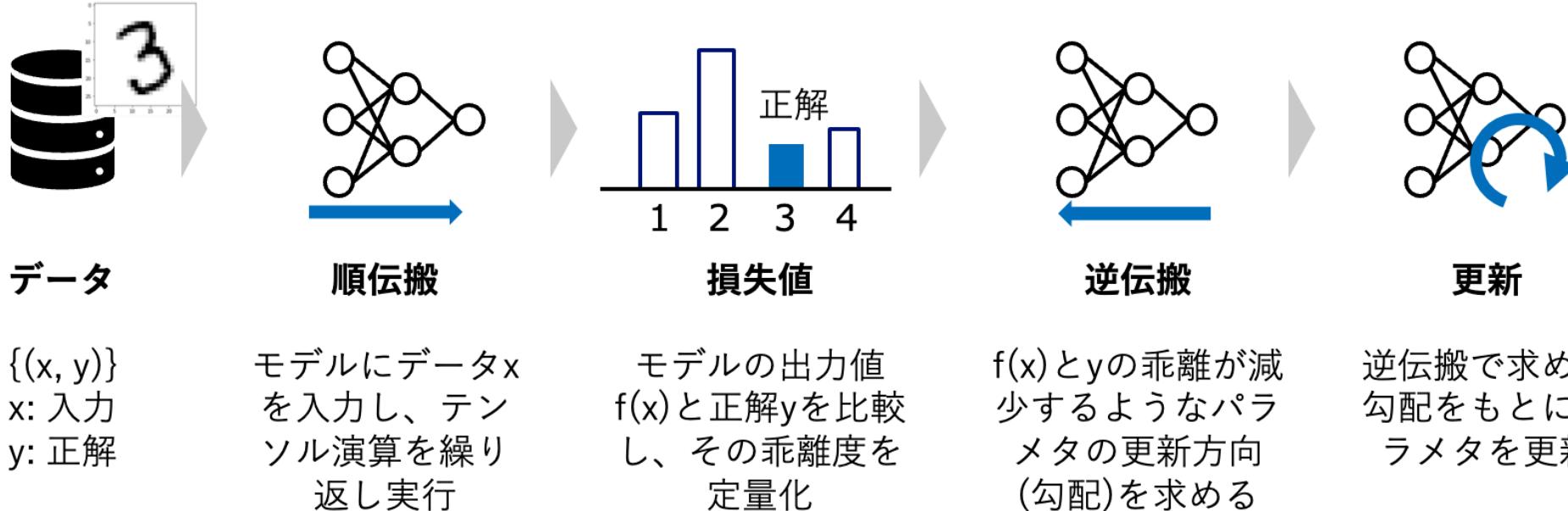


モデルの分散学習

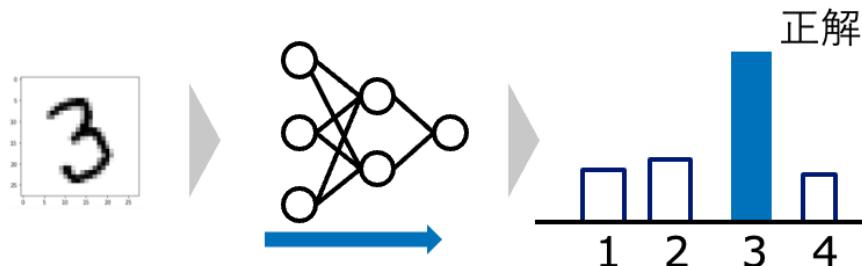
学習の基本手順



順伝播(損失関数の計算), 逆伝播(勾配の算出), パラメタ更新



学習後のモデル



分散深層学習(Distributed Deep Learning)



複数のGPUを用いて一つの大きなモデルを学習すること¹

- 訓練スループットの向上、 使用メモリの分散化
- GPU間の通信が発生

分散実行時は下記のような基本的な並列手法の組み合わせで行われる。

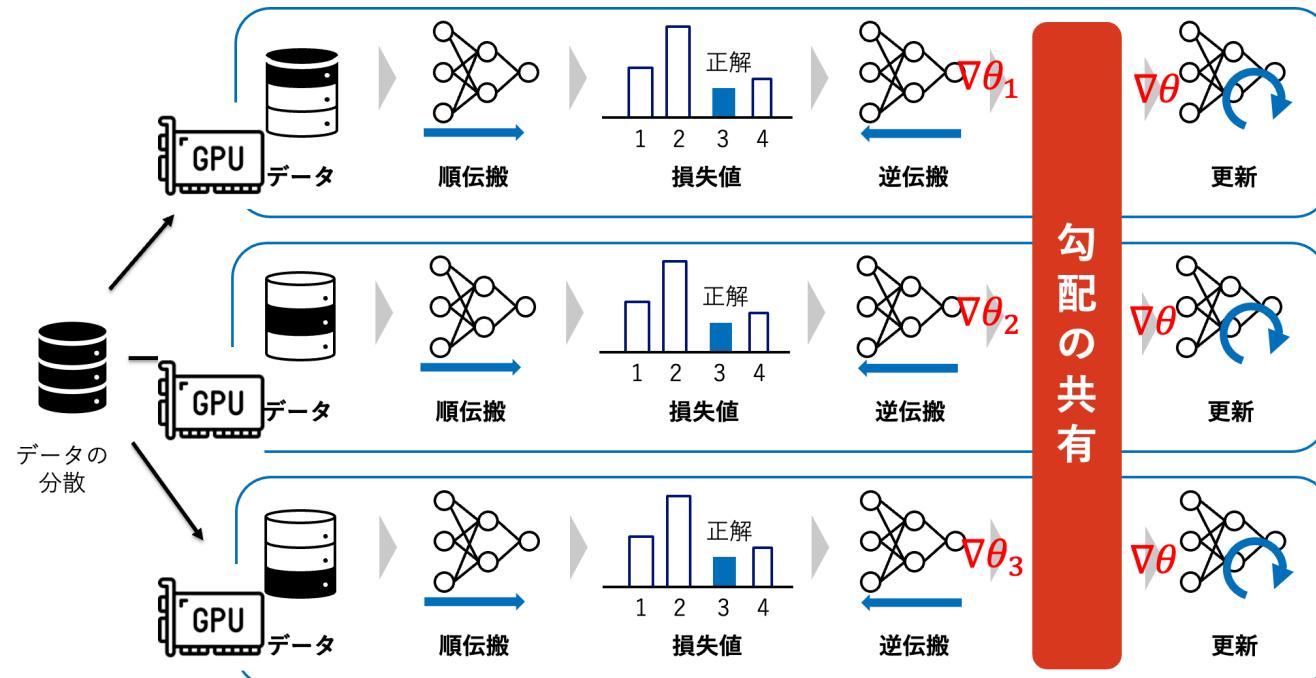
- データ並列
- テンソル並列
- パイプライン並列

1: 文脈によっては複数のモデルを分散して学習させる連合学習を指すこともある

データ並列

複数のプロセスがモデルの複製を保持、勾配計算を並列化

- Allreduceモデル¹: 勾配の集約値を全プロセスで共有し、モデルを更新
- LLMのような大規模モデルの台頭によって、メモリを節約するZeRO(FSDP)²に発展



1: Allreduceモデルの他にPSモデルもある。こちらはマスタが勾配を集約、モデルを更新し、更新後のモデルを分配

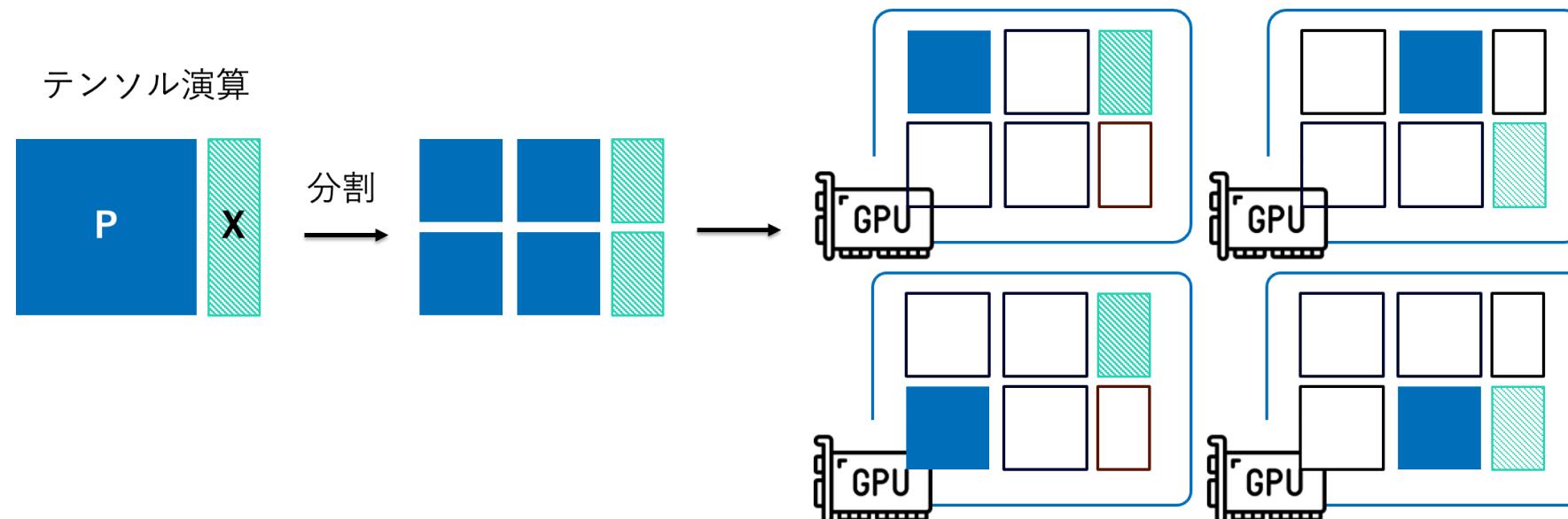
2: ZeRO(FSDP)はモデルで分散し、データ並列と同様の処理を行うように通信

テンソル並列



テンソル演算($Y = PX$)におけるパラメタP分割し複数のプロセスで保持、演算の一部をそれぞれのプロセスで実行し結果を都度結合

- 一定のテンソル演算ごとに結果の収集と、次のテンソル演算のための中間出力の分配のための通信が発生

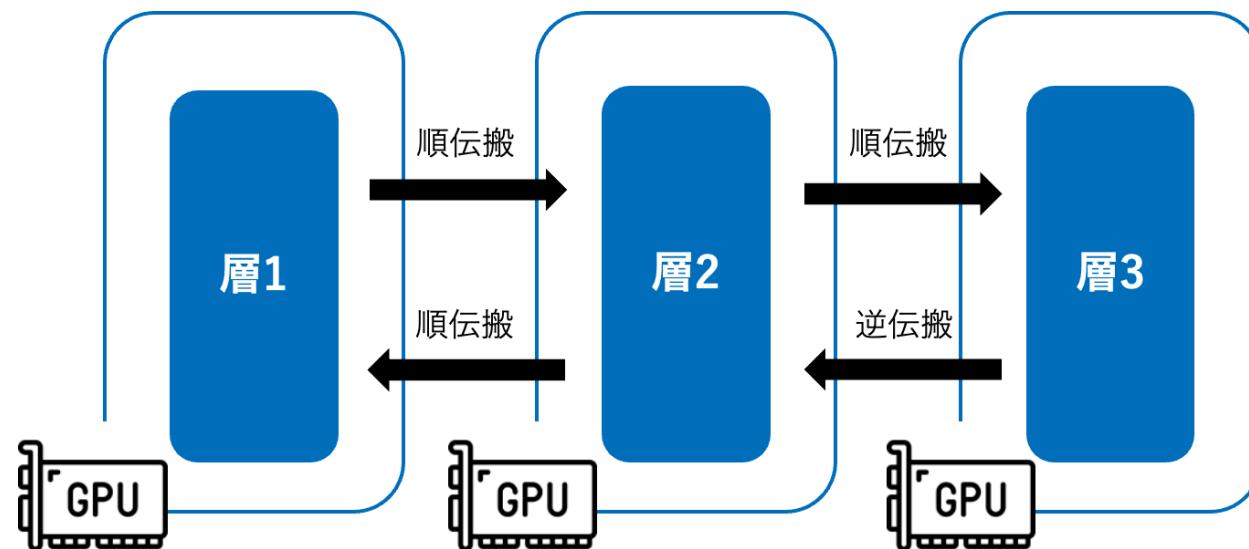


パイプライン並列



モデルを層単位で複数個に分割しプロセスに分配、中間出力をP2P通信で受け渡す

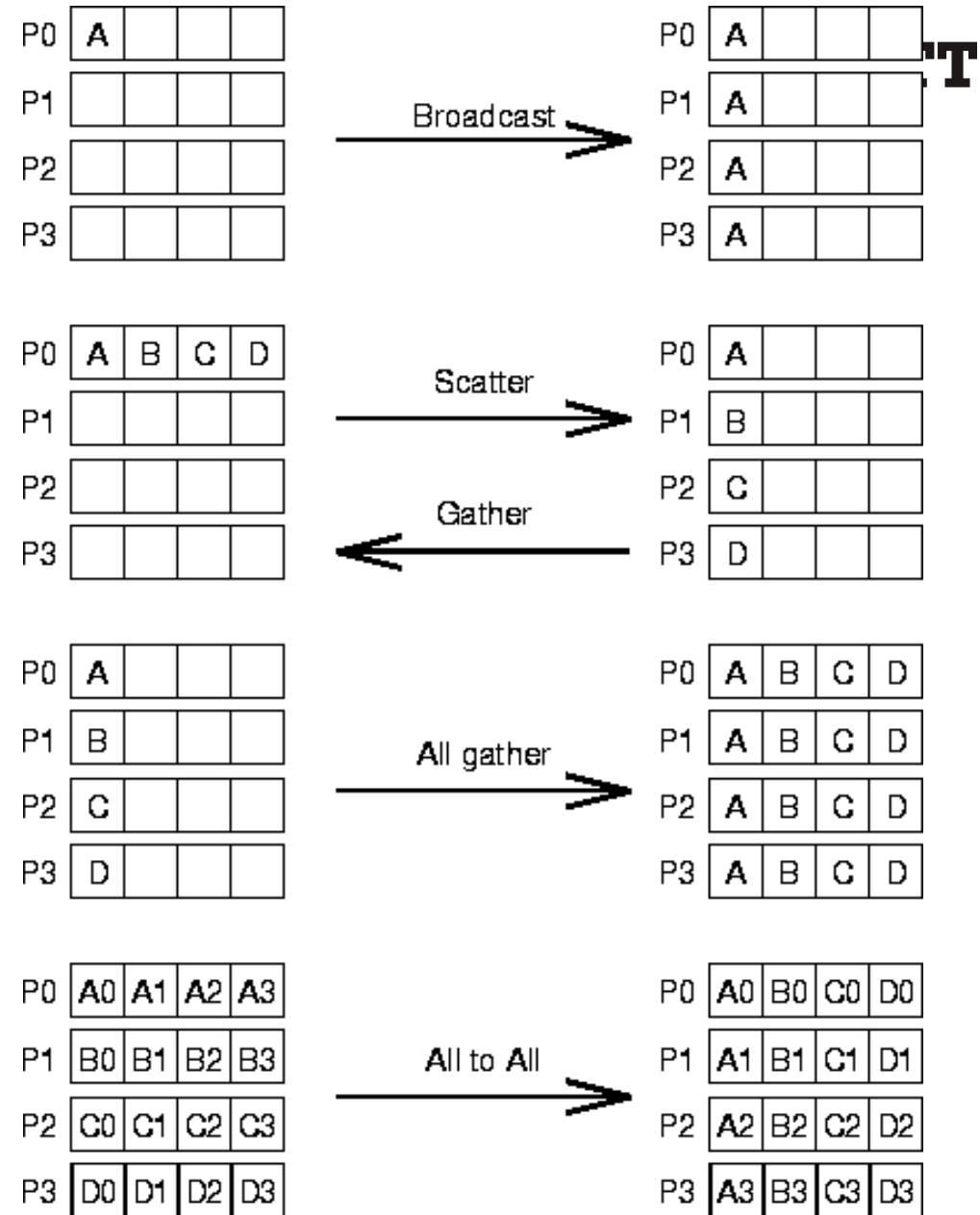
- GPU使用率を少しでも高めるために、入力データを細かく分割し、パイプライン化を行い、順伝播と逆伝播を行う



集団通信

並列手法では集団通信と呼ばれる通信パターンが発生する

- 複数のプロセスがデータ列を交換、集約する通信パターン
- MPI(Message Passing Interface)で規格
- 集団通信のための通信ライブラリ
 - MPI系**: OpenMPI, MPICH, MVAPICH
 - xCCL系**: Gloo, NCCL(NVIDIA), RCCL(RoCE), SCCL, MSCCL, TACCL



分散深層学習



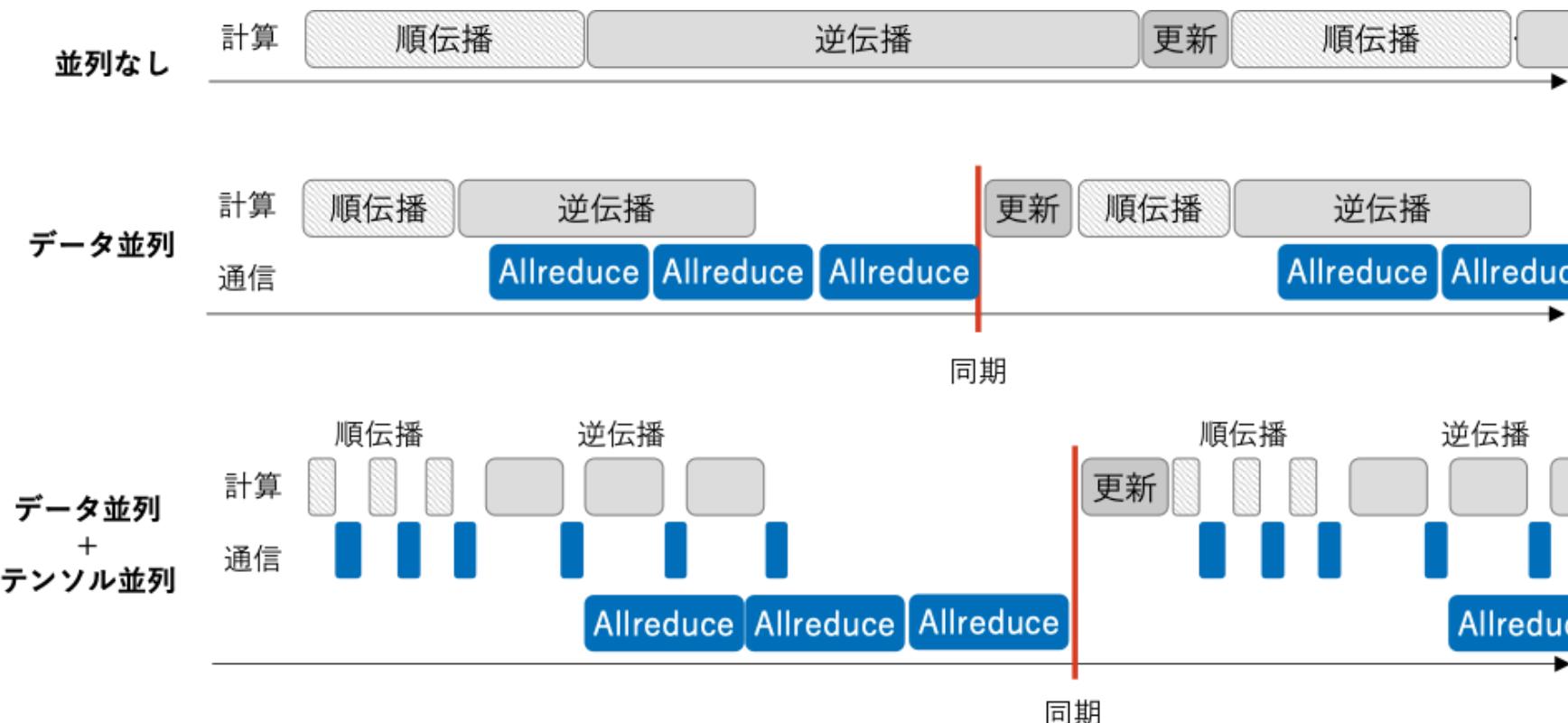
分散実行時は下記のような基本的な並列手法の組み合わせで行われる

並列方式	通信内容	通信方式	説明	ライブラリ
データ並列	勾配	Allreduce	データのみ分散	torch.DDP tensorflow.MWMS
テンソル並列	中間出力	Allreduce	テンソル重みを分散	torch.distributed.tensor Metagron-LM
パイプライン並列	中間出力	Send/Recv	モデルを分散	torch.distributed.pipeline fairscale.nn.Pipe

分散深層学習のワークロード



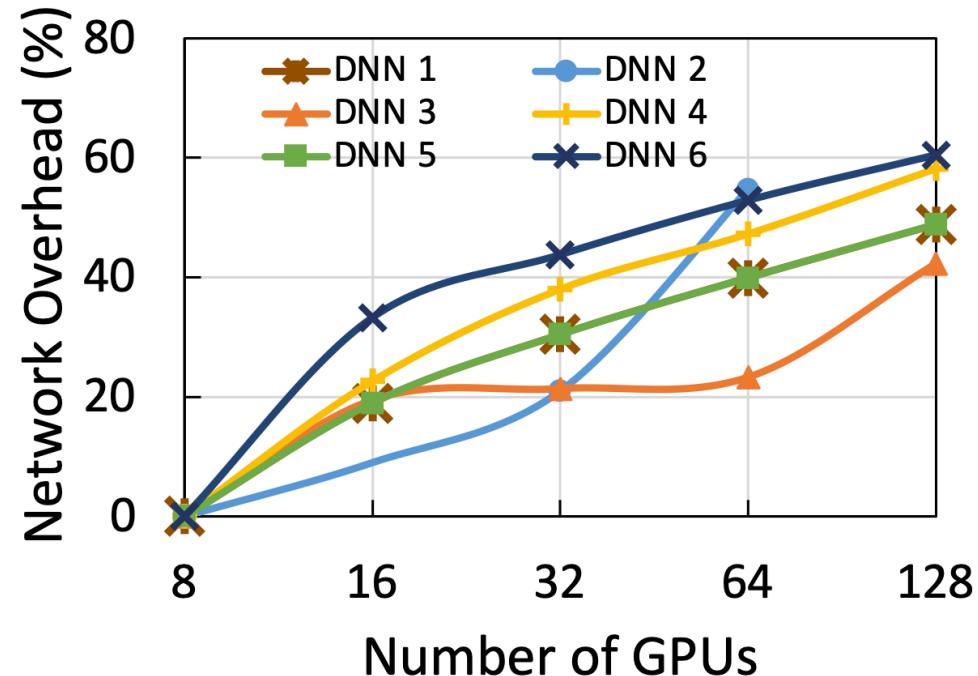
- データ並列では計算と通信のオーバーラップが可能、逆伝播後の勾配共有のためのAllreduce通信で同期待ちが発生
- テンソル並列では計算と通信のオーバーラップ不可
- テイルレンテンシに敏感、通信終了が遅れたプロセスあるとそこで律速する



分散深層学習のワークロード



- 通信がボトルネックになるかどうかは、モデルの規模、計算と通信環境に依存
- 一般的に、GPU数増加にしたがって、ワークロード中通信が占める時間の割合は増加
- 図はMetaで運用されている6つの深層学習モデルのワークロードに占める通信の割合(Network Overhead)を示したもの
 - この例では最大60%の通信オーバーヘッドが発生
- GPU等の演算性能の向上に従って、ネットワーク負荷は今後も継続的に増加

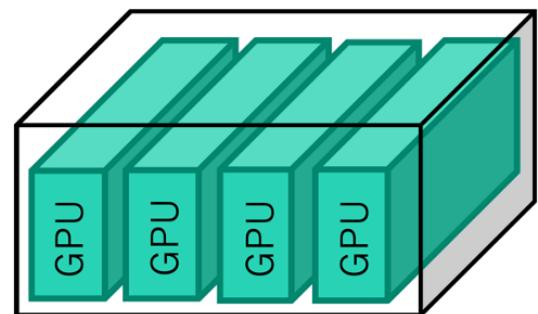


3D-Parallel

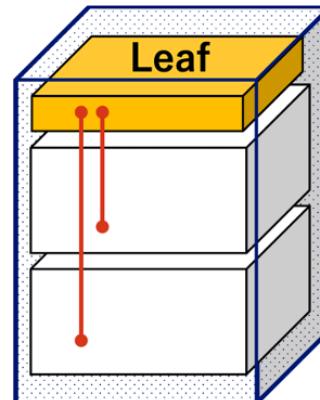


これらの並列化は大規模モデルの訓練の場合組み合わせて使用される

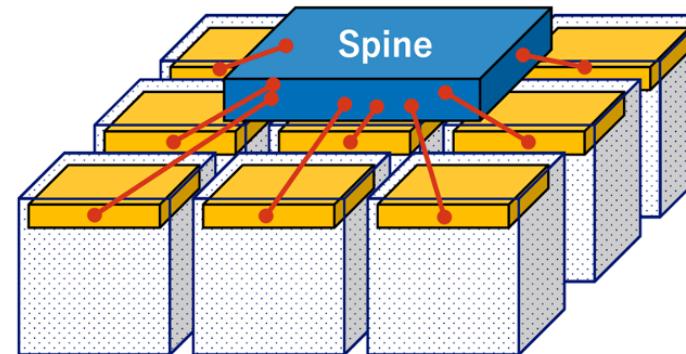
- ノード内でテンソル並列
- ノード間でパイプライン並列
- ラック間でデータ並列、ZeRO (FSDP)



ノード内
テンソル並列



ラック内
パイプライン並列



ラック間
データ並列

高速な通信帯域を使用し
通信での律速を軽減

パイプライン並列による
メモリ分散

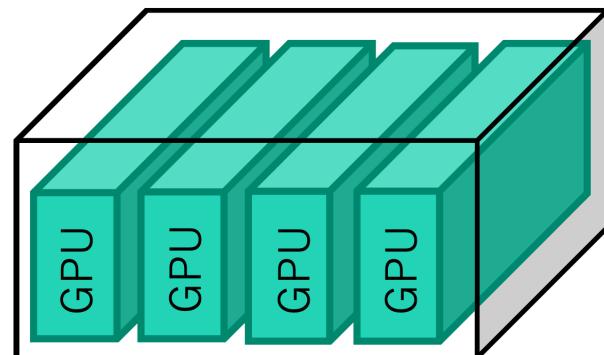
スケールアウトが容易な
データ並列をラック間で使用

3D-Parallel



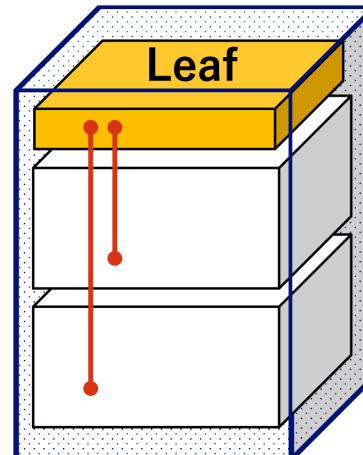
より小さい環境では...

- ノード内でテンソル並列
- ノード間でデータ並列、ZeRO (FSDP)



ノード内
テンソル並列

高速な通信帯域を使用し
通信での律速を軽減



ラック内
データ並列およびZeRO

データ並列もしくは、ラック内の(比
較的高速な)帯域を用いてZeRO並列

並列手法の良い組み合わせのためにはモデルのアーキテクチャと実行環境に応じたチューニングが必要

- GPU間の通信帯域は局所的に異なる
 - ノード内、ノード間、ラック間
 - クラウドのマルチテナント環境であればジョブにどのGPUリソースが割り当てられるか、他のジョブがどれほど帯域を使用しているかによって使用可能な最大帯域が異なる
- GPUの性能(FLOPS, VRAMメモリ)も異なる場合がある
- 現状は人が手動でチューニング

どのようなインフラが必要か?



- **高い計算能力**

- transformer, 具体的にはattentionは「入力系列長の2乗に比例して計算量が増大」
- 混合精度計算、量子化(8bit, 4bit)計算のサポート

- **高いメモリ容量**

- 訓練時はパラメタ量 θ に対し 16θ 程度、一時データをVRAMに保持する必要あり(勾配、モメンタム、アクティベーション(テンソル演算結果))
- メモリの断片化も、大きな場合は30%のメモリ損失

- **高い通信性能**

- 集団通信($\text{GPU} \Leftarrowtail \text{GPU}$)、チェックポイントの保存($\text{GPU} \rightarrow \text{SSD/HDD}$)

- **高い可用性、移植性**

- システムが巨大になるほど、故障率が高く、訓練が止まった場合の損失も大きい
- 異なる環境、異なる設定でも継続動作

分散深層学習ワークロード高速化のために



キーワード

効率的な通信アルゴリズム × 低遅延高帯域で高機能なネットワーク

それぞれの進展だけでなく、協調的な最適化が必要

以下では

- 集団通信の最適化
- 低遅延広帯域 光回路スイッチネットワーク
- ネットワーク内集約(in-network aggregation)

の話題を取り上げる

→ 集団通信の最適化

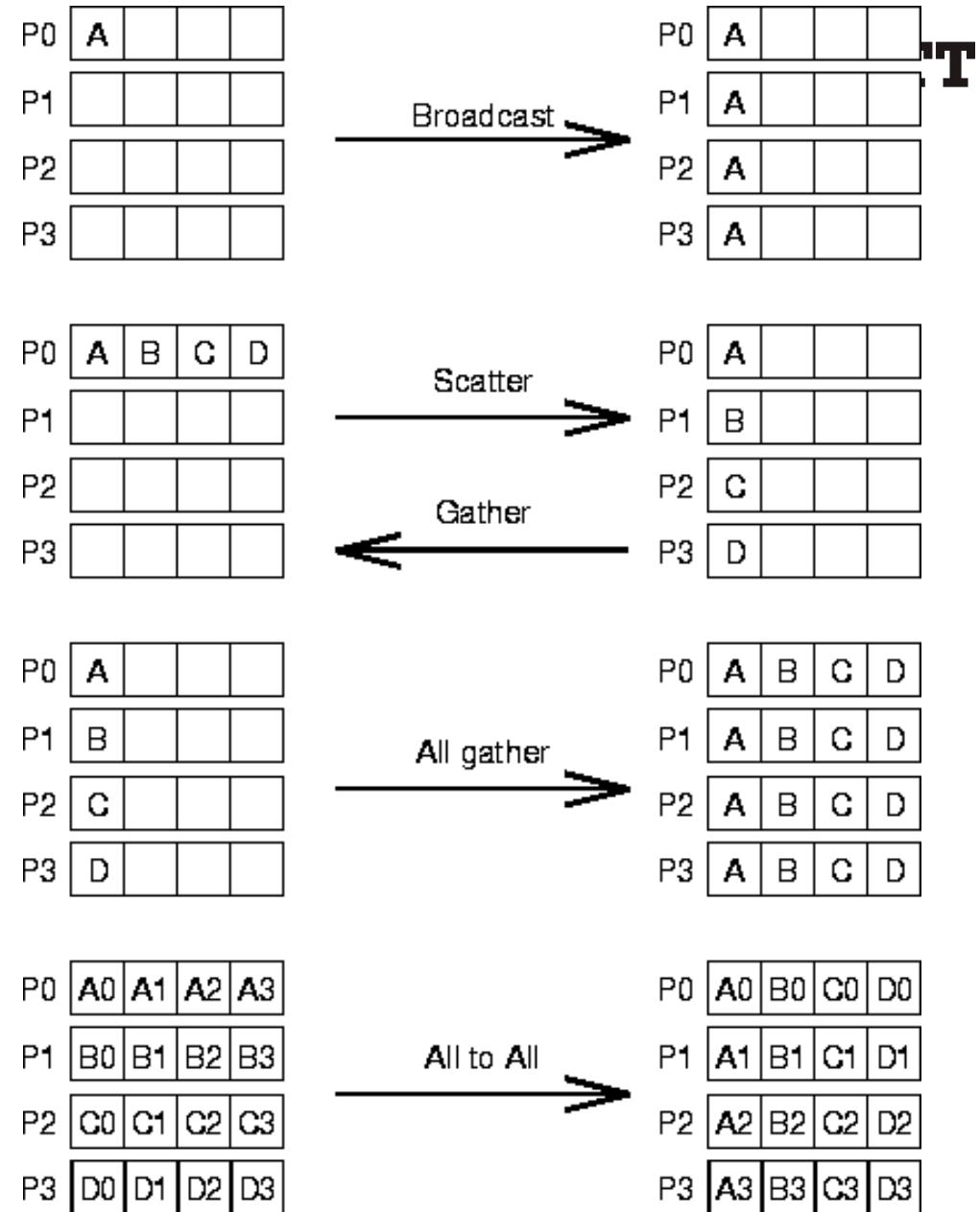
光回路スイッチネットワーク

ネットワーク内集約(In-network aggregation)

集団通信(再掲)

並列手法では集団通信と呼ばれる通信パターンが発生する

- 複数のプロセスがデータ列を交換、集約する通信パターン
- MPI(Message Passing Interface)で規格
- 集団通信のための通信ライブラリ
 - MPI系**: OpenMPI, MPICH, MVAPICH
 - xCCL系**: Gloo, NCCL(NVIDIA), RCCL(RoCE), SCCL, MSCCL, TACCL



集団通信の最適化



MLワークロードと通信ネットワークに合わせたアルゴリズムの最適化

- 既存のアルゴリズムは、どのようなネットワークトポロジでも平均的に効率よく動作するように設計
- 分散深層学習ワークロードでは発生する通信が決定的¹であり、与えられたGPU間接続に応じて最適化(チューニング)の余地がある

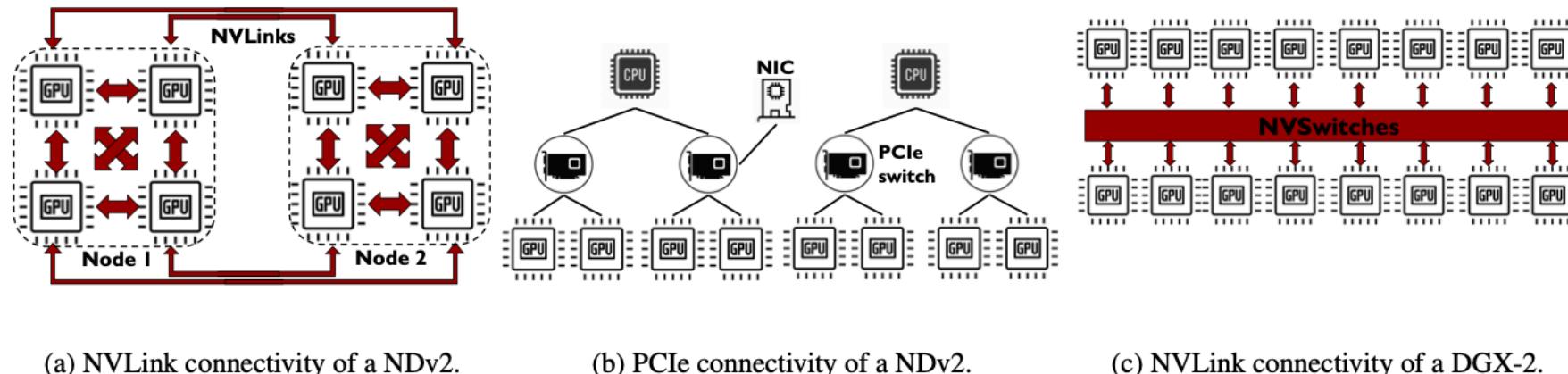


Figure 5: Aspects of physical topologies in various GPU systems.

1: 並列戦略の組み合わせによってワークロード内での通信リクエストが決定される

図: [Shah+, 2023] Shah, Aashaka, et al. "{TACCL}: Guiding Collective Algorithm Synthesis using Communication Sketches." 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23). 2023.

集団通信の最適化



- 目的: 集団通信全体のデータ転送時間と遅延の総量(もしくはどちらか)を最小化する
- 変数: GPU間の1対1通信スケジュール
 - どのデータを、どこからどこへ、どのタイミングで送信するか
- 入力: 集団通信の種類、データサイズ、GPU間接続トポロジ、通信帯域、遅延
- 手法: 組合せ最適化問題として定式化しソルバを用いて求解
 - Blink (全域木パッキング) [Wang+, 2020]
 - SCCL (SMT; Satisfiability Modulo Theories) [Cai+, 2021]
 - TACCL (混合整数計画問題) [Shah+, 2023]

[Wang+, 2020] Wang, Guanhua, et al. "Blink: Fast and generic collectives for distributed ml." Proceedings of Machine Learning and Systems 2 (2020): 172-186. [Cai+, 2021] Cai, Zixian, et al. "Synthesizing optimal collective algorithms." Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. 2021. [Shah+, 2023] Shah, Aashaka, et al. "{TACCL}: Guiding Collective Algorithm Synthesis using Communication Sketches." 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23). 2023.

例: 集団通信の最適化 > MSCCL



Microsoftが管理するオープンソース集団通信ライブラリ

- NCCL¹ライブラリをベースとし、集団通信のアルゴリズムのみをSCCLの手法で最適化したものに置き換える
- 例えばPytorchコードを変更せずにSCCLで最適化された集団通信を使用できる

```
# Change NCCL to SCCL Runtime
RUN cd ${STAGE_DIR} && \
    git clone https://github.com/pytorch/pytorch.git && \
    cd pytorch && \
    git checkout tags/v1.9.0 -b v1.9.0_sccl && \
    # NCCLのリポジトリURLをMicrosoftのMSCCLのリポジトリURLに置き換え
    perl -p -i -e 's/url = https:\/\/github\.com\/NVIDIA\/nccl/url = https:\/\/github\.com\/microsoft\/msccl/g' .gitmodules && \
    git submodule sync third_party/nccl && \
    ...
```

1: NVIDIA Collective Communication Library

2: コード: <https://github.com/microsoft/msccl/blob/b23e9cd5dd63f82ee1c5aae7e0a2042079be903a/dockerfiles/Dockerfile#L65C1-L80C32>

集団通信の最適化

→ 光回路スイッチネットワーク

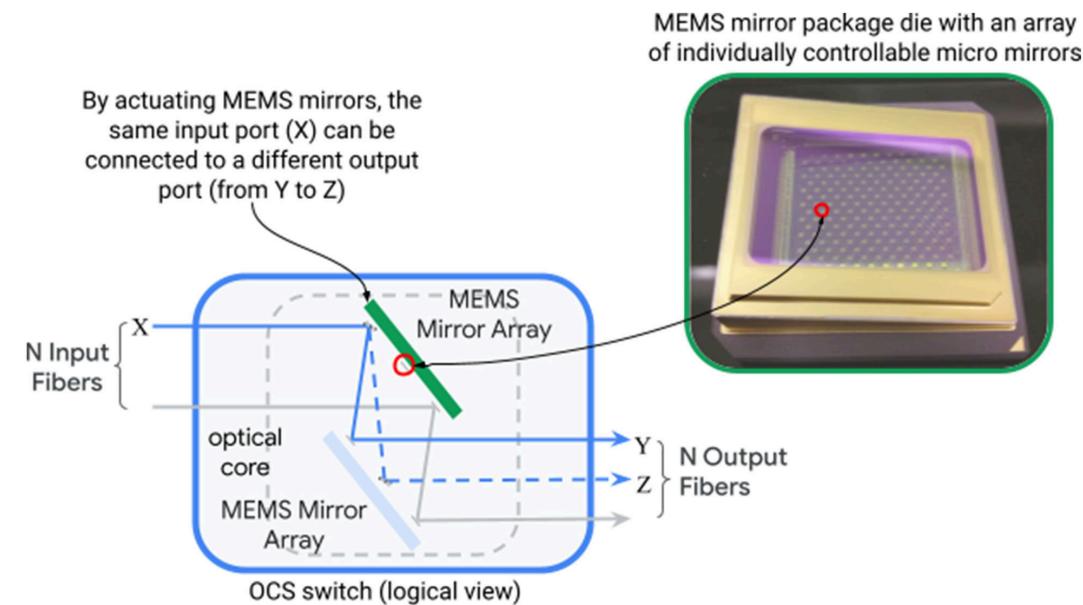
ネットワーク内集約(In-network aggregation)

光回路スイッチ (OCS)



高帯域、低遅延、低消費電力ネットワーク
を実現する次世代ネットワークスイッチ

- OCS¹: 入力ポートと出力ポート間を電気信号に変換せずに光信号のまま接続
- MEMS方式では(多くの場合)鏡面で光を反射させポート間を接続、鏡面の角度を変えることで接続を変更²
- デメリット: 接続先が限定される、接続の変更に一定の時間が必要



1: OCS (Optical Circuit Switch)

2: MRR(Micro-Ring Resonator)を用いた他の実装方法もある

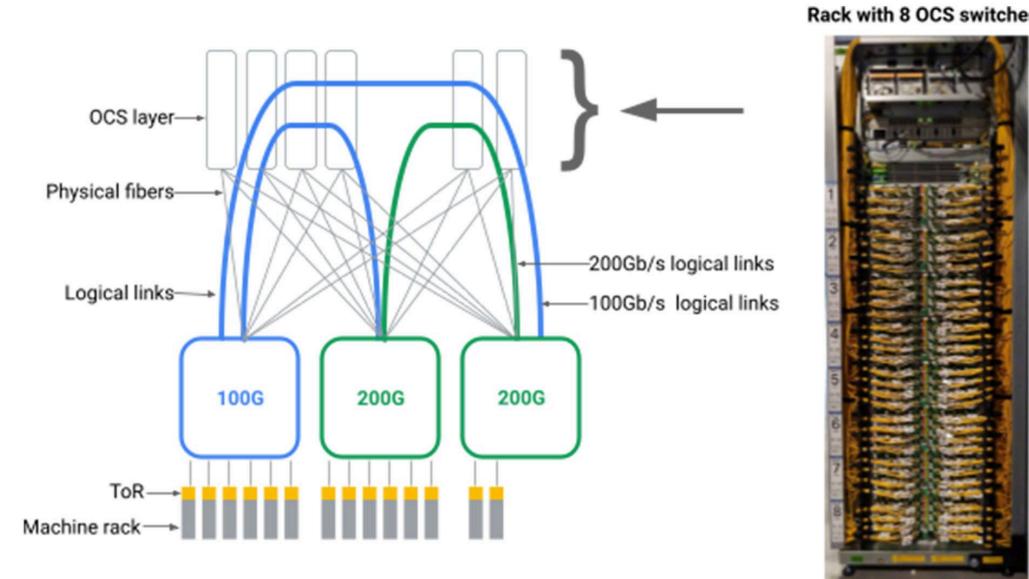
3: OEO (Optical-Electric-Optical)

例: Google Jupiter Datacenter



OCSを導入した初の大規模データセンタ

- Leaf-SpineネットワークのSpine層をOCSに置き換え
- 異なるデータレートの混在を許容し環境のアップデートを逐次的に行える、トランシーバ数の削減による故障率の低下



例: Google TPUv4

可用性に優れたAI/ML用ハードウェア

- TPUユニット(アクセラレータ)を銅線と光ケーブルで接続
- OCSを通してTPUユニットの柔軟にネットワークトポロジを構築可(3Dトーラス等)
- 故障ユニットを避けてトポロジを再構築することで可用性が向上
- 複数の並列規模の異なるジョブに対してトポロジを形成できる

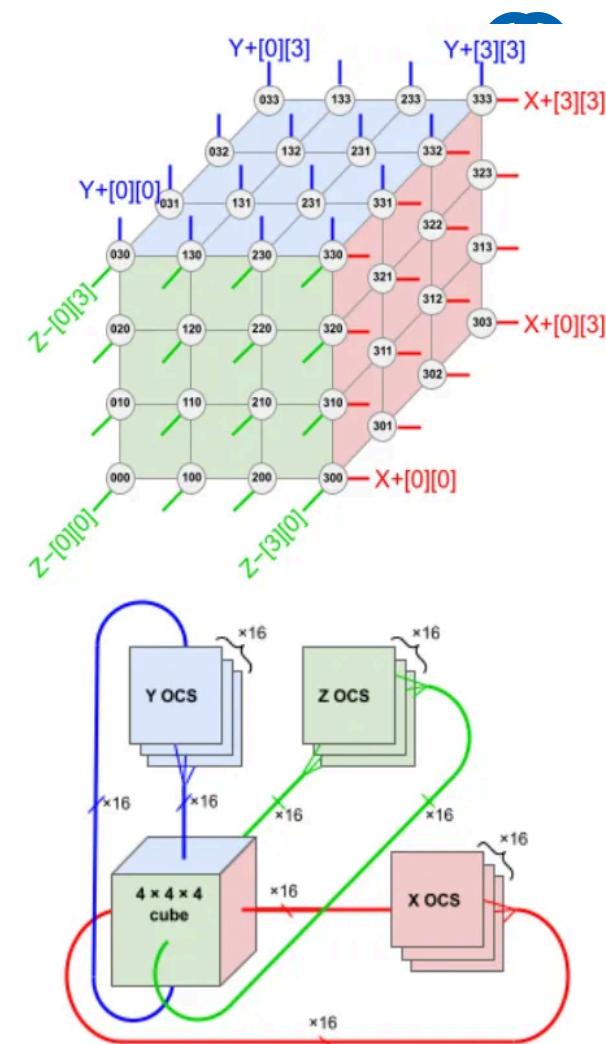


Figure 1: Connectivity of a $4 \times 4 \times 4$ cube (top) to 3 OCSes (bottom). The “+” and “-” connections with the same dimension and index are connected to the same OCS; 48 of these in-out pairs each connect to a distinct OCS.

光回路スイッチネットワークでの分散深層学習

- 光回路スイッチネットワークの**利点**
 - 高帯域性→分散AI訓練時の大容量フローの高速化に有利
 - 高可用性→大規模実行時の故障率低減、GPUの利用率向上
 - 低遅延性→推論の高速化に有利
- 光回路スイッチネットワークの**欠点**
 - 限られた接続性→電気スイッチのように柔軟にパケットルーティングが行えない。直接の接続がないホスト間の通信には複数ホストを経由する(host-forwarding)必要がある
 - 接続変更に時間がかかる→ワークロードの途中での接続切り替えは不適

分散深層学習の場合、事前に通信パターンは決定される。そのため、ジョブ実行前に通信パターンにネットワークトポロジを適応させることで、これらの欠点を緩和

例: TopoOpt [Wang+, 2023]



MLワークロードの通信パターンに合わせたネットワークトポロジの最適化

- 光回路スイッチ(OCS)にGPU/Serverが直接結合されたハードウェアを想定
- Allreduce集団通信とP2P通信からなるML通信パターンに適合するようにトポロジを重ね合わせ

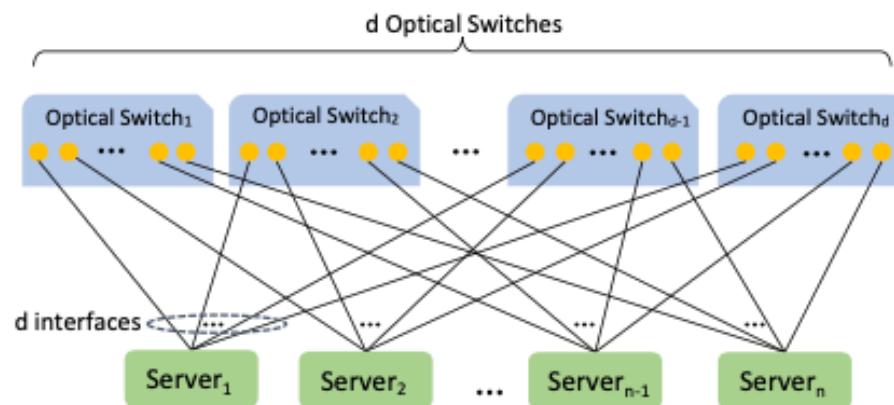


Figure 5: Illustration of TOPOOPT's interconnect.

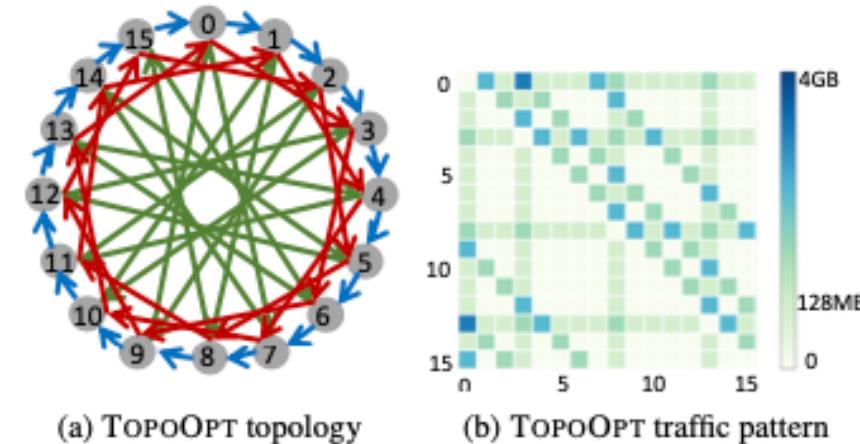


Figure 9: TOPOOPT's topology and traffic matrix.

集団通信の最適化

光回路スイッチネットワーク

→ ネットワーク内集約(*In-network aggregation*)

ネットワーク内集約(In-network aggregation) NTT

- 集約計算をノード(GPU)ではなく、ネットワーク機器にオフロード(委託)する
- Switch-based: ネットワークスイッチで集約計算を行う
- Middlebox-based: 既存のネットワークスイッチに集約機能を持つネットワークファブリック(デバイス群)を追加
- NIC-offloading: 演算機能を持ったNIC(SmartNIC, DPU)に集約計算をオフロード

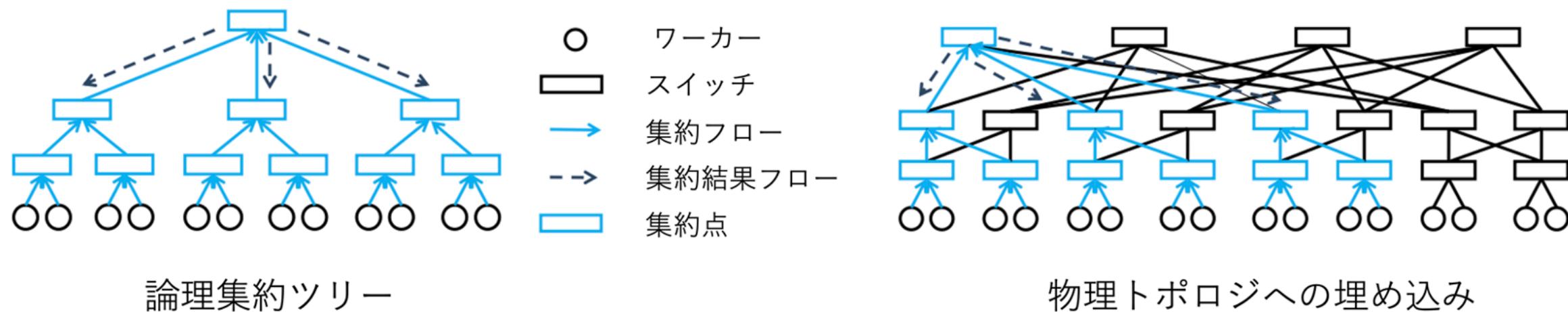


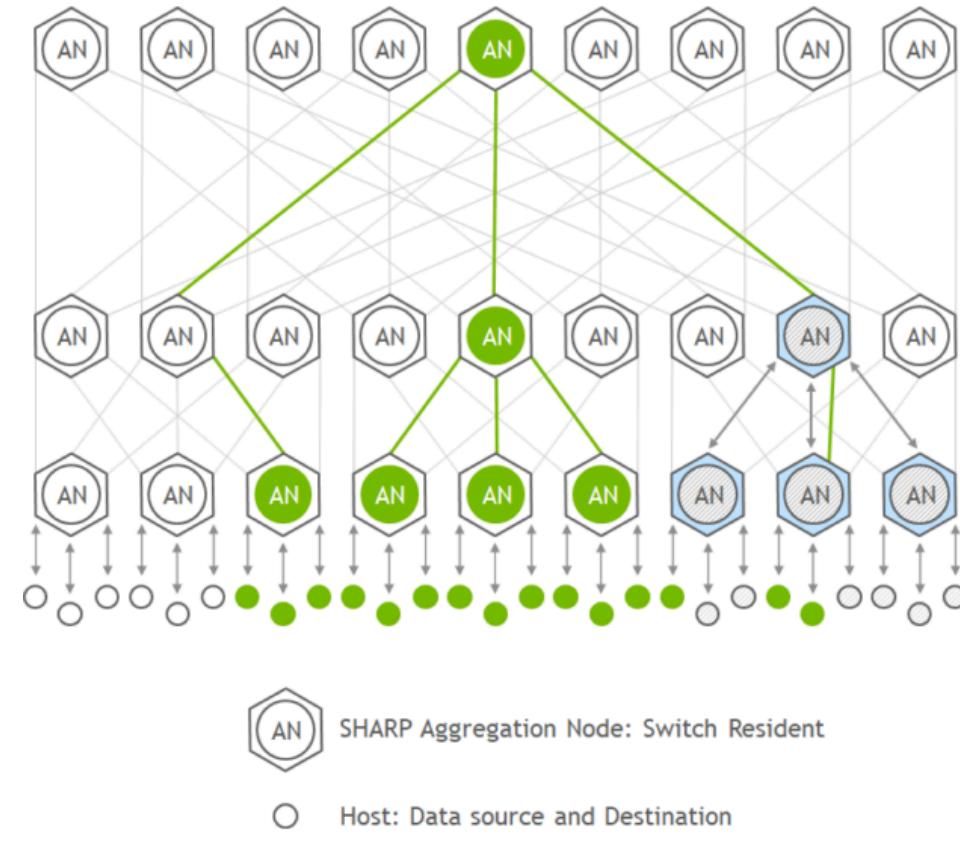
図: [Feng+, 2023] Feng, Aoxiang, et al. "In-network aggregation for data center networks: A survey." Computer Communications 198 (2023): 63-76. を改変

例: NVIDIA SHARP



In-network aggregationの実装例¹

- NVIDIA Quantumスイッチで使用可能
スイッチ内集約機能 [Graham+, 2020]
- ホストを葉、スイッチを節とする木を構成
 - 1: スイッチでデータを集約しながら上位ノードに送信
 - 2: 根でグローバルな集約値を取得後、逆向きの経路で集約値を放送



1: 知る限り唯一の製品化された実装

[Graham+, 2020] "Scalable hierarchical aggregation and reduction protocol (sharp) tm streaming-aggregation hardware design and evaluation." ISC High Performance 2020 (2020)
図: <https://developer.nvidia.com/blog/accelerating-cloud-native-supercomputing-with-magnum-io>

ネットワーク内集約によるAllreduceの高速化



- Allreduceはデータ並列、テンソル並列実行時に現れる通信形式
- ネットワーク内集約によりAllreduceのデータ転送時間と遅延が理論上の下限に
- N: ワーカー数、K: 勾配データのサイズとすると次の表の通り

	Ring Allreduce (従来のアルゴリズム)	Tree Allreduce (従来のアルゴリズム)	ネットワーク内集約
データ転送量	$2((N-1)/N)K$ $\approx 2K$	$2((N-1)/N)K$ $\approx 2K$	K
遅延	$\mathcal{O}(N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(1)$

我々の取り組み

我々の取り組み



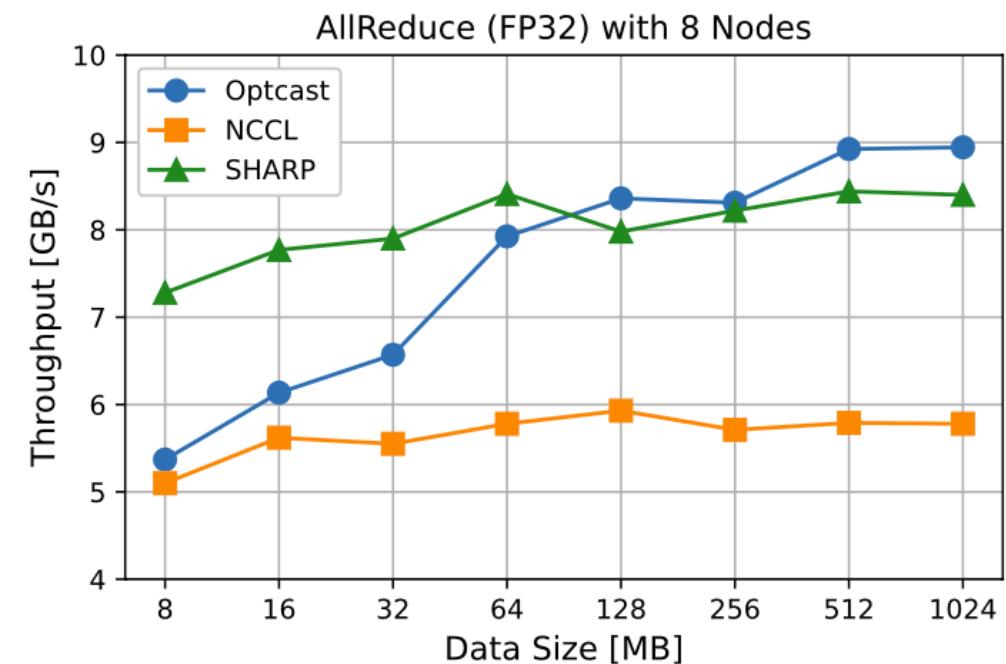
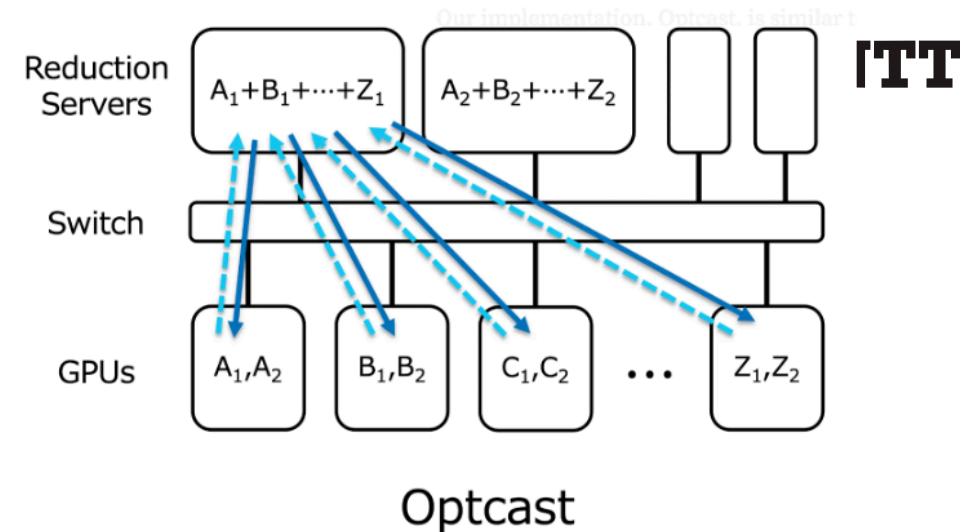
ネットワーク内集約を用いたAllreduce集団通信高速化

- 対象ネットワーク(NW)	ネットワーク内集約先	効果
1 電気スイッチNW	CPU	データ転送量の削減
2 光回路スイッチNW	SmartNIC	遅延時間の削減
3 光回路スイッチNW	集約デバイス (CPU, FPGA, SmartNIC, etc.)	データ転送量の削減 必要な集約デバイス数の削減

1: Optcast

余剰CPUに集約計算をオフロードさせ
Allreduceを高速化 (OCP¹ Regional
Summit, 2024)

- NCCL(デファクトのGPUライブラリ)を拡張
- NVIDIA Quantumスイッチ不要
 - プロプライエタリの回避
- SHARPよりも多くの通信プロトコルに対応
 - RoCEv2, InfiniBand, AWS EFA, ...
- NCCLと競合性能(右図下)

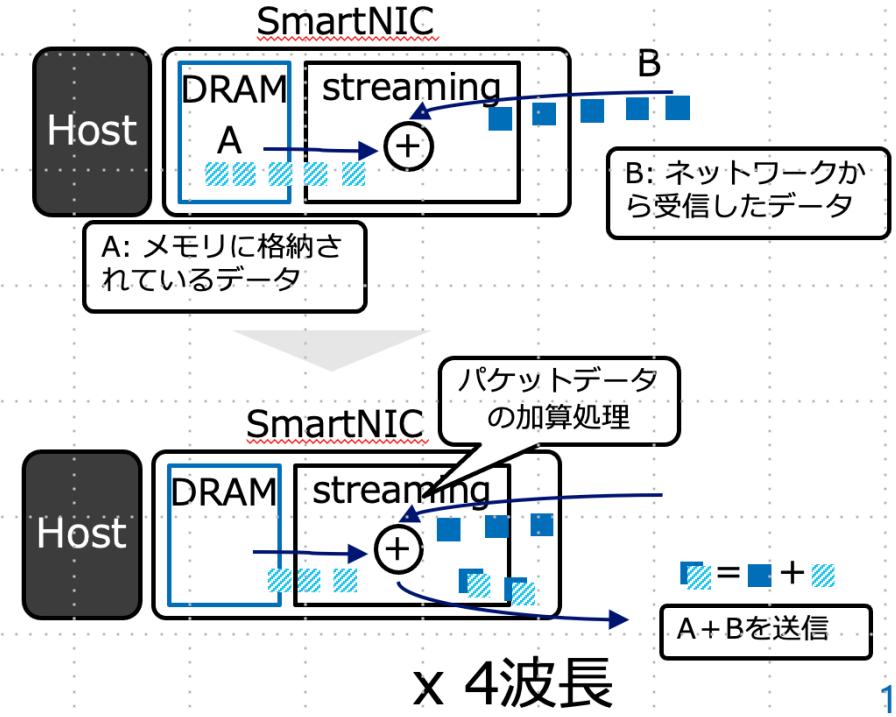


1: OCP ... Metaを発起人とする、データセンタ技術のオープンナレッジ化規格化の業界有数の団体

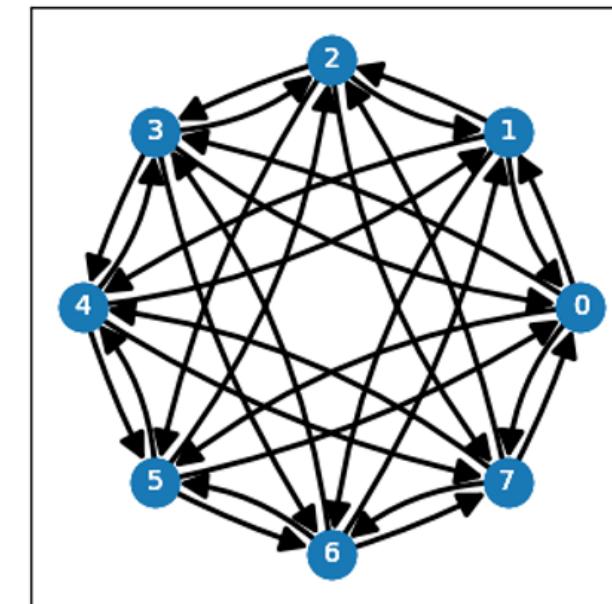
2: SmartNICへの集約オフロード+トポロジ探索

SmartNICで集約計算をストリーム処理させ、さらに同時に同時にネットワークトポロジとAllreduceアルゴリズムを最適化することで遅延削減

SmartNICでの約ストリーミング



最適化されたトポロジ



10

2: SmartNICへの集約オフロード+トポロジ探索

- 既存のAllreduceアルゴリズム最適化の定式化に「SmartNICでのストリーム集約計算」「ネットワークトポロジの最適化」の要素を追加
- 組み合わせ問題(混合整数線形計画法)で定式化
- データ転送回数の削減に伴う遅延削減

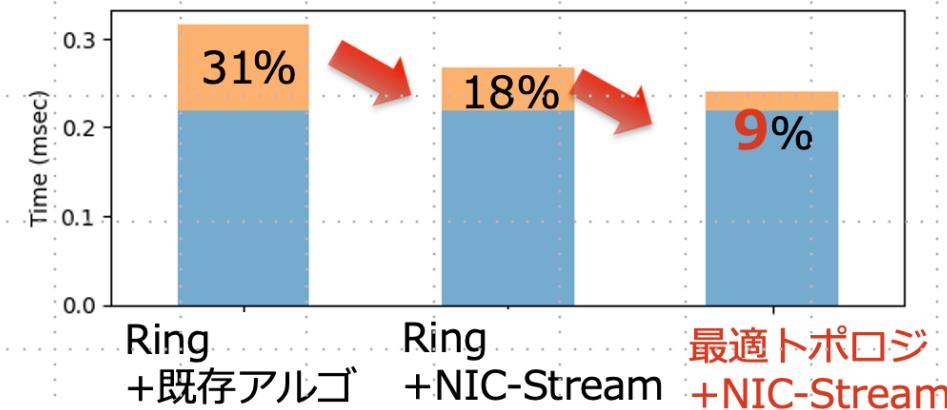
	トポロジ	アルゴリズム	転送回数	転送時間 [Lb*]
(既存) Ring-Allreduce	Ring	Ring-Allreduce	14	0.4375
Ring + NIC-Streaming	Ring	最適化	7	0.4375
最適トポロジ+NIC-Streaming	最適化	最適化	3	0.4375

*L … Allreduceデータサイズ [bit]

*b … 帯域の逆数[1/bps]

- 25MB-Allreduceにおいて遅延が占める割合
 - 31% → 9%に削減 (1.6Tbps帯域)
 - 10% → 2%に削減 (400Gbps帯域)

1.6Tbps帯域での 25MB-Allreduce通信時間:
転送時間(青), 遅延(黄)



3: 光回路スイッチネットワーク+INA



光回路スイッチNWへのネットワーク内集約(INA)を導入する初の試み

目的

- ・ 「低遅延、高帯域、高可用性に優れた光回路スイッチNW」で「ネットワーク内集約」を用いてAllreduceを高速化できるか
- ・ スイッチの空きポートに集約計算が可能なデバイス(=集約デバイス)を追加で導入
 - 現状のOCSでは原理上集約計算負荷

問題

- ・ 既存のIn-Network Aggregation方式をそのまま光回路スイッチNWに用いると、場合によっては**使用するGPU数の数倍にも及ぶ、多くの集約デバイスが必要**
 - 光回路スイッチの限られた接続性により、既存のツリー型の集約プロトコルの構築には多くの集約デバイスが必要

3: 光回路スイッチネットワーク+INA



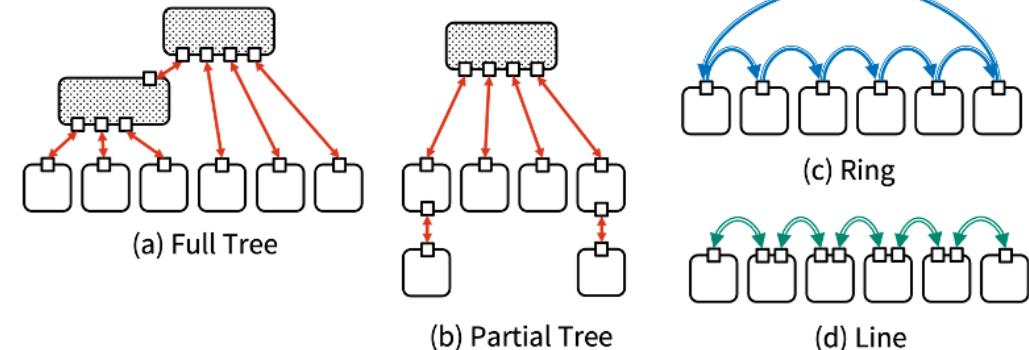
光回路スイッチNWのような直接結合網ではネットワークトポロジがAllreduceアルゴリズムの性能に大きく影響

- ネットワークトポロジとAllreduceアルゴリズムのペアを決めると、最終的なアルゴリズム性能(データ転送時間、遅延時間)が決まる
- トポロジを決める→そのトポロジにおけるAllreduceアルゴを最適化させる...を繰り返すのはどうか?
 - 取りうるトポロジの個数は膨大(ホストの個数、次数(=ポート数)に応じて指数的)
 - Allreduceアルゴ最適化は計算コストが高い(36ホスト環境で1day ~ 1week)
- ネットワークトポロジとAllreduceアルゴを全列挙するのは現実的な時間では不可能

3: 光回路スイッチネットワーク+INA

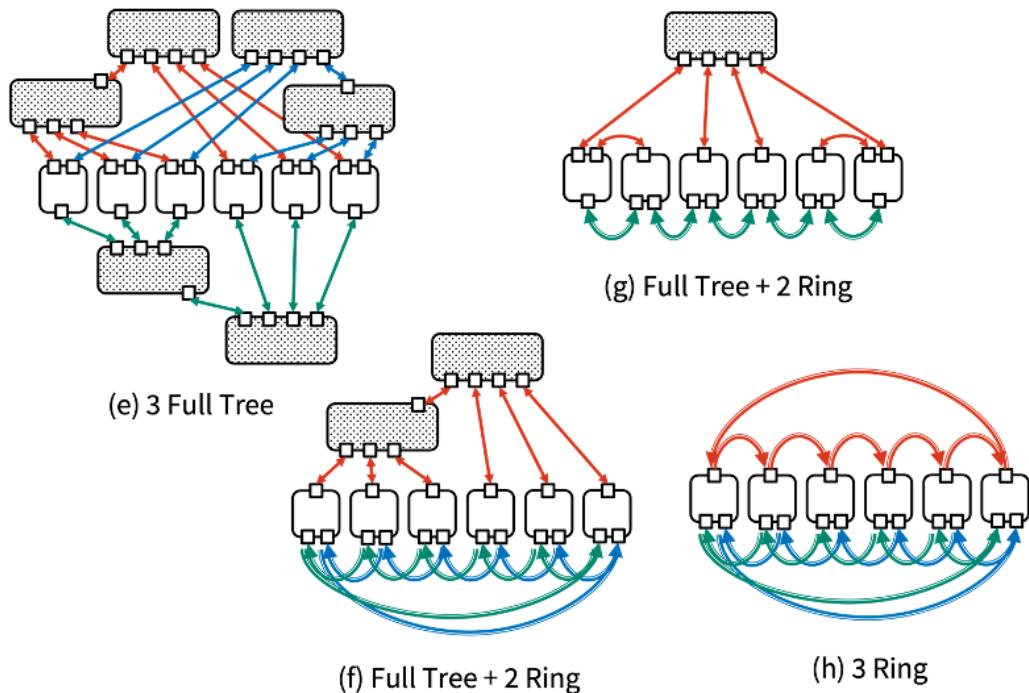
- Allreduceアルゴリズムを定義した小トポロジを複数用意
 - 集約デバイスを含ないもの、1つ含むもの、2つ含むもの...
- スループットが最大になるように小トポロジを組み合わせ
 - 小トポロジごとにデータ量を変えてAllreduceストリームを流す
- 多次元ナップサック問題に定式化
 - 集約デバイス数、workerのインターフェイス数の上限を考慮

Step 1: Topology Enumeration



Step 2: Topology Synthesizing:

Topology Selection and Worker Re-assignment

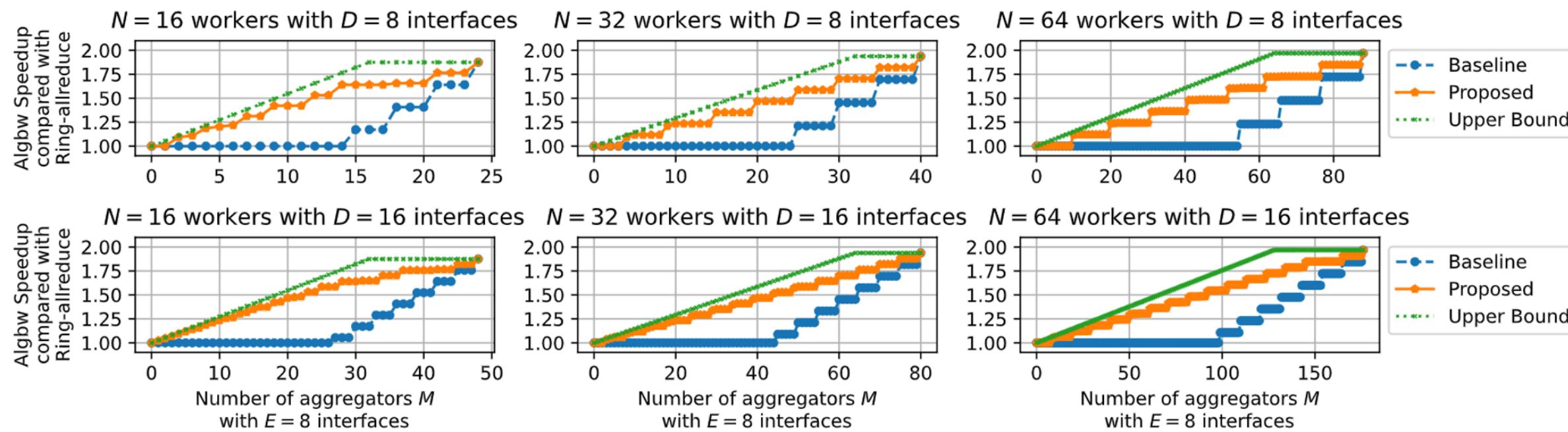


3: 光回路スイッチネットワーク+INA



アルゴリズムの性能評価 (データ転送時間 = Algorithm Bandwidth)

- 既存の手法(Baseline)よりも
 - データ転送時間の削減のために最低限必要な集約デバイスの個数を削減
 - 同等のデータ転送時間の削減を達成するのに少ない集約デバイス数で済む



まとめ



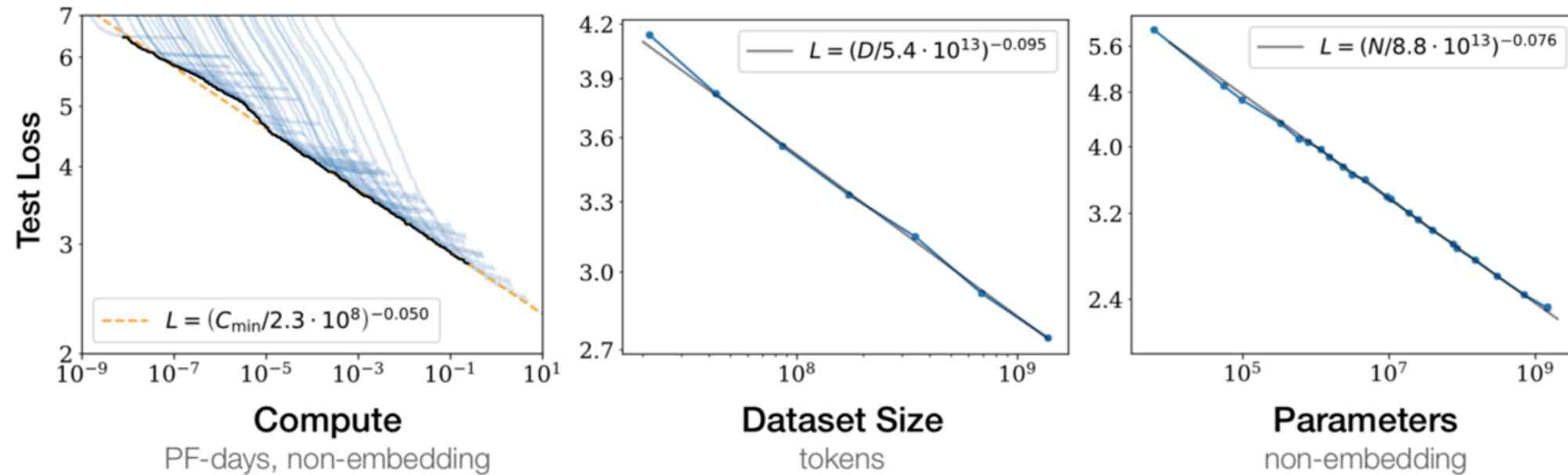
- 言語、マルチモーダルモデルの精度向上のために基盤モデルサイズは増加
- 事前学習、追加学習ともに順伝播/逆伝播の際のメモリ分散、スループット向上のための分散処理は必須
- 分散時に GPU間の(集団)通信が発生、演算性能の飛躍的な向上に伴い、ネットワークへの負荷偏好は今後も増加
- 集団通信アルゴリズムとネットワークアーキテクチャの進化が必要
 - 集団通信アルゴリズムの最適化
 - 低遅延、高帯域、高可用性な光回路スイッチネットワーク
 - ネットワーク内集約
- これらの協調的進化により、AI/MLのワークロードを高速化

Appendix

Scaling Law [Kaplan+, 2020]



- Transformer (言語モデル) の性能はパラメータ数(P), データサイズ(D), 計算資源(C)を変数としたべき乗則に依る
- またモデルサイズとデータサイズの関係にも言及:
 - $D \geq (5 \times 10^3)P^{0.74}$: パラメタを8倍した場合、データサイズは4.7倍が良い



ChinChilla Scaling Law [Hoffmann+, 2022]



- 400種類のモデルと学習トークン数の組み合わせで実験し、学習に必要なトークン数を回帰
- モデルとデータサイズは1:20が良いという結果に、これは[Kaplan+, 2020]とは異なる
- 多くの事前学習モデルでデータサイズが足りていないことを示唆

Parameters	FLOPs	FLOPs (in Gopher unit)	Tokens
400 Million	1.92e+19	1/29,968	8.0 Billion
1 Billion	1.21e+20	1/4,761	20.2 Billion
10 Billion	1.23e+22	1/46	205.1 Billion
67 Billion	5.76e+23	1	1.5 Trillion
175 Billion	3.85e+24	6.7	3.7 Trillion
280 Billion	9.90e+24	17.2	5.9 Trillion
520 Billion	3.43e+25	59.5	11.0 Trillion
1 Trillion	1.27e+26	221.3	21.2 Trillion
10 Trillion	1.30e+28	22515.9	216.2 Trillion

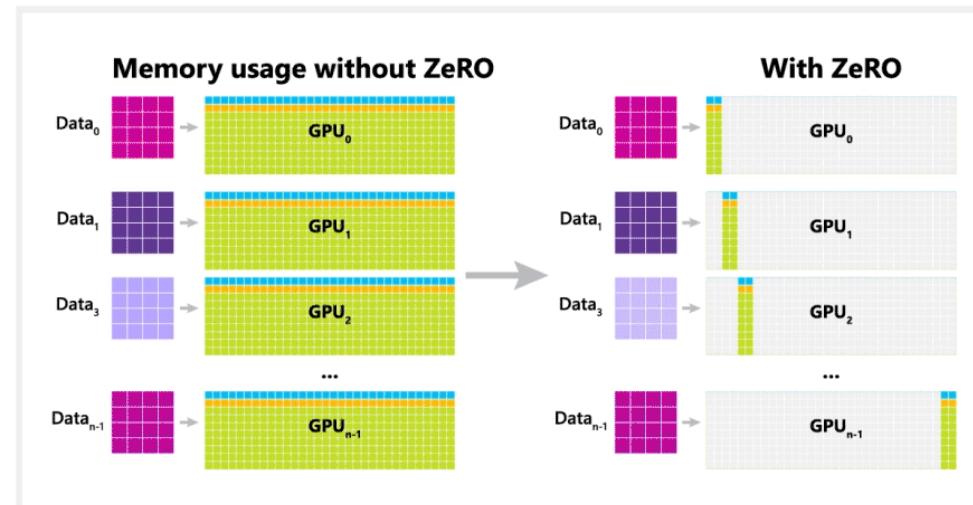
GPT-3のサイズなら
3.7Tトークン必要
(実際は300B)

ZeRO (FSDP)



- モデルパラメタや勾配を分配させつつ、あたかもデータ並列を行っているかのように通信を組み込む手法
- テンソル、パイプライン並列が実行環境(GPUメモリ、ネットワークトポジ、通信帯域)に対してモデル分割のチューニングが必須依存であることに対し、チューニングの負担がほぼない

DeepSpeed + ZeRO

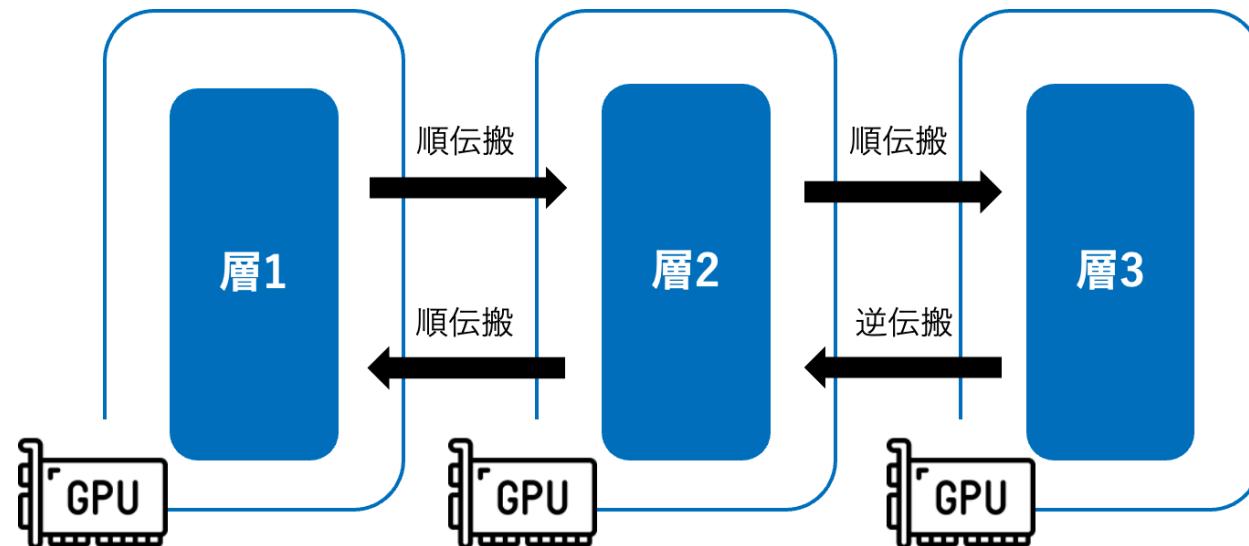


パイプライン並列



モデルを層単位で複数個に分割しプロセスに分配、中間出力をP2P通信で受け渡す

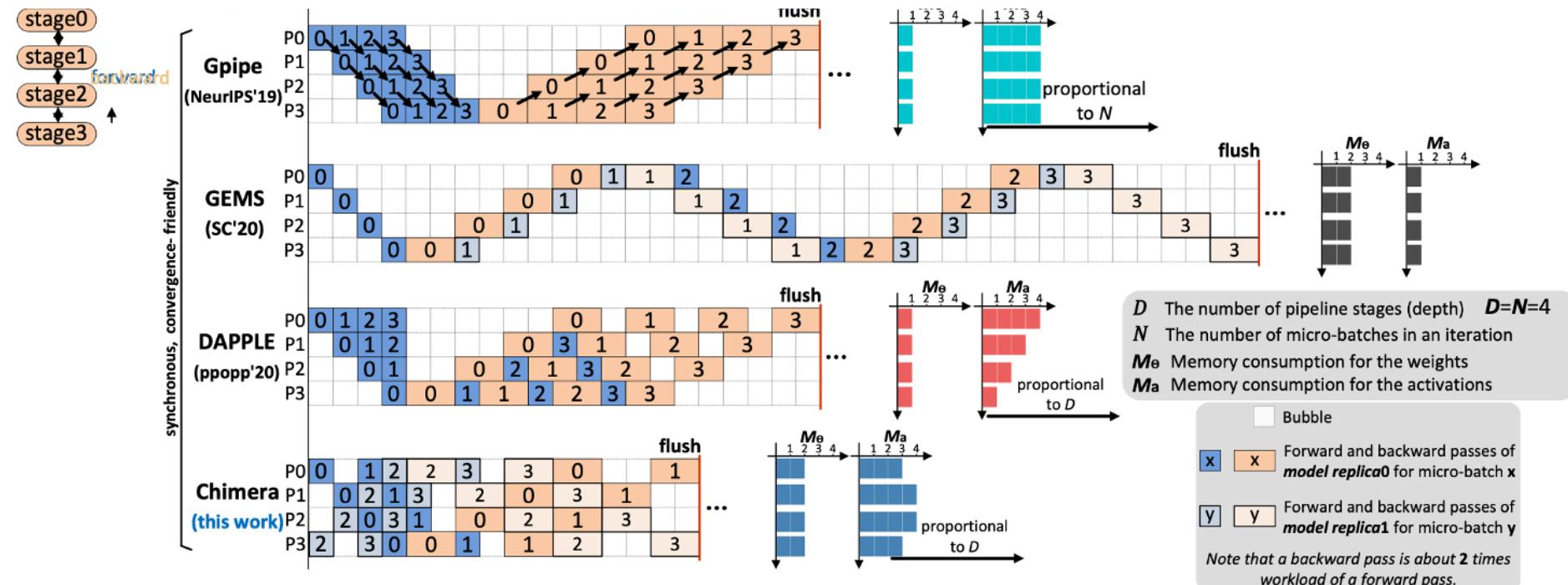
- GPU使用率を少しでも高めるために、入力データを細かく分割し、パイプライン化を行い、順伝播と逆伝播を行う
- プロセスの計算負荷に偏りがある場合、大きくアクセラレータ使用率が低下
- 後述する**パイプラインバブル**の問題により並列度に実質上限 (≤ 64 並列)



パイプライン並列 > パイプラインバブル



- 順伝播と逆伝播をパイプラインで実行する際に生じるプロセスの遊休時間
 - バッチサイズを細かくする、順伝播と逆伝播のスケジュールを調整するなどでバブルを削減可能



入力データが大きい(高精度画像など)場合は特にデータを一度に多く流すことができないためバブルが増加

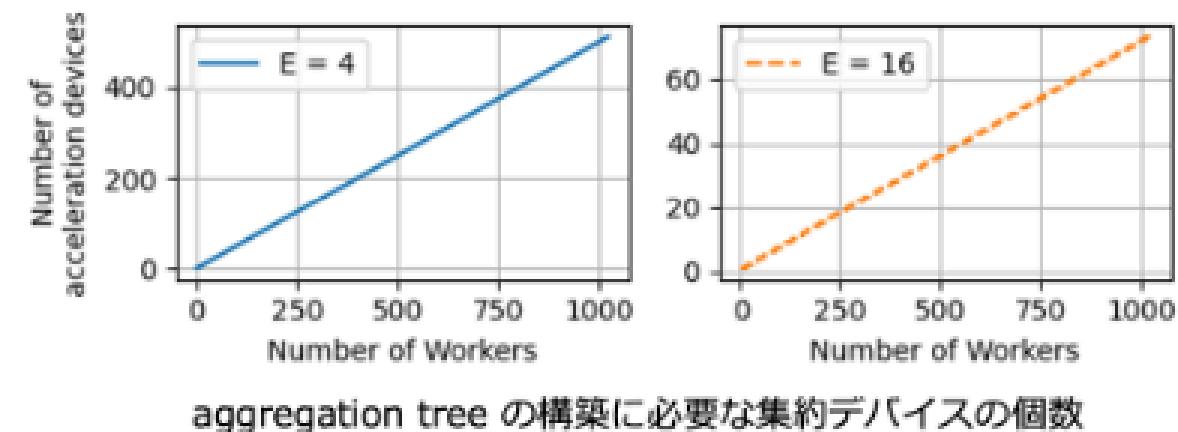
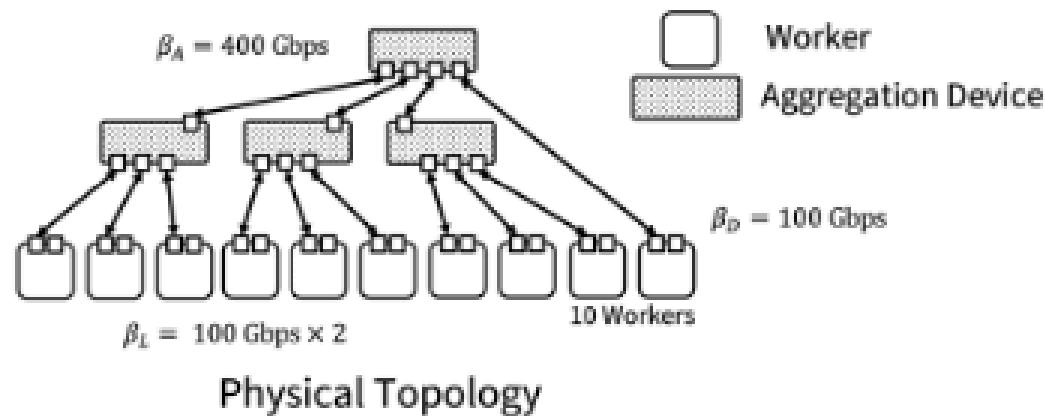
[✉]: [Li and Hoefer, 2021]: Li, Sheng, and Torsten Hoefer. "Chimera: efficiently training large-scale neural networks with bidirectional pipelines." Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. (2021). 56

3: 光回路スイッチネットワーク+INA



- 間接結合網(電気スイッチNW): 1つの集約デバイスがあれば高速化可能
 - 集約デバイスの計算帯域を超えないようにworkerがaggregation treeのデータレートを調節、残りをRingに流す
- 直接結合網(光回路スイッチNW): まとまった個数の集約デバイスが高速化のために必要
 - 節の分岐数はinterface数によって制限されるためtreeの構築にまとまった個数のデバイスが必要
 $\approx (N-1)/(E-2)$ (N: worker数, E: 集約デバイスinterface数) - N = 128 worker, E = 4の場合, 63個の集約デバイスが少なくとも必要
 - 高速化に必要なデバイス数をインフラが提供できない / マルチジョブへの公平なリソース割り当てができない

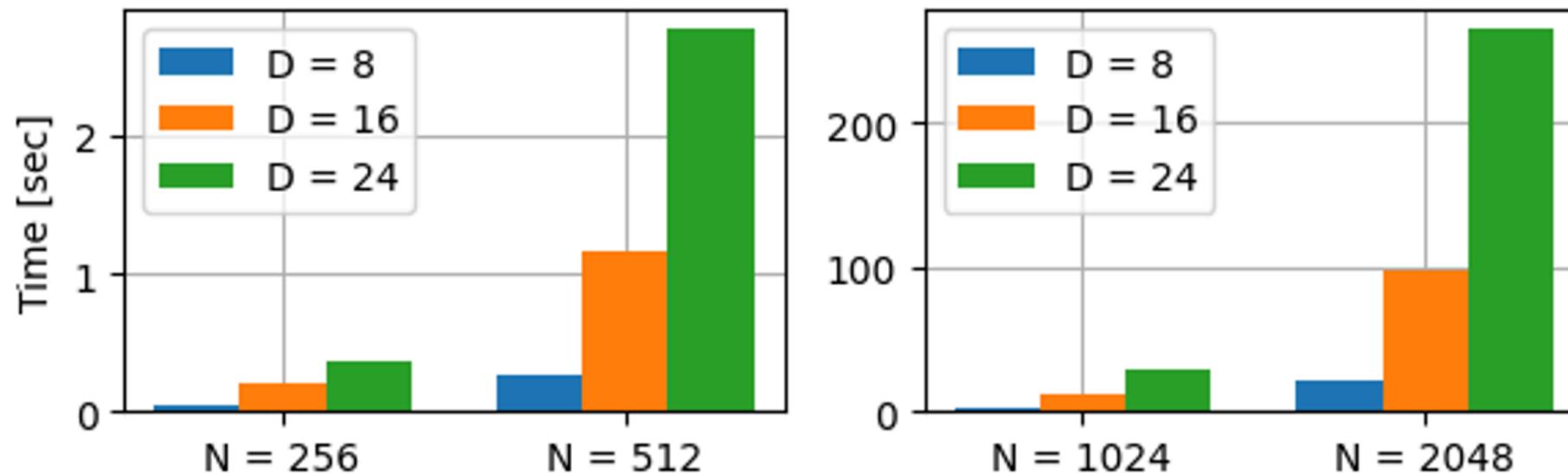
Direct connection network



aggregation tree の構築に必要な集約デバイスの個数

3: Partial Allreduce

- 高いスケーラビリティ: 実行時間 $\mathcal{O}(N^2 DM/E)$
 - N worker数, M 集約デバイス数, D workerのinterface数, E 集約デバイスのinterface数
- 1024 worker 規模で1分未満、2048 worker 規模で5分未満での探索可能
- 下図は集約デバイスのinterface数が8の場合の計算時間



参考文献



- [Vaswani+, 2017] Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).
- [Kaplan+, 2020] Kaplan, Jared, et al. "Scaling laws for neural language models." *arXiv preprint arXiv:2001.08361* (2020).
- [Hoffmann+, 2022] Hoffmann, Jordan, et al. "Training compute-optimal large language models." *arXiv preprint arXiv:2203.15556* (2022).
- [Gholami+, 2024] A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney and K. Keutzer, "AI and Memory Wall," in IEEE Micro, doi: 10.1109/MM.2024.3373763.
- [Wang+, 2023a] Wang, Weiyang, et al. "How to Build Low-cost Networks for Large Language Models (without Sacrificing Performance)?" *arXiv preprint arXiv:2307.12169* (2023).
- [Wang+, 2023b] Wang, Weiyang, et al. "TopoOpt: Co-optimizing Network Topology and Parallelization Strategy for Distributed Training Jobs." *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 2023.
- [Zhang+, 2024] Zhang, Duzhen, et al. "Mm-llms: Recent advances in multimodal large language models." *arXiv preprint arXiv:2401.13601* (2024).
- [Wang+, 2020] Wang, Guanhua, et al. "Blink: Fast and generic collectives for distributed ml." *Proceedings of Machine Learning and Systems* 2 (2020): 172-186.
- [Cai+, 2021] Cai, Zixian, et al. "Synthesizing optimal collective algorithms." *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 2021.
- [Shah+, 2023] Shah, Aashaka, et al. "{TACCL}: Guiding Collective Algorithm Synthesis using Communication Sketches." *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 2023.
- [Li and Hoefler, 2021]: Li, Sheng, and Torsten Hoefler. "Chimera: efficiently training large-scale neural networks with bidirectional pipelines." *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. (2021).
- "NTT版大規模言語モデル「tsuzumi」 | NTT R&D Website." NTT R&D Website, 25 Apr. 2024, www.rd.ntt/research/LLM_tsuzumi.html.