Vol.2018-HPC-165 No.3
2018/7/30

大規模メニーコアシステムOakforest-PACSにおけるディープラーニングの性能評価

田村 紘平[1]　塙 敏博[2,1]

概要：近年，ディープラーニングの応用は様々な分野において注目を浴びている．しかし，ディープラーニングの学習において，構成するネットワークが複雑になり，学習に用いるデータセットも増大してきているため，多数の GPU を搭載した大規模 GPU クラスタを用いた高速な学習が提案されている．一方，CPU 中に多数のコアを搭載したメニーコアプロセッサでも，GPU と同様にディープラーニングの基本演算に向いた演算ユニットを備えている．本研究では，ディープラーニングフレームワークである ChainerMN を用いて，メニーコア型プロセッサ Intel Xeon Phi を搭載した Oakforest-PACS における性能評価を実施した．

1. はじめに

Deep Learning(以下 DL) は Deep Neural Network(以下 DNN) と呼ばれる計算モデルを用いた機械学習であり，近年多くのコンペティションやタスクで著しい成果を上げ注目が高まっている．しかし，DNN の学習パラメータ数や演算処理が膨大であることから非常に長い時間を要することが知られている．GPU を効率よく利用して学習することが標準的なアプローチとして採用されてきたが，モデルの巨大化や複雑化に伴い，最新の GPU を使用した場合でも依然として長い時間を必要とする問題が挙げられる．

そこで高速化の手法として Message Passing Interface（MPI）等を利用して複数のノードで並列分散処理をする方法がある．[4] では各ノードに 1 つの GPU を搭載した 256 ノードの環境において，MPI を用いたデータ並列によって 217 倍の処理速度の高速化を達成されている．[5] では 1,024GPU で構成される並列コンピュータ MN-1 と，分散学習パッケージ ChainerMN を用いて ImageNet データセットを利用した ResNet-50 の学習を 15 分で完了している．

このように複数の GPU を用いた高速化が多く提案されているが，多数のコアを搭載したメニーコアプロセッサにおいてもディープラーニングの基本演算に向いた演算ユニットを備えているため高速化が期待できる．メニーコア型プロセッサである Intel Xeon Phi（開発コード名:KNL=Knights Landing）を搭載した Cori 上で気象データの分類タスクの学習を行なったところ 15.07PFLOPS を達成した報告もある [6]．そこで本稿では，GPU 搭載クラスタにおける複数ノードでの実行において高い性能を示している ChinerMN をメニーコアプロセッサに適用するため，Intel Xeon Phi プロセッサを搭載した Oakforest-PACS（OFP）（最先端共同 HPC 基盤施設）上で性能評価を行なった．

[1] 東京大学大学院工学系研究科電気系工学専攻
Department of Electrical Engineering and Information Systems, Graduate School of Engineering, The University of Tokyo
[2] 東京大学情報基盤センター
Information Technology Center, The University of Tokyo

2. 背景

2.1 ChainerMN

ChainerMN とは Preferred Networks 社によって開発された DL のフレームワーク Chainer の追加パッケージである．Chainer は Python によって実装されており，define-by-run モデルによって簡便な記述で自由度の高いネットワークを記述することができる．現状では GPU に向けた実装がメインとなっており，numpy の代わりに CUDA を native に用いる cupy を使っている．CPU に向けた実装は主にデバッグのために用意されているため性能に関する考慮はあまりされていない．

ChainerMN はこの Chainer をベースにして分散処理によって学習を高速化することができる．ChainerMN を用いた学習における 1 イテレーションのプロセスを**図 1** に示す．各ステップについて説明する．

- Forward
  Forward 処理とは，DNN において相互連結して構成されたノードの出力が他のノードの入力として次々に値が渡されていく処理を指す．このときノードの出力に掛けられる重み係数や加えられるバイアス値を学習

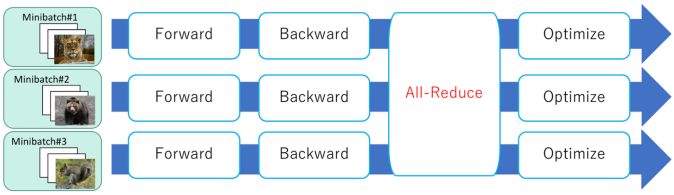

図 1　ChainerMN における学習プロセス

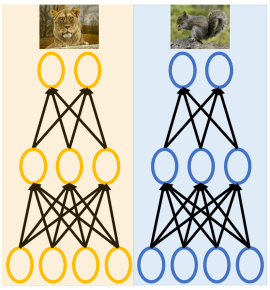

図 2　データ並列学習

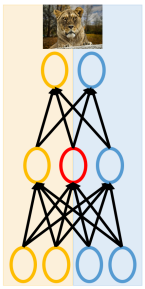

図 3　モデル並列学習

パラメータと呼ぶ．

- Backward
  Backward 処理とは，Forward 処理によって得られた出力層の値と正解データとの誤差を損失関数 E によって計算し，この誤差を最小にするパラメータを決定する処理を指す．この時に用いる各学習パラメータに関する微分値 ∇E を勾配と呼び，出力層から入力層に向かって誤差を伝播させることから Backward と呼ばれている．
- All-Reduce
  All-Reduce 処理とは，ノード間の通信処理を行うことを指す．各ワーカーが Backward で求めた勾配の平均を計算し，その結果を全ワーカーに配る．Optimize の処理が開始されるときは全ワーカーは学習開始後常に同じパラメータを持つ．
- Optimize
  Optimize 処理とは，Backward 処理によって得られた勾配値をもとに学習パラメータを更新する処理を指す．最も基本的なアルゴリズムとして確率的勾配法（SGD: Stochastic Gradient Descent）が挙げられる．

2.2　モデル並列とデータ並列

DNN 学習処理を複数の演算器に分割する方式として，モデル並列学習とデータ並列学習がある．それぞれの特徴についてまとめる．

モデル並列学習は学習処理を行う DNN を分割し，それぞれ別の演算器で処理を行う方法であり，演算器間を各層の入出力データが転送される．複数の演算器に一つの DNN を分割して処理するため，巨大な DNN を演算する手法として用いられるが，各演算器の処理量を一定にすることが難しくなる点がデメリットとして挙げられる．さらに，各処理において演算器間で転送されるデータの要素数にばらつきが生じやすく，バッチサイズにも比例して分散が増大する．そのためノード数の増加に伴い，特定のノードの演算や通信がボトルネックとなり性能が低下しやすい．

データ並列学習は各演算器が同じ DNN モデルを保持した状態で，それぞれが異なるバッチデータを用いて学習する方法である．学習処理は，各演算器で異なるバッチデータから算出された勾配情報 ∇E を集約し．パラメータの更新を行った上で次のバッチ処理を行う．モデル並列とは異なり，各演算器が独立して Forward 処理から Backward 処理を実行できるため，演算器は各層の処理を中断することなく実行できる．しかしデータ並列学習は，DNN の重みパラメータのサイズによって決まる ∇E を通信し集約するためパラメータサイズの大きい DNN の場合，集約処理の負荷が大きくなる特徴がある．

2.3　バッチサイズと精度の関係

DL における学習のほとんどがミニバッチ学習を行う．ミニバッチ学習とは学習データをほぼ等しいサイズのグループに分割し，各グループごとに損失を計算し，重みを更新する方法である．バッチサイズが大きいほどイテレーションの回数が減るため 1epoch の計算速度が速くなるが精度が悪くなることが知られている．

データ並列においては各プロセスのバッチサイズにプロセス数を乗じた値が全体のバッチサイズに相当する．そのためプロセス数を増やしすぎると精度が劇的に劣化してしまう．加えて，勾配の分散が小さくなることにより，性質の悪い局所解に進みやすくなり，結果として得られるモデルの汎化性能が悪くなってしまうという現象も知られている．そこでバッチサイズに比例させて学習率を大きくする (Linear Scaling) ことで 8GPU の時とほぼ同様の精度を保ったまま 256GPU(バッチサイズ: 8192) で ImageNet の学習を 1 時間で完了した報告もある [9]．この時の学習曲線を**図 4** に示す．

2.4　Oakforest-PACS

本稿では計算機環境として Oakforest-PACS（OFP）（最先端共同 HPC 基盤施設）を使用した．システムの構成及び計算環境の概要をそれぞれ**表 1** に示す．OFP は 8,208 台の計算ノードで構成されており，総理論演算性能（倍精度浮動小数点演算性能）は 25PFLOPS である．計算ノードは Intel Xeon Phi プロセッサ（開発コード名:KNL=Knights Landing）1 ソケットで構成されており，それまでの Xeon Phi x100 シリーズ (Knights Corner) がアクセラレータと

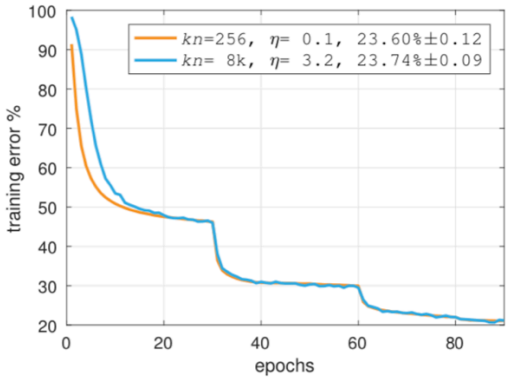

図 4　Linear Scaling による巨大ミニバッチの学習曲線 [9]

表 1　計算環境の概要

CPU	Intel Xeon Phi 7250 (Knights Landing)
動作周波数 (GHz)	1.40
コア数 (最大有効スレッド数)	68(272)
理論演算性能 (GFLOPS)	3,046.4
主記憶容量 (GB)	MCDRAM: 16 DDR4: 96
メモリバンド幅性能 (GB/s)	MCDRAM: 490 DDR4: 84.5
インタコネクト	Intel Omnipath Architecture (100 Gbps) Full-bisection BW Fat-tree

して Xeon プロセッサなどと PCIe 接続される必要があったのに対し，それ自身がブート可能な，メニーコアシステムである．KNL の CPU 上には DDR4 と，MCDRAM と呼ばれる 3 次元積層メモリが搭載されている．KNL は，この MCDRAM を利用するためのモードとして以下のメモリモード，クラスタリングモードを備えている．

メモリモード

- Flat: MCDRAM と DDR4 が独立したアドレス
- Cache: MCDRAM は DDR4 メモリのキャッシュとして動作
- Hybrid: MCDRAM の容量を分割して，Flat モードと Cache モードの両方を利用可能化

OFP では Flat と Cache それぞれに専用のキューを設けて使い分けられる．

クラスタリングモード

- All-to-all:アドレス情報が全体に分散
- Quadrant, Hemisphere:内部でアドレス情報が 4(または 2) に分割
- SNC-4, SNC-2:NUMA ドメインが明示的に 4(or 2) に分割

OFP では Quadrant モードで動作する．

3. 評価

本研究では Oakforest-PACS をディープラーニングの大規模計算環境として提供することを目的としている．そこで ChainerMN によって性能評価を行なった．

本稿では，複数ノードでの性能評価に先立って，1 ノードにおける複数スレッド実行，および複数スレッド+MPI ハイブリッド実行についての詳細な評価を述べる．これらの分析を元に今後複数ノードでの性能評価を実施する．

Platform: Oakforest-PACS
Memory: Flat
Dataset: ImageNet
Model: Resnet50

図 5　評価環境

表 2　ソフトウェア

Software	Version
Intel Python	3.5.2
Intel MPI	2018.1.163
MPI4py	2.0.0
Chainer	4.2.0
ChainerMN	1.3.0
iDeep4py	1.0.4

表 3　1 イテレーションあたりの実行時間

KNL	Broadwell
88.42[sec]	56.91[sec]

3.1 評価環境

実験環境を図 5，使用したソフトウェアを表 2 に示す．Python は Intel 社が提供する Intel Distributed for Python を使用した．これは従来の Python とは異なり，演算のコアになる Numpy や Scipy において Intel Xeon Phi 向けに最適化された MKL を使用するため，性能の向上が期待できる．また，Chainer のバージョンは v4.2.0 を用いた．2018 年 4 月にリリースされた Chainer v4 から Intel Deep Learning Package(iDeep) に対応し，Intel CPU での学習及び推論の高速化が実現されると共に，OpenMP によるスレッド並列が可能となった．

メモリモードは Flat に設定し，I_MPI_HBW_POLICY 環境変数を hbw_preferred として各プロセスに MCDRAM を割り当てるようにし，収まりきらない場合は DDR4 も使用するように設定した．モデルは ResNet-50 を使用し，データセットは ImageNet データセットを 10 クラスまで削減したものを用いた．

3.2 iDeep を無効にした場合

3.2.1 実行時間と学習曲線

iDeep による最適化を行わずに Intel Xeon Phi 7250(以下 KNL) 上で 1 プロセス実行した場合の結果を図 6, 図 7，および表 3 に示す．比較のために Intel Xeon E5-2695 v4(以下 Broadwell) を使用した場合の結果も併記する．

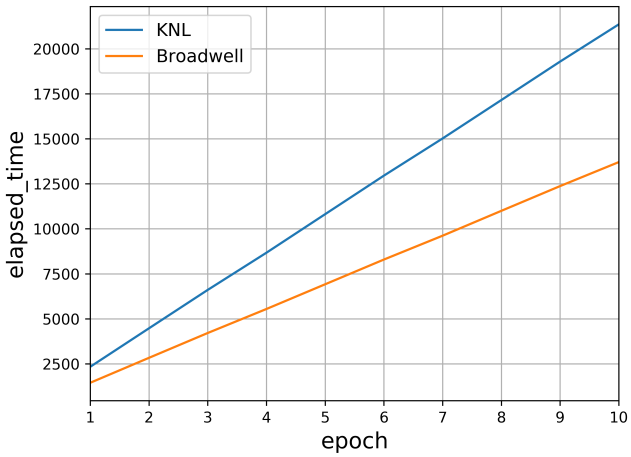

図 6　実行時間

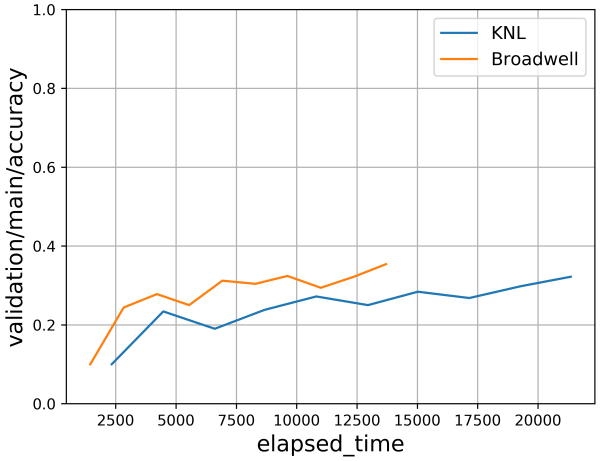

図 7　学習曲線

結果が示すように，最適化をしない場合 KNL は 1 イテレーションあたりの実行時間が非常に遅く，Broadwell と比べても約 1.6 倍の差があることがわかる．これはコアあたりのクロック数による差が原因であると考えられる．KNL の動作周波数は 1.4GHZ であるのに対して Broadwell は 2.1GHz であるため，並列化をしない場合は Broadwell の方が速い結果となった．しかし，KNL は Broadwell に比べて高いスレッド数で並列化することができる点や高バンド幅メモリ MCDRAM を利用することができる点において優位であることから最適化やチューニングを図ることで十分に性能を引き上げられることが期待できる．

3.2.2　メモリの消費量

1 プロセスにおける各カーネルでの使用メモリを図 **8** に示す．図が示すように 1 プロセスでのメモリの消費量は約 6GB であり、その多くが畳み込みで消費されていることがわかる．KNL1 基が使用できるメモリは DDR4(16GB)×6(96GB)+MCDRAM(16GB)=112GB　であるため，現状ではノード内の KNL 全ての 68 コアを全て使用した場合は使用メモリが上限を超えてしまう．現状ではノード内で 8 プロセスまでは実行できるが，それを超えると上限を超えてしまいエラーとなる．

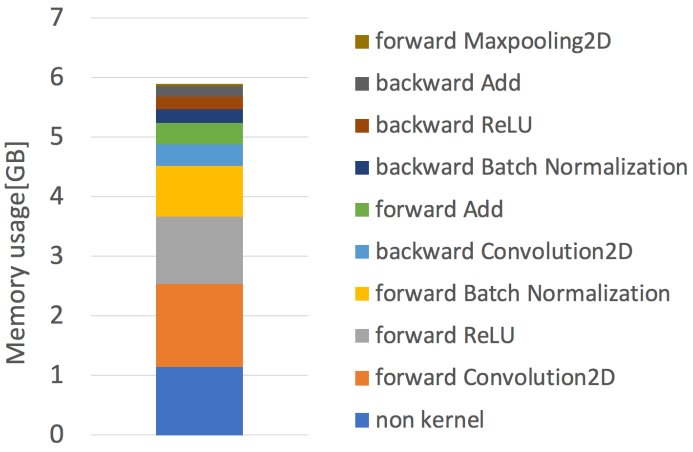

図 8　各カーネルにおけるメモリの消費量

表 4　1 イテレーションあたりの実行時間

KNL	Pascal
3.70[sec]	0.68[sec]

3.3　iDeep による最適化

iDeep を用いて OpenMP によるマルチスレッド並列と，OpenMP+MPI によるハイブリッド並列を行なった．iDeep は 3.1 節で述べたように Chainerv4 から追加されたバックエンドであり，これにより Intel CPU でのマルチスレッド並列が可能になった．

3.3.1　マルチスレッド並列

プロセス数を 1 とし，OpanMP によるマルチスレッド並列の結果を図 **9** に示す。図 9 は各フェーズごとにおける時間実行時間を示しており，スレッド数をあげるごとに各フェーズでの実行時間を短縮できており，並列化の効果が出ていることが確認できる．1 ノードにおける最大の並列数である 272 スレッドにおいては，並列化をしない場合と比べて約 26 倍の高速化を達成した．

また，最大のスレッド数である 272 スレッドでの結果と NVIDIA Tesla P100 の結果を比較する．それぞれの実行時間と学習曲線を図 **10**，図 **11** に示す．

3.3.2　ハイブリッド並列

OpenMP によるマルチスレッド並列に，MPI によるマルチプロセスを組み合わせたハイブリッド並列の結果を図 **12** に示す．メモリの制約により 8 プロセスまでの結果を示している．横軸は全体のスレッド数を表しており，括弧内の数字は 8 プロセスにおけるスレッド数を表している．8 プロセスの場合，68 の物理コアにスレッドを均等に割り当てることができないため，64 の物理コアを使用して

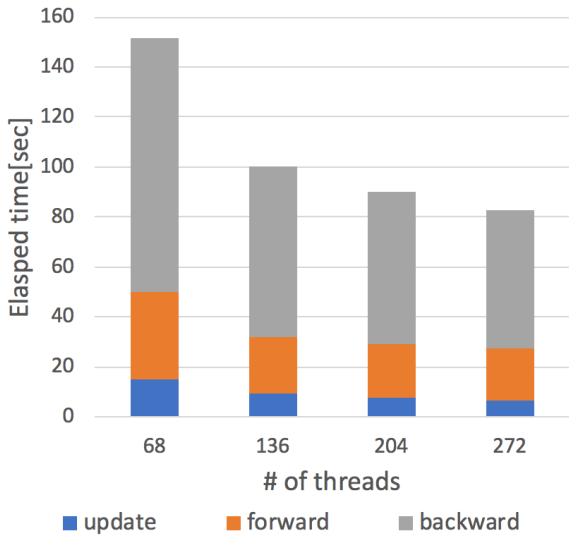

図 9　マルチスレッド並列における結果

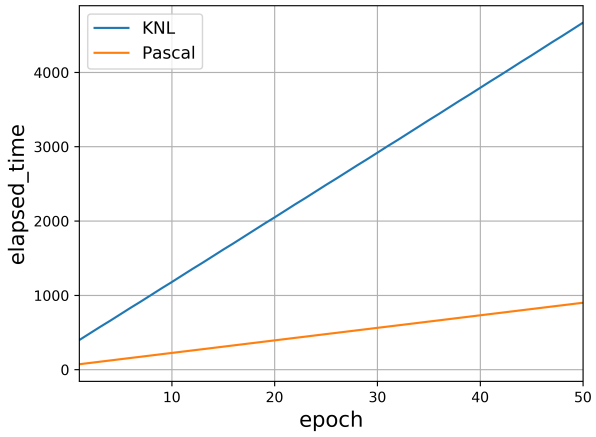

図 10　実行時間

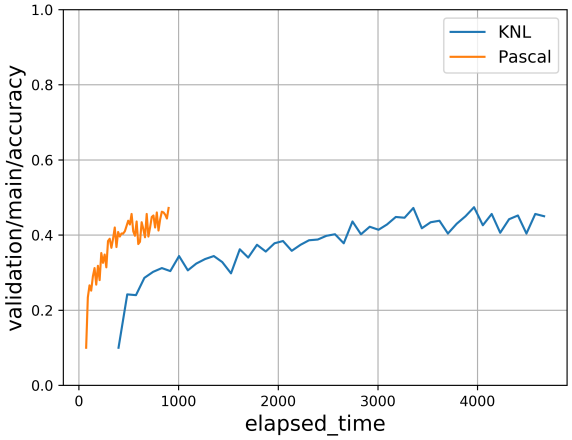

図 11　学習曲線

いるためである．結果から，プロセス数を増やすことにより実行時間が短縮できていることからハイブリッド並列に

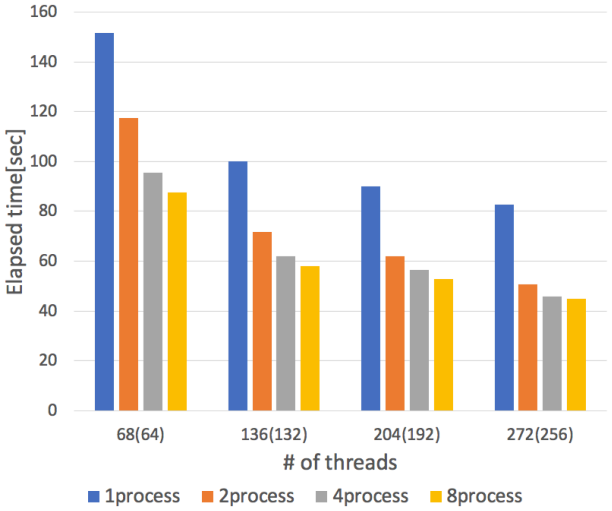

図 12　ハイブリッド並列における結果

よる効果の有効性が確認できた．しかし，2.3 節で述べたようにプロセス数を大きくするにつれて全体のミニバッチサイズも増大してしまうため，精度が劣化する可能性がある．精度に関する評価は今後の課題として検討する．

4. まとめ

本研究では Oakforest-PACS を用いて ChainerMN の性能評価を実施した．Resnet-50 で ImageNet データセットを用いて学習した場合，並列化を行なっていない KNL は非常に遅い結果となることがわかった．そこで Chainer v4 から追加された iDeep バックエンドを利用して OpenMP によるスレッド並列と OpenMPI+MPI によるハイブリッド並列を行なった．その結果，スレッド並列によって最大で約 26 倍の高速化を達成した．また，ハイブリッド並列によってスレッド数に加えてプロセス数を増やすことにより高速化が実現できることを示した．

今後の展望としては，本評価実験では単一ノードにおける評価を実施したため，マルチノードでの評価を実施したい，マルチノードで実行することで大規模なディープラーニングが可能になると同時にマルチスレッドによる通信最適化が期待できる．この時の精度に関する評価も実施したいと考えている．

謝辞　本研究について多くのご助言をいただいた Preferred Networks 社福田圭祐氏，鈴木脩司氏に深く感謝いたします．そして本研究についてご議論いただいた，東京大学情報理工学研究科 田浦健次朗教授，樋口兼一氏に感謝いたします．

参考文献

[1] O. Russakovsky, J. Deng, H. Su, et al.: “ImageNet Large Scale Visual Recognition Challenge”, International journal of Computer Vision (IJCV), 2015.

[2] K. He, X. Zhang, S. Ren, and J. Sun.: “Deep residual learning for image recognition”, CVPR, 2016.
[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei.: ”ImageNet: A Large-Scale Hierarchical Image Database”, CVPR, 2009.
[4] 山崎雅文，笠置明彦，田原司睦，中平直司: ”MPI を用いた Deep Learning 処理高速化の提案”，情報処理学会研究報告, Vol. 2016-HPC-155, 2016.
[5] T.Akiba, S.Suzuki, and K.Fukuda: ”Extremely Large Minibatch SGD: Training ResNet-50 on ImageNet in 15 Minutes”, NIPS, 2017
[6] T. Kurth, J. Zhang, N. Satish, et al.: ”Supervised and Semi-Supervised Classification for Scientific Data” SC, 2017
[7] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang: ”On Large-batch Training for Deep Learning: Generalization Gap and Sharp Minima”, arXiv preprint arXiv:1609.04836, 2016.
[8] T.Akiba, S.Suzuki, and K.Fukuda: ”ChainerMN: scalable distributed deep learning framework”, CoRR, abs/1710.11351, 2017.
[9] Goyal, P., Dollár, P., Girshick, R., et al: ”Accurate, large minibatch sgd: Training imagenet in 1 hour.”, arXiv preprint arXiv:1706.02677, 2017.
[10] A.Viebke and S.Pllana: ”The Potential of the Intel Xeon Phi for Supervised Deep Learning”, IEEE, 2015, pp. 758–765
[11] 塙敏博，星野哲也，中島研吾，大島聡史，伊田明弘：” Xeon Phi+OmniPath 環境における OpenMP, MPI 性能最適化”, 情報処理学会研究報告, Vol. 2017-HPC-158, 2013.