スケーラブルニューラル機械翻訳

小野 淳也　　内山 将夫　　隅田 英一郎
国立研究開発法人　情報通信研究機構
{junya.ono , mutiyama , eiichiro.sumita}@nict.go.jp

1　はじめに

近年，ニューラル機械翻訳の研究が精力的に行われ，翻訳精度が向上している．訓練速度もまたハードウェアの進化により向上しているが，ネットワークモデルや訓練データも増加傾向にあり，訓練には多くの時間が必要である．特に本研究の 3.6 億対訳文のような大規模データの訓練においては，1 ノード内のマルチ GPUs による高速化だけではなく，複数計算機を使用したマルチノードによる訓練時間の短縮も必要である．本研究は Sequence-to-Sequence モデル内の Encoder-Decoder 部分に Model parallel を適応し，Attention-Softmax 部分に Data parallel を適応する Hybrid parallel を提案する．1 ノード 4GPUs を利用した Hybrid parallel の場合，Baseline(1GPU)に対し 5.07 倍，Data parallel に対し 3.78 倍の高速化を実現した．本研究の訓練(4 日間)において BLEU は Baseline(1GPU)から+3.83 に向上し，Data parallel から+1.39 向上した．また，32 ノード(128GPUs)の訓練においては，1 ノードに対し 27.0 倍(効率: 0.844)の高速化を実現し，1 epoch (3.6 億対訳文) 13.5 時間に短縮した．学習率の減衰効果により BLEU は Baseline(1GPU)から+6.05 向上，1 ノード(4GPUs)から+2.22 向上した．

2　関連研究

翻訳精度を向上させたニューラル機械翻訳モデルの一つとして，Sequence-to-Sequence モデル [4] (以下，Seq2Seq)が挙げられる．Seq2Seq は Encoder-Decoder の入力が時系列であるため，並列化の効果が十分に得られない傾向にある．最近では Transformer モデル [5] により更に翻訳精度が向上し主流ではあるが，本研究では Seq2Seq を対象とする．BPE や素性 [3, 6] を使用することで語彙数を削減する手法も有力である．本研究は単語と素性を別々に入力し，二つの Embedding 層の結合（concat）を LSTM への入力とする．出力は単語と素性で別々の Softmax 層にて推定する．また，BLEU の評価は単語と素性の出力に対し前処理(BPE と素性)の復元後に行う．

並列化手法として，大きく二つの方法があり，一つ目はバッチ内を分割，またはバッチごとに別々の GPU，別々の計算機で処理する Data parallel [7]である．二つ目はモデルを分割する Model parallel [8]である．多くの深層学習フレームワークや翻訳ツールは，Data parallel による訓練の機能を備えておりモデルに依存しないため，広く利用されている．一方，Model parallel はモデルに依存し分割方法も様々であり，モデルに応じた GPU 割り振りと負荷調整が必要となるため，一般化は難しい．

高速化手法には GPU 間の同期コストの影響を小さくするため，複数バッチ(forward-backward)後に同期する方法や backward と同期処理を重ねる方法がある[1]．本研究では，1 ノードのマルチ GPUs の訓練おいては 1 バッチごとに同期し，マルチノードの訓練においては複数バッチ後の同期による訓練速度と翻訳精度の変化を考察する．マルチノードの同期方法は ChainerMN, CNTK が使用する MPI 方式と，MXNet, TensorFlow が使用する Parameter Server 方式が広く使われ，本研究では MPI 方式を使用する．

3　並列化手法

3.1　Data parallel の特徴

Data parallel はパラメータを更新するために各 GPU で処理した勾配を同期する必要がある．パラメータ数が多くなるほど，同期するコスト(時間)も多くなる．そのため，訓練速度の高速化が十分に得られない可能性がある．同期するコストと，同期までに要するコストとの比が重要である．例えば，同期以外の 1 バッチに要する時間が短い場合，同期コストが支配的になり，スケールの効果が小さくなる．

3.2　Model parallel の特徴

モデル内を GPU ごとに分割し，GPU 間でパラメータを共有せずに更新する Model parallel の場合，GPU 間は入出力の転送のみとなる．GPU 間を同期する必要がなくなるため，同期コストを減らすことが可能である．ただし，モデルの分割方法は様々であり，各 GPU のメモリ使用量も変則となり，特定の GPU に負荷が集中する場合があり調整が必要となる．また，計算機の GPU 数や GPU 間の高速転送(nvlink)を考慮し，ロードバランスを調整する必要があり，調整次第で高速化の効果は小さくなる．

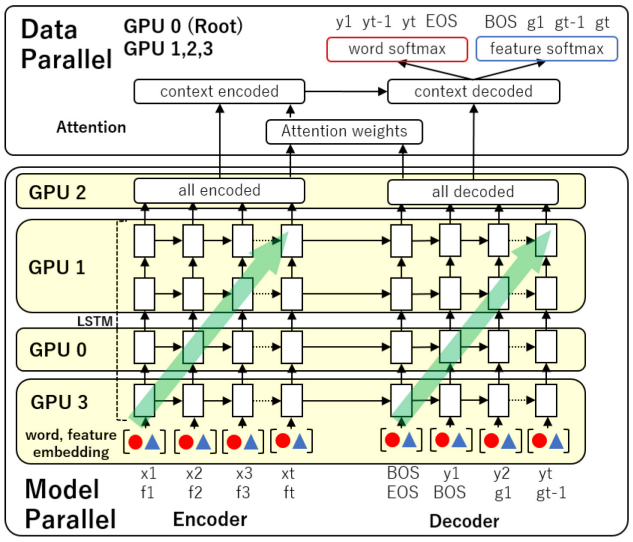

図 1: Hybrid parallel－4GPUs, LSTM 4 層の例

3.3 Hybrid parallel の提案

Data parallel と Model parallel の利点を組みわせた Hybrid parallel の提案手法を示す. 図 1 のように Attention-Softmax 層と Encoder-Decoder 部分で並列化の手法を変える．前者は Forward-Backward 処理が重たく，パラメータ数が 6 個と少ないため，Data parallel による並列化が有効である．一方，後者の Encoder-Decoder は LSTM 4 セット(4 層)の場合，パラメータ数は Embedding 層 4 個，LSTM 層 32 個，全部で 36 個と多く，また入力の系列ごとに処理する必要があり，Forward-Backward 処理よりも GPU 間の同期コストの影響が大きくなる．この場合は Model parallel により層ごとに別々の GPU を割り振ることで同期コストの影響を小さくし, 図 1 のような系列と層方向に対し斜め方向の並列化が可能である．Data parallel におけるパラメータの更新を GPU 0 とする場合，GPU 0 のメモリ使用量が大きくなる．そのため，Model parallel では GPU 0 の使用量を削減するため，図 1 のように 1 層の LSTM のみにする．本モデルは単語と素性の Embedding 層を結合(concat)し LSTM の入力とする．出力は単語と素性の Softmax 層で別々に推定し系列単位で復元する. また, t 系列の Attention の出力を t+1 系列の入力に含める Input- feeding [4]は使用しないため，Attention-Softmax 層の処理は系列間に依存関係がなく独立で処理が可能である．よって，Decoder 側の最終層 LSTM の全出力(all decoded)を(batch-size, seq-length, hidden-size)のテンソルとし，系列ごとに処理せず，一回の処理によりオーバヘッドを削減する．

本手法を一般化すると，二つ以上のサブモデルに分割し，処理コストが小さくなる並列化手法を選択することであり，Transformer 等の別モデルや，Seq2Seq が利用される要約や対話等の別タスクにおいても本手法の適応が期待できる．Hybrid parallel の効果については第 4 章で示す.

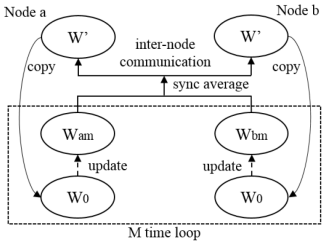

図 2: M バッチ後のパラメータ同期処理(平均)

3.4 マルチノード訓練の問題点

マルチノード訓練においてもパラメータを同期するコストが問題となる．各ノードの backward 処理を待たずにパラメータを更新する非同期処理の場合，同期待ちを削減可能であるが，収束が難しいことが報告されている [9].

本研究では同期処理とし，同期コストを小さくする方法を 2 ノードの例で図 2 に示す. 図 2 は訓練開始時に全ノード間でパラメータを同期 W_0 し，各ノードで M 回(M バッチ)パラメータを更新した後に全ノード間でパラメータを平均化して同期 W'する．同期タイミングを M バッチ後に調整することで，同期までの処理時間が多くなり同期コストの影響を小さくする. マルチノードおけるパラメータの平均化は，各ノードのパラメータが近い方向に向いていることで収束に向かう. そのため, M 値が大きくなるほど，マルチノードのスケール効率は上がるが，各ノード間でのパラメータの差異は大きくなるため，平均化により収束が難しくなる可能性がある. 特に訓練初期のパラメータは大きく変わるため，その傾向が顕著になる．M 値によるスケール効率，収束速度の違いを第 4 章で考察する.

4 実験

4.1 実験環境

実験は NICT(情報通信研究機構)が所有する特許データを使用する．訓練データは 3.6 億対訳文，開発データは訓練データと一致しない 7782 対訳文を使用する．言語方向は日英とし，BPE と素性により原言語(日)の語彙数は単語側：85510，目的言語(英)は 93459，素性はどちらも 9 とする(BOS, EOS 含む)．ハードウェアは，1 ノードの GPU：NVIDIA Tesla V100 x 4 枚 (SXM2 版)，インターコネクト：InfiniBand EDR (12.5GB/s) を使用する.

提案手法含めた Seq2Seq モデルの実装は MXNet(v1.3.0)を使用し，1 ノード内の GPU 間は nvlink による P2P Direct 転送を使用する. マルチノードの数は 32 とし, 通信は Intel MPI(v2018 update2)で実装する．Embedding：単語 1000 次元，素性 4 次元，LSTM：4 層，1000 次元とする．本実装を HybridNMT と定義する．HybridNMT の訓練速度と翻訳精度の比較のため，単語と素性の入力が可能，かつネットワークモデルが同じである Lua 版の OpenNMT-lua(v0.9.2) [3]を使用する．GPU 間転送は nccl(v1.3.5)を使用する.

	SRC tokens / sec	Scaling fator	Expected days / 1 epoch	BLEU Training: 4 days
OpenNMT-lua				
- Baseline (1GPU)	2280	—	72.5	36.32
- Data parallel (4GPUs)	2921	1.28	56.6	38.25 (+1.93)
HybridNMT				
- Baseline (1GPU)	2147	—	77.0	35.84
- Data parallel (4GPUs)	2886	1.34	57.6	38.28 (+2.44)
- Hybrid parallel (4GPUs)	**10910**	**5.08**	**15.2**	**39.67 (+3.83)**

表 1: Hybrid parallel の効果

4.2 マルチ GPUs による Hybrid parallel の効果

1 ノード 4GPUs(バッチサイズ 4 倍)による提案手法 Hybrid parallel の訓練速度と翻訳精度の結果および，比較のため Baseline (1GPU)と既存の Data parallel(4GPUs)の結果を表 1 に示す．訓練時間は計算機の都合上 4 日間とする．BLEU は開発データに対する結果である．訓練速度の結果から 1 epoch(3.6 億対訳文)の見込み日数を算出する．比較対象の OpenNMT-lua は，Data parallel により 1 epoch が 72.5 日から 56.6 日に 1.28 倍の高速化となり，BLEU は +1.93 向上した．一方，HybridNMT は Data parallel により 77.0 日から 57.6 日に 1.34 倍の高速化となり，BLEU は +2.44 向上した．既存の Data parallel では十分なスケールが得られない原因として，GPU 間の同期コストが影響していると考えられる．同期コストを抑える Hybrid parallel においては，15.2 日に短縮し 5.08 倍の高速化となり，BLEU は+3.83 向上した．Data parallel においては OpenNMT-lua と同程度の性能であるが，Hybrid parallel においては OpenNMT-lua よりも訓練速度と翻訳精度ともに優位であると言える．パラメータの大部分を占める Encoder-Decoder のパラメータを同期する必要がないことが高速化の要因であり，本結果から提案手法が並列化に有効であることが言える．しかし，提案手法による 1 ノード内の高速化だけでは 3.6 億対訳文の訓練に多くの時間(1 epoch 15.2 日)を要するため，マルチノードによる更なる高速化が必要である．

4.3 マルチノードの同期タイミングの比較

第 3.4 節で記載したとおり，訓練開始時は各ノードの初期パラメータの向きは大きく異なるため収束が遅くなる．そのため，事前に 1 ノード(4GPUs)による訓練時間 1 日，2432 万対訳文を訓練したパラメータを各ノードの初期パラメータとする．第 4.2 節での訓練時間 4 日間と合わせるため，マルチノードの訓練時間は 3 日間とする．3.4 節の各ノードにおけるパラメータの同期タイミング M の値を変えてスケールアウトと精度を比較する．例えば，M=1 の場合，毎バッチごとにパラメータを同期するため，32 ノードによるスケールは 2.58 倍(効率 : 0.081)となり，同期コストが支配的になりノード数を増やしてもスケールは不可能である．本実験では，開発データにおける収束速度を

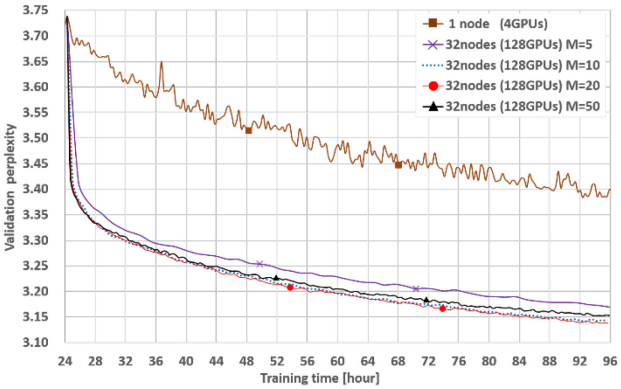

図 3: マルチノードの同期タイミング M 値の比較

M=5,10,20,50 ごとに比較した結果を図 3 に示す．スペースの関係上，スケールの効果は本文で示す．M=5 なら 9.58 倍(効率: 0.299)，M=10 なら 14.30 倍(効率: 0.447)，M=20 なら 19.1 倍(効率: 0.596)，M=50 なら 24.1 倍(効率 : 0.752)となる．M 値が大きいほど，スケール値は大きくなる．また，validation perplexity の差は大きくはないが，M=10,20 に比べて M=5,50 が比較的大きいことから，M の値が大きいほど同時間で訓練するバッチ数が増えるためスケールするが，各ノードのパラメータの差が大きくなり，平均化により収束が遅くなると考えられる．スケールと収束速度のバランスから M 値を調整することが重要である．

4.4 最適化手法のステート

これまでの実験の最適化手法は Adam を使用した．Adam のようなステートを持つ場合，全ノードで平均化したパラメータと，各ノードのステートで不整合が発生する．そのため，一般的にマルチノード訓練ではステートなしの SGD が使われる [1, 2, 9]．ただし，1 ノードの訓練において SGD はステートなしのため収束が遅い傾向にある．本実験では，ステートを同期しない Adam と，ステートなしの SGD を比較する．図 4 はマルチノードにおいても SGD よりも Adam の収束が速いことを示している．最小値の比較ではないため Adam が完全に優位とは言えないが，最小値までの訓練が厳しい場合においては Adam が優位と言える．

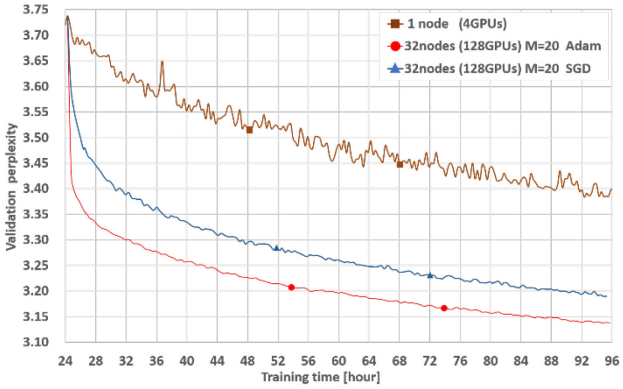

図 4: マルチノードにおける Adam と SGD 比較－M=20

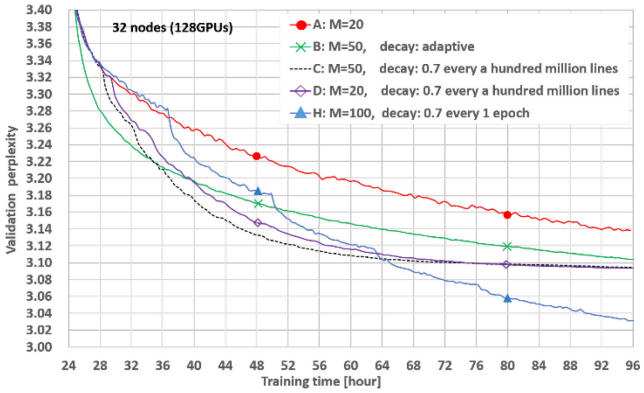

図 5: M 値と学習率の減衰方法の比較（その 1）

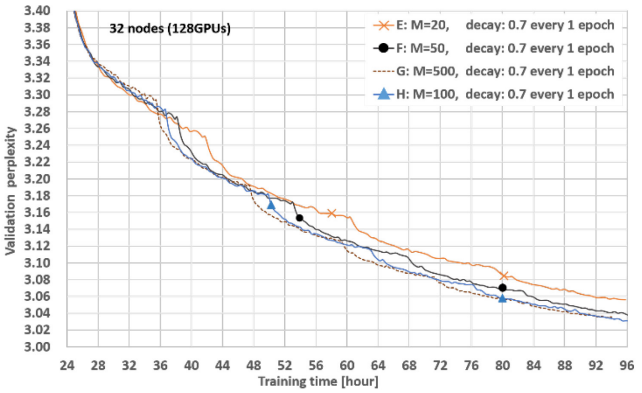

図 6: M 値と学習率の減衰方法の比較（その 2）

Case	Node / GPU	M batch sync	Learning rage decay	Perplexity	BLEU	SRC tokens /sec	Scaling fator (efficiency)
—	1 / 4	—	—	3.399	39.67	10910	—
A	32 / 128	20	—	**3.138**	**41.09**	208224	19.1 (0.596)
B		50	adaptive	3.104	41.34	262464	24.1 (0.752)
C			0.7 every a hundred million lines	3.095	41.36		
D		20		3.094	41.39		
E			0.7 every 1 epoch	3.056	41.74		
F		50		3.038	41.85		
G		500		3.032	41.88	328928	30.1 (0.942)
H		100		**3.031**	**41.89**	294560	27.0 (0.844)

表 2: M 値と学習率の減衰方法の比較（まとめ）

4.5 学習率の減衰効果

学習率の減衰はモデルやデータに依存するため，減衰方法は様々である．例えば，OpenNMT-lua のデフォルトでは epoch 単位で Perplexity が上がる場合，または 10 epochs 以降に強制的に 0.7 倍で減衰する．この設定での本実験では 1 epoch(3.6 億対訳文)の間隔が長く，減衰は発生しない．M 値と減衰を調整した結果を表 2，図 5，図 6 に示す．A は減衰なし，B は Transformer モデルのバッチ単位で減衰させる方法[5]とし，学習率の初期値 0.001 と一致させるため d=1000, warmup_steps=1000 とした．この設定では学習率の減衰が速く早い段階で収束速度が低下した．また，C, D では 1 億文ごとに固定倍率 0.7 で減衰し M 値を 50,20 に変えた場合も同様に減衰間隔が短く，早い段階で収束した．一方，E,F,G,H では epoch 単位で減衰させ M 値を 20,50,500,100 に変えた場合，M 値が大きくなるほど訓練速度も速くなるため，減衰の間隔が小さくなり収束速度も速くなる傾向である．しかし，M=500 よりも M=100 の Perplexity が少し低いことから，M 値が大き過ぎても収束に有利とは限らない．

本実験の訓練において 32 ノード(128GPUs)で M=100 により 27.0 倍(効率：0.844)の高速化，1 epoch: 13.5 時間の短縮を実現し，epoch 単位で 0.7 倍の減衰により BLEU が 41.89 となり Baseline(1GPU)から+6.05 向上，1 ノード(4GPUs)から+2.22 向上した．データに依存するため減衰方法の一般化は難しいが，学習率の調整は重要である．

5 おわりに

Seq2Seq モデルの Encoder-Decoder に Model parallel を適応し，Attention-Softmax に Data parallel を適応する Hybrid parallel の手法を提案した．1 ノード 4GPUs で 5.07 倍の高速化を実現した．本手法は「二つ以上のサブモデルに分割し有利な手法の選択」として一般化できる．別モデルや，要約や対話の別タスクにも適応が期待でき，有効性を示す予定である．また，32 ノードによるスケール率と精度の変化や，本訓練における Adam の優位性を示した．複数モデルの平均化は有効な手法であり，訓練中のパラメータの平均化も同様の効果として精度向上の要因の一つである．

参考文献

[1] Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018b. Scaling Neural Machine Translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*.

[2] Amith R Mamidala. Georgios Kollias, Chris Ward, and Fausto Artico. 2018. MXNET-MPI: Embedding MPI parallelism in Parameter Server Task Model for scaling Deep Learning. *arXiv:1801.03855*.

[3] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. 2017. OpenNMT: Open-Source Toolkit for Neural Machine Translation. *arXiv:1701.02810.*

[4] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. In *Proc. of EMNLP*.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *Proc. of NIPS*.

[6] Rico Sennrich and Barry Haddow. 2016. Linguistic Input Features Improve Neural Machine Translation. *arXiv:1606.02892.*

[7] Adam Coates, Brody Huval, Tao Wang, David J. Wu, Bryan Catanzaro, and Andrew Y. Ng. 2013. Deep learning with COTS HPC systems. In *Proc. of ICML*.

[8] Jeffrey Dean, Gregory S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. 2012. Large Scale Distributed Deep Networks. In *Proc. of NIPS*.

[9] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. 2016. Revisiting Distributed Synchronous SGD. a*rXiv:1604.00981*.