学習条件を考慮した大規模非同期ディープラーニングシステムの性能モデリング

大山 洋介[1,a)] 野村 哲弘[1] 佐藤 育郎[2] 西村 裕紀[3] 玉津 幸政[3] 松岡 聡[1]

1 東京工業大学

2 デンソーアイティーラボラトリ

3 株式会社デンソー

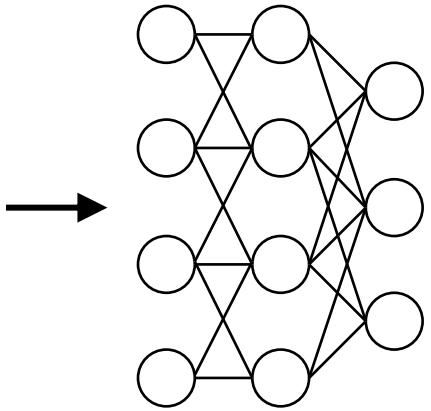
a) oyama.y.aa@m.titech.ac.jp, 発表者

研究背景
Deep Learning (DL)

- “深い”Neural Network (NN)を用いた機械学習手法
  - NNのエッジの重み(パラメータ)を学習により最適化
- 画像・音声認識などで他の手法を超える認識性能を達成
- 深いNN × 巨大な入力データセット → 大規模計算の必要性
  - TSUBAME2.5 16ノード(48 GPU)での15層CNNの学習に199時間 (8.2日)
  - NNの構造・学習パラメータをトライ・アンド・エラーで調整するために複数回の学習が必要

入力(画像)

(出典: http://image-net.org/)



Neural Network

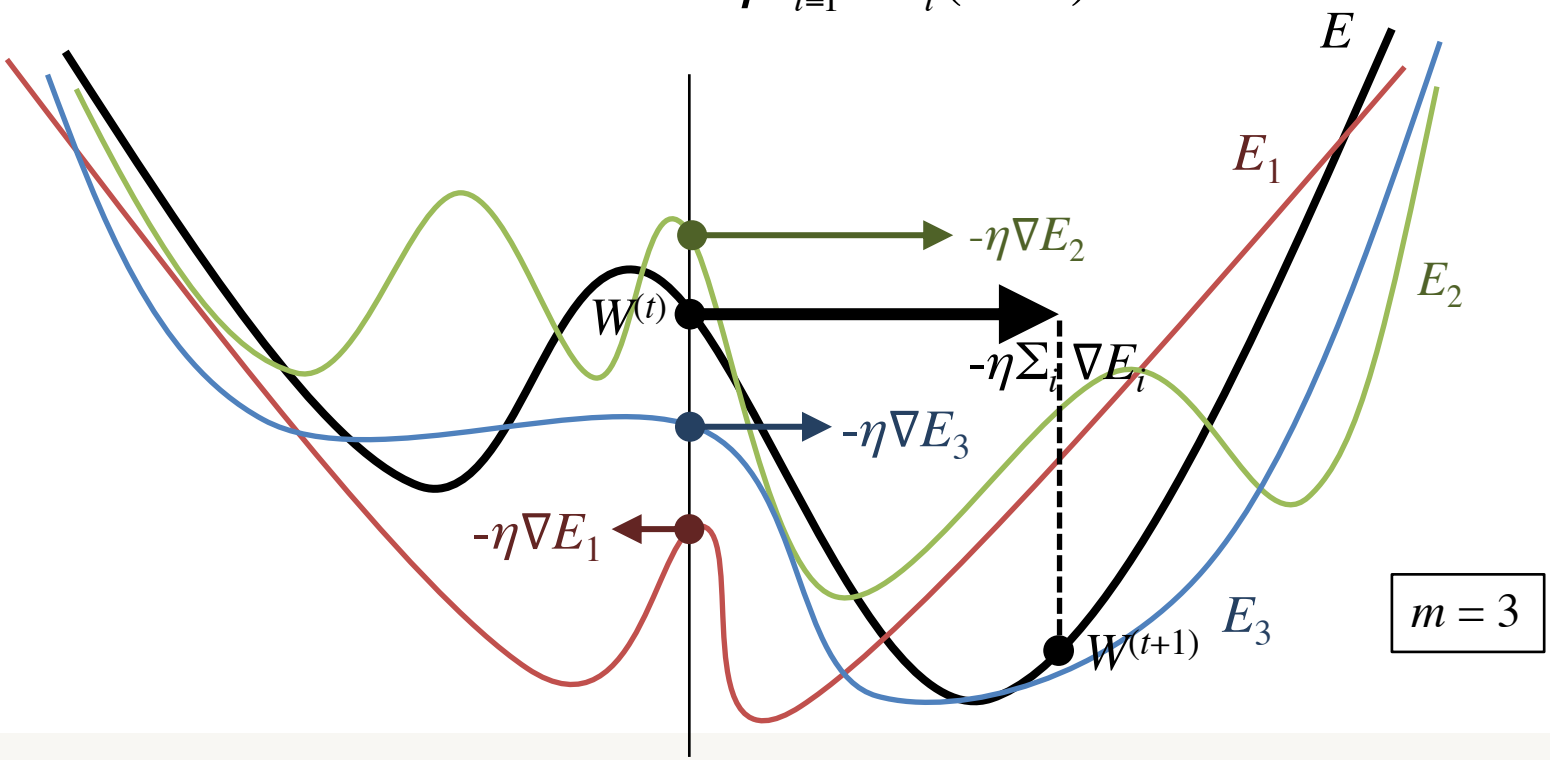
Barn swallow = **0.95**
Police dog = 0.03
Water beetle = 0.01
⋮

出力(クラス)

研究背景
Stochastic Gradient Descent (確率的勾配降下法)

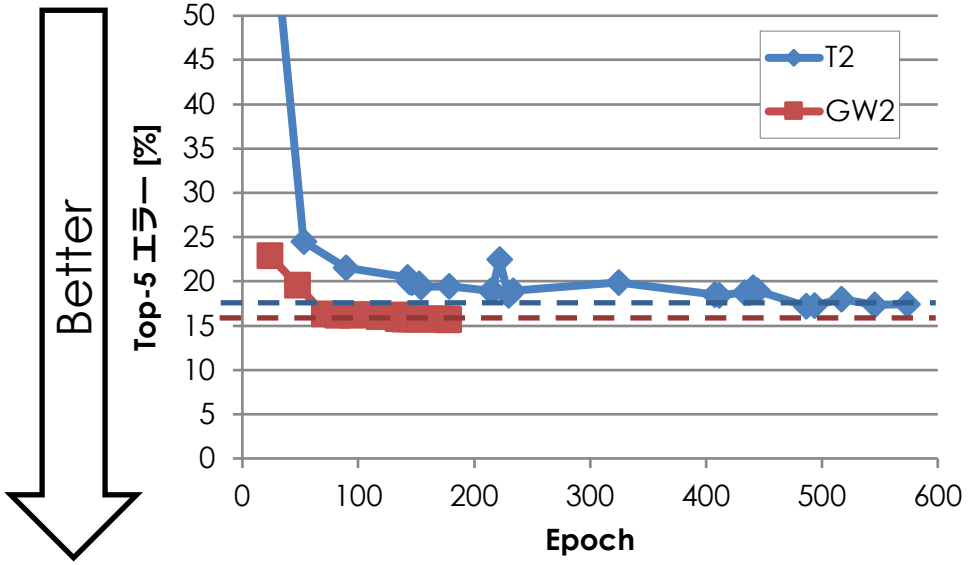
- データセット中の m 個のサンプル(ミニバッチ)から計算したコスト関数の勾配 ∇E_i の総和を用いてNNの重み $W^{(t)}$ を更新する手法
  - 大域的な勾配 ∇E の計算量(数十～数百PFlop)が膨大なDLには最適

$$W^{(t+1)} = W^{(t)} - \eta \Sigma_{i=1}^{m} \nabla E_i(W^{(t)})$$



研究背景
ミニバッチサイズと認識性能

- ミニバッチサイズがNNの精度(認識性能)と学習時間に影響
  - 小さいミニバッチサイズ → パラメータの収束に多数の反復が必要
  - 大きいミニバッチサイズ → 勾配のランダム性の低下により局所最適解に陥りやすい
    - 並列計算によるミニバッチサイズの増加で(学習時間が短縮される一方で)認識性能が悪化する可能性

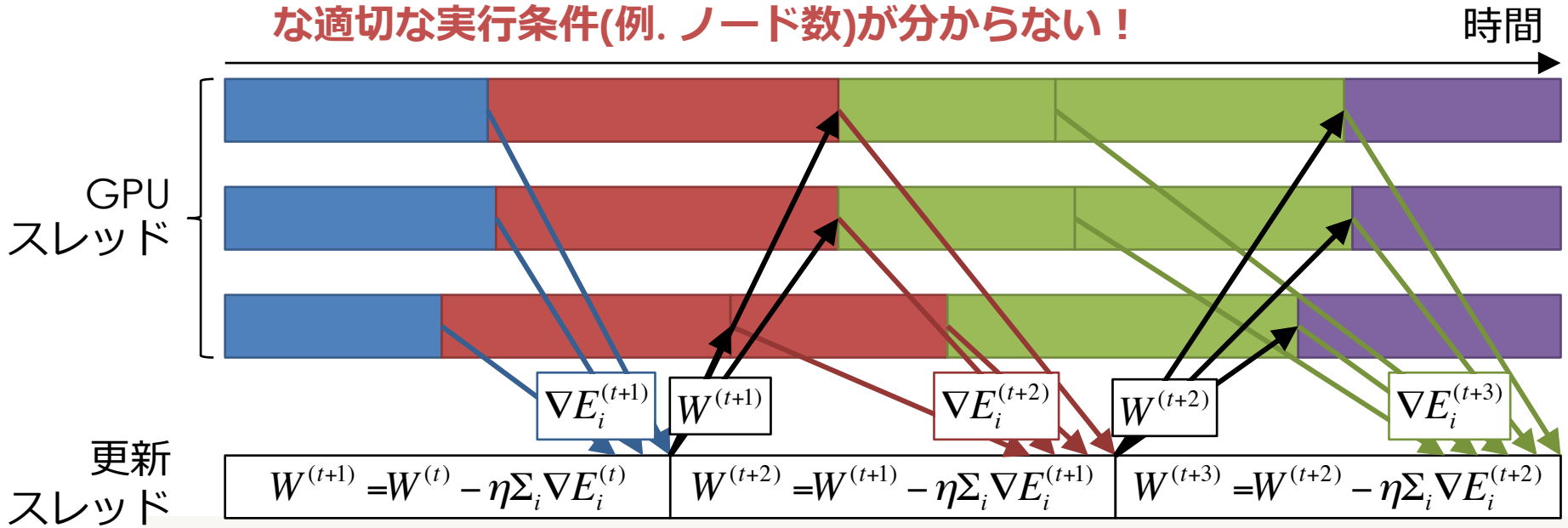


デンソーらが行った11層CNNの
ImageNet2012データセットの学習結果

	T2	GW2
GPU数	48	1
学習時間 [日]	6.04	43.8
Epoch時間 [秒]	910	28235
平均ミニバッチサイズ	1076	50
最良 top-5 エラー[%]	17.2	15.7

研究背景
SuPeRcomputation In Neural Training (SPRINT)

- 株式会社デンソーおよびデンソーアイティーラボラトリが開発した画像認識のための大規模DLシステム
- **GPUスレッド**が独立して勾配を計算し，**更新スレッド**が非同期的に全ノードのパラメータを更新 (Asynchronous SGD, ASGD)
  - スレッドの実行速度によりミニバッチサイズが動的に定まる
    - **ミニバッチサイズが自明でない → 学習の質・速度を損なわないような適切な実行条件(例. ノード数)が分からない！**



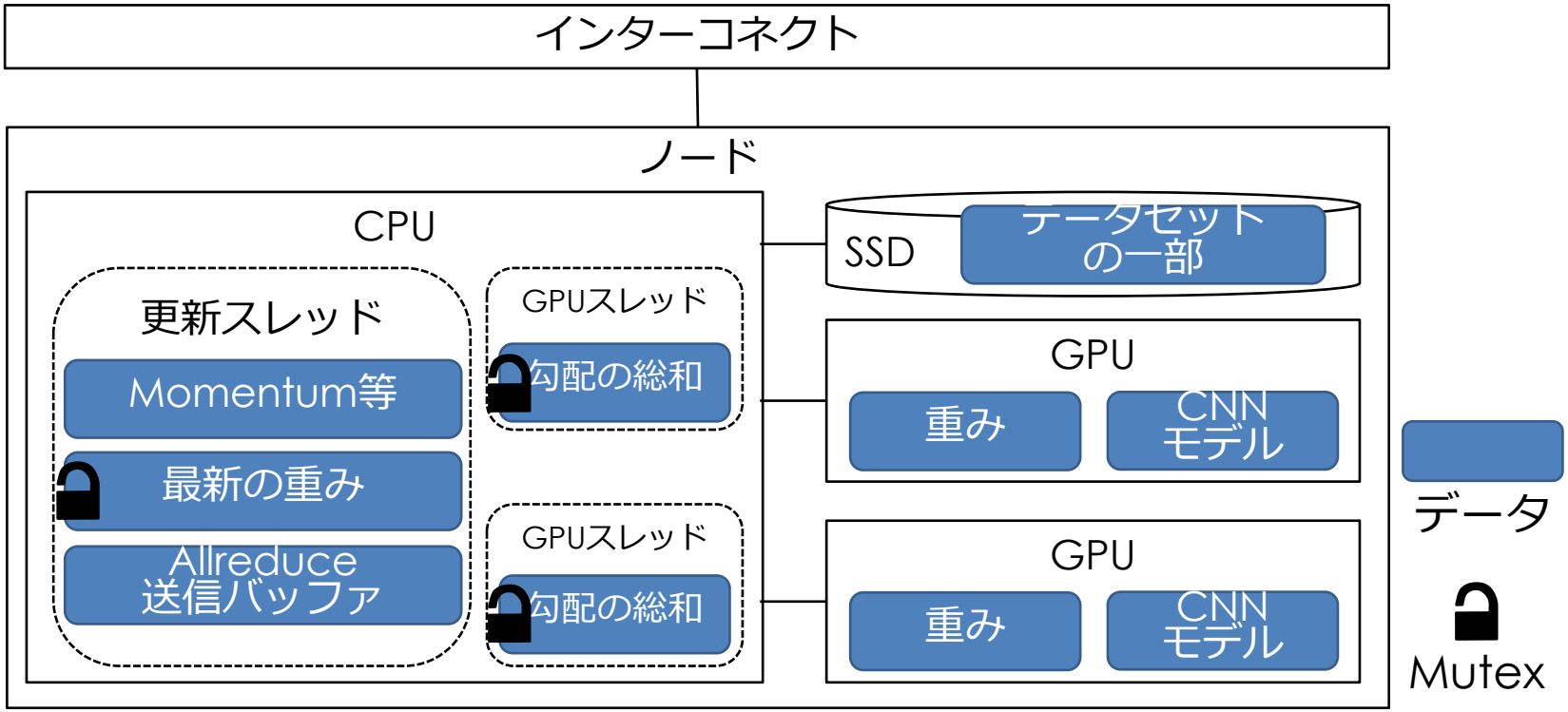
研究目的と成果

- **目的**: 適切な実行条件が自明でないASGDでConvolutional Neural Network (CNN)の教師あり学習を行うDLシステムに対し，平均ミニバッチサイズを考慮した性能モデルを構築する
- **提案手法**: DLシステム"SPRINT"の部分ごとの実行時間を計測し，回帰分析によりEmpiricalな性能モデルを提案
  - マシン設定，CNN構造を入力としてEpoch時間(データセット全体を学習する時間)と平均ミニバッチサイズを出力
- **成果**:
  - TSUBAME 2.5とTSUBAME-KFC/DLでEpoch時間，平均ミニバッチサイズを平均誤差6%, 8%で予測
  - 複数のCNNについて勾配計算時間を平均誤差12%以下で予測
  - 平均ミニバッチサイズを基準として異なる実行環境での学習時間を比較する手法を提案

SPRINTの構造

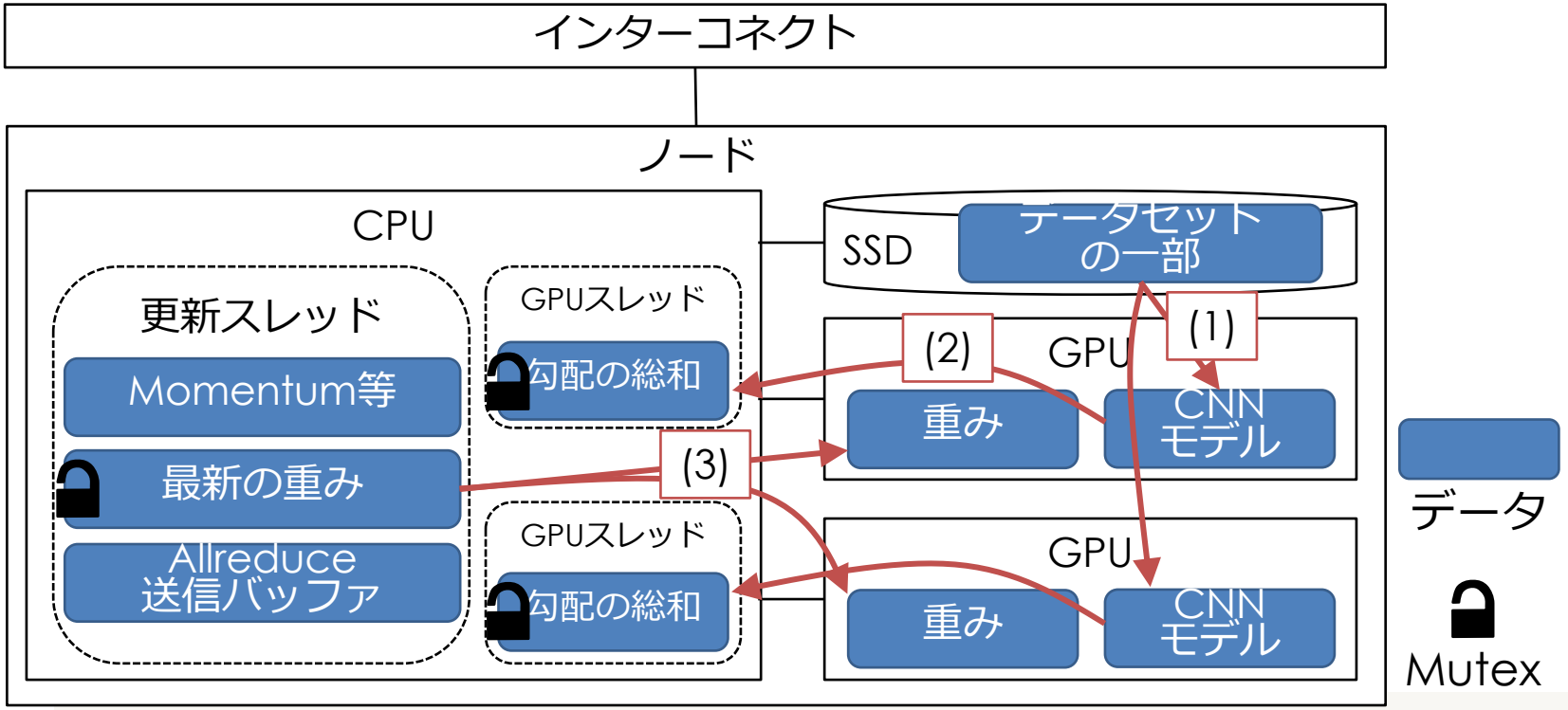
SPRINTの構造

- GPUスパコンで動作
  - 各ノードは1個以上のGPUとSSDを持つ
  - データセットはSSDに分割して配置



SPRINTの構造
GPUスレッド

1) $N_{Subbatch}$ 個のデータサンプルをSSDからデバイスメモリにコピー
2) 勾配(数十MB)を計算してホストメモリに積算
   - 行列積を $N_{Subbatch}$ 個のサンプルについて一括して計算する
3) 最新の重みが更新されている場合はデバイスメモリにコピー

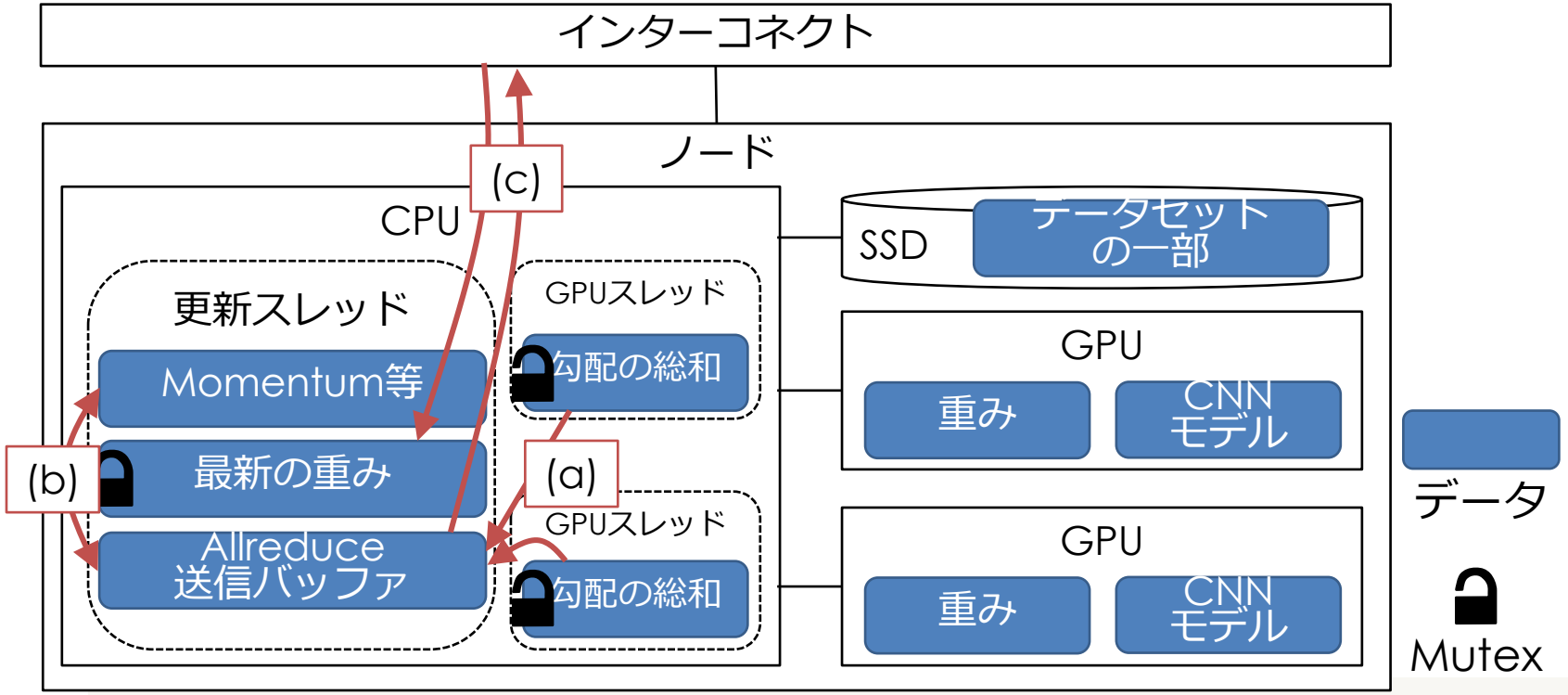


SPRINTの構造
更新スレッド

a) GPUスレッドの勾配の総和をバッファに積算

b) ノードの担当部分のみ重み(勾配)のベクトル演算

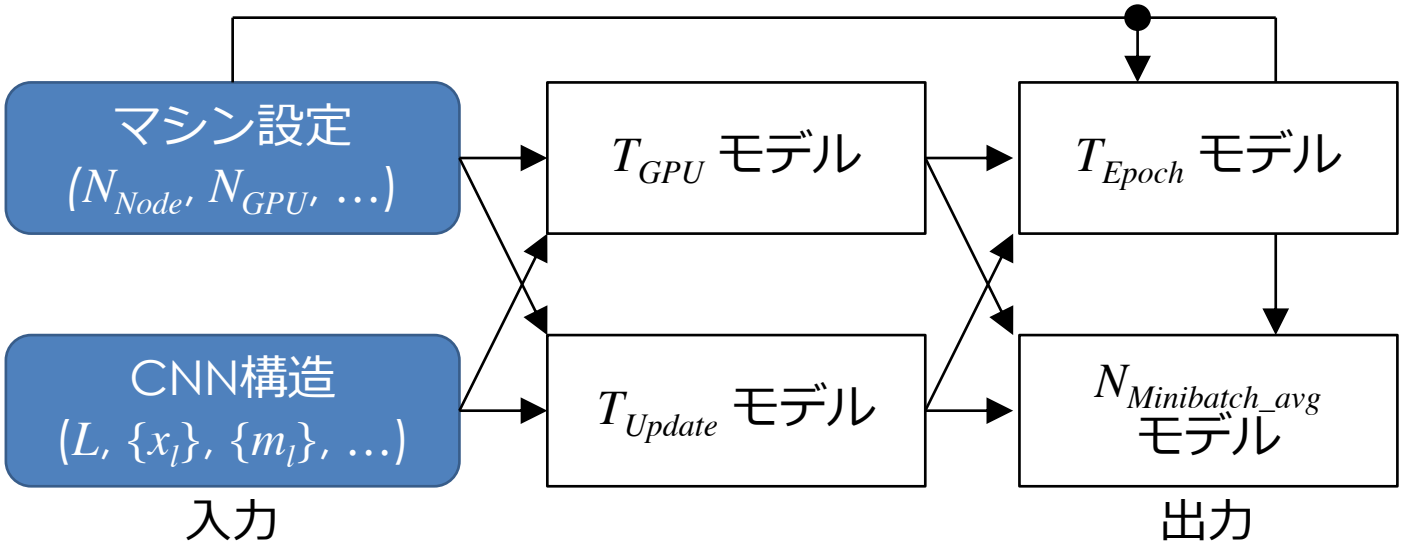
c) Allreduce通信で重みを更新



提案する性能モデル

提案する性能モデル
概要

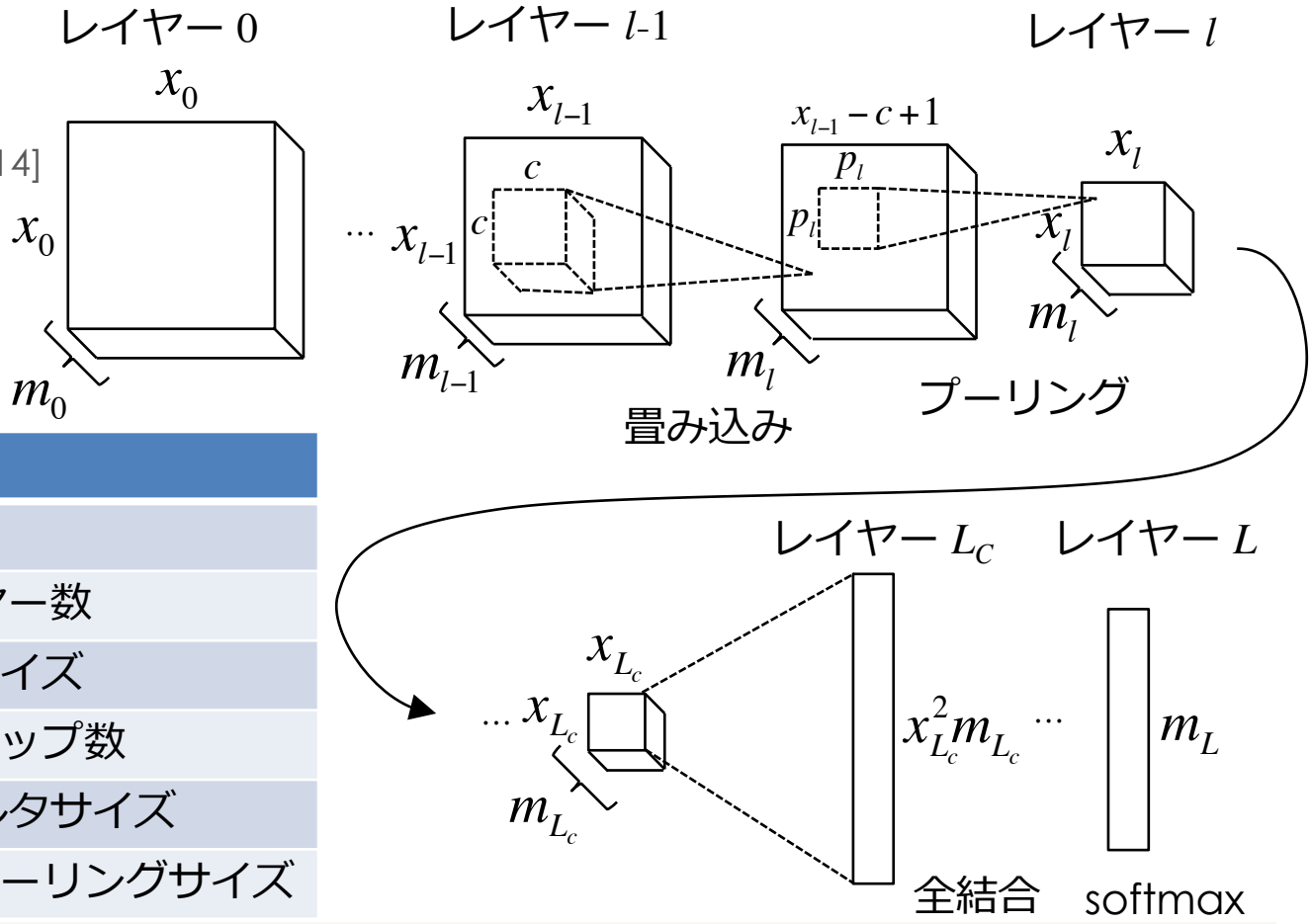
1. ノード数 N_{Node}，ノード内GPU数 N_{GPU}，CNN構造などを入力
2. GPUスレッド，更新スレッドの1イテレーションの平均実行時間 T_{GPU}, T_{Update} を予測
3. T_{GPU}, T_{Update} からEpoch時間 T_{Epoch}，平均ミニバッチサイズ $N_{Minibatch\_avg}$ を予測



提案する性能モデル
CNN構造変数

- 畳み込み層，プーリング，全結合層，softmaxで構成されるCNNに対応
  - 例: VGG

[Simonyan, 2014]

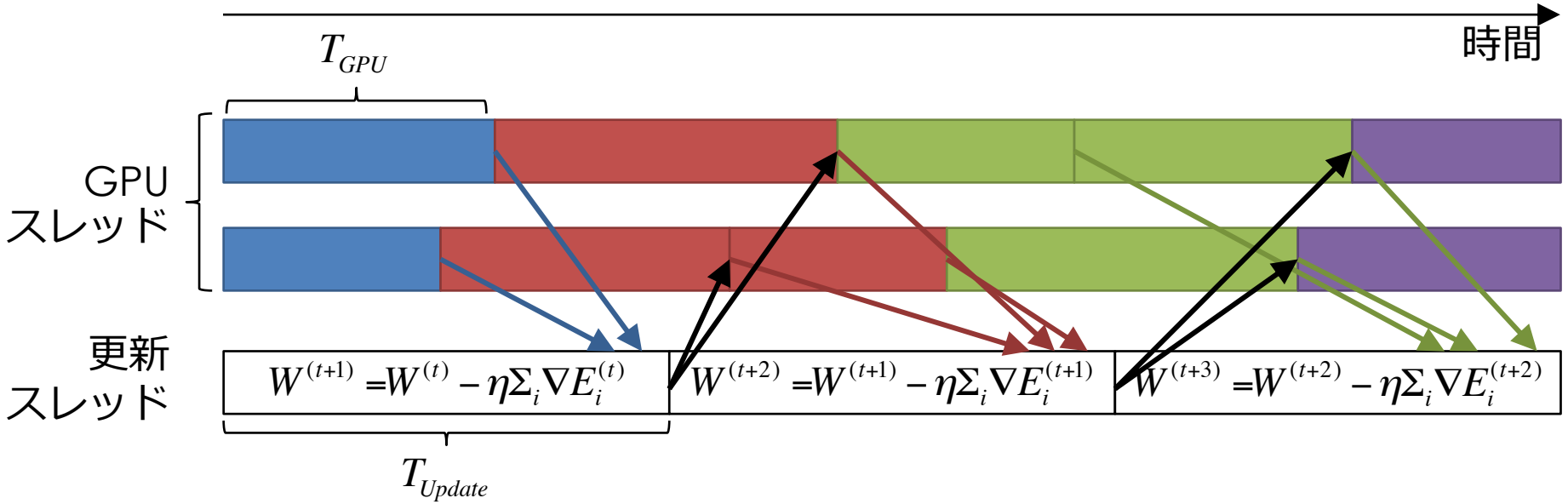


入力変数	意味
L	レイヤー数
L_C	畳み込みレイヤー数
x_l	レイヤー l のサイズ
m_l	レイヤー l のマップ数
c	畳み込みフィルタサイズ
p_l	レイヤー l のプーリングサイズ

提案する性能モデル
スレッドの実行時間

- 各スレッドの1イテレーションあたりの実行時間を分解し，それぞれを線形モデル等で表現
  - モデル中の係数は実測値を用いて決定

$$T_{GPU} = T_{LoadImage} + T_{ComputeGradient} + T_{UpdateGradient} + \cdots$$
$$T_{Update} = T_{SumGradient} + T_{Allreduce} + T_{UpdateWeights} + \cdots$$

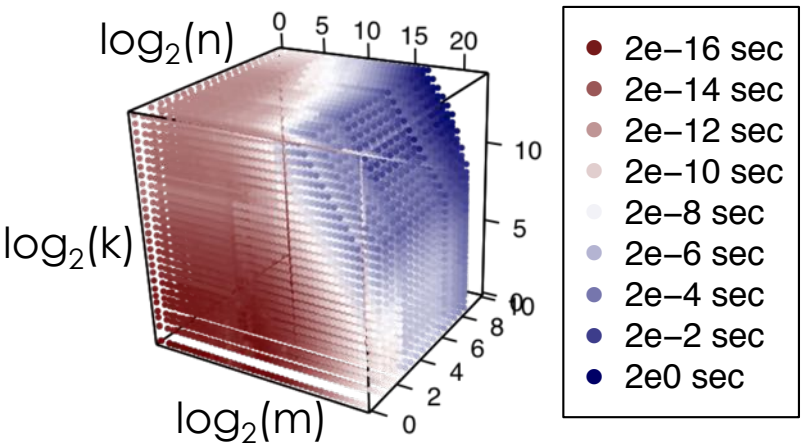


提案する性能モデル
GPUスレッド・勾配計算時間($T_{ConputeGradient}$)

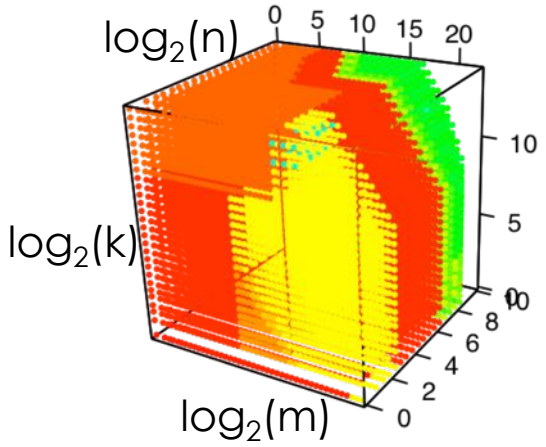
- レイヤーごとの2種類のカーネルの実行時間の総和でモデル化
  - 15層のCNNで計100回以上のカーネル呼び出し
  - **SGEMM(単精度行列積)カーネル**
    - 行列サイズはCNN構造・$N_{Subbatch}$により決定
    - convolution: 畳み込み層Feed-forward計算
    - fc: 全結合層Feed-forward計算
    - dedw, dedb, dedx_*: Back-propagation計算
  - **CUDAカーネル**
    - 配列サイズはCNN構造・$N_{Subbatch}$により決定
    - im2col: 畳み込み用の並び替え
    - activation: 活性化関数適用
    - pooling: max-pooling
    - c2f: 畳み込み層→全結合層の並び替え
    - ⋮

提案する性能モデル
GPUスレッド・勾配計算時間($T_{ConputeGradient}$)

1. **SGEMM(単精度行列積)カーネル**
   - 実装が最適化されており行列サイズ $m \times n \times k$ のみによる予測が困難



cublasSgemmの実行時間



cublasSgemmで実行される カーネルの組み合わせ

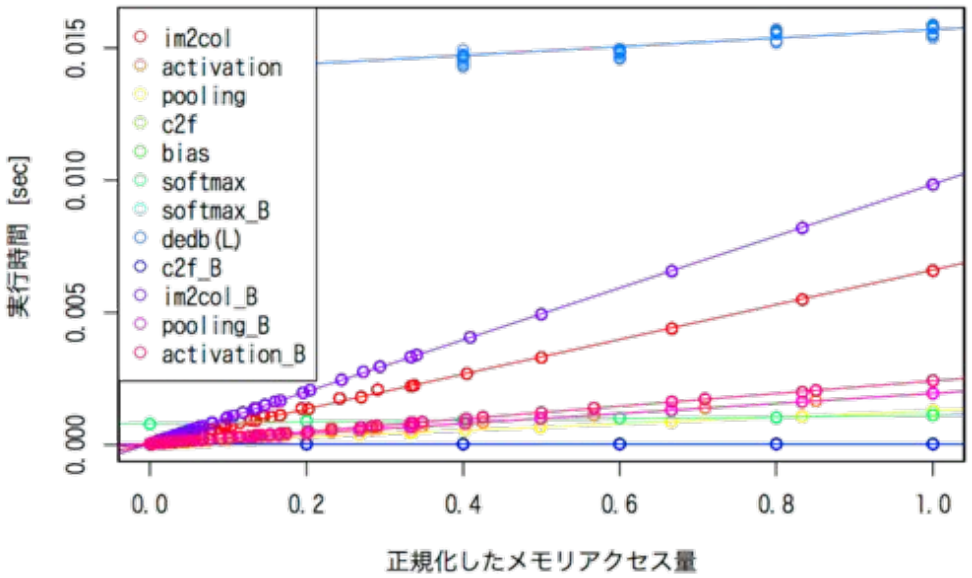
   - $2^{x/2} \times 2^{y/2} \times 2^{z/2}$ 行列積の実行時間8近傍を行列サイズの多項式で補間

$$T_{convolution}(l) = \alpha_{mnk} c^2 m_{l-1} m_l x_l'^2 N_{Subbatch} + \alpha_{mn} m_l x_l'^2 N_{Subbatch} + \alpha_{mk} c^2 m_{l-1} x_l'^2 N_{Subbatch}$$
$$+ \alpha_{nk} c^2 m_{l-1} m_l + \alpha_m x_l'^2 N_{Subbatch} + \alpha_n m_l + \alpha_k c^2 m_{l-1} + \beta$$

提案する性能モデル
GPUスレッド・勾配計算時間($T_{ConputeGradient}$)

2. **CUDAカーネル**
   - 全てメモリアクセス量の線形モデルで実行時間をモデル化
     - 係数は実測時間から線形回帰で決定

$$T_{im2col}(l) = \alpha x_l'^2 c^2 m_{l-1} N_{Subbatch} + \beta \quad (l \le L_C)$$



CUDAカーネルの実行時間(点)と決定されたモデル(直線)

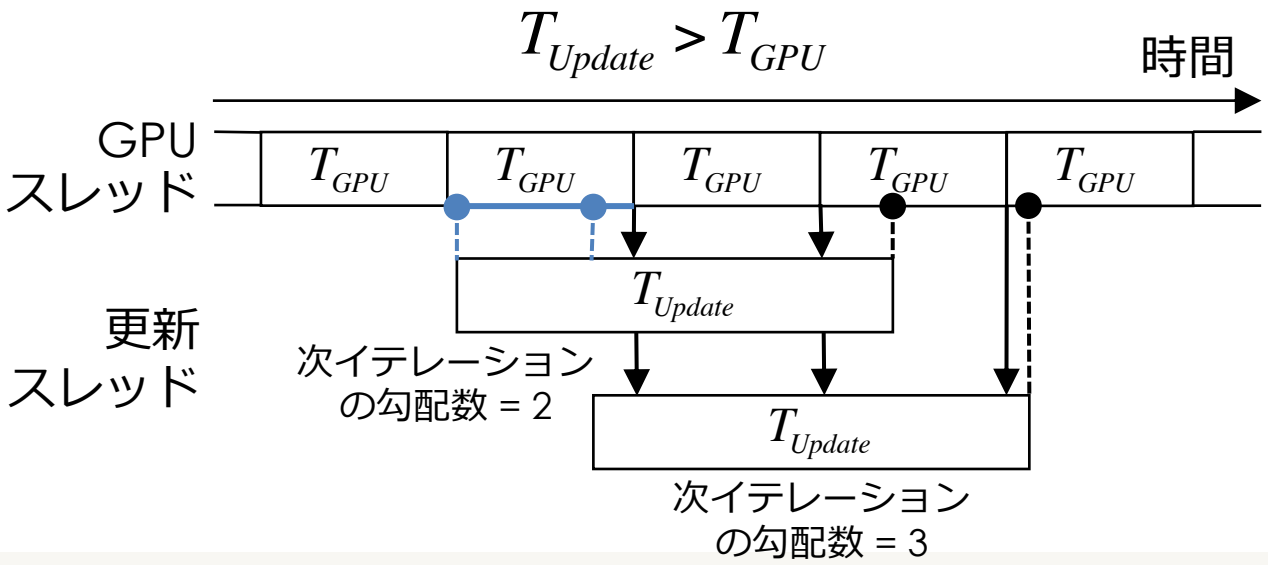
提案する性能モデル
更新スレッド・勾配加算時間($T_{SumGradient}$)

- GPUスレッドが勾配を計算し終わっている場合のみ加算する
  → 更新スレッドとの処理速度比で実行時間が動的に変化
  - スレッドの位相差が一様であると仮定し計算済み確率の期待値をモデルに使用

$$T_{SumGradient} = \alpha \times N_{Param} \times N_{GPU} \times \min(T_{Update} / T_{GPU}, 1)$$

重み(パラメータ)数
(CNN構造から決定)

勾配が計算済みである確率の期待値



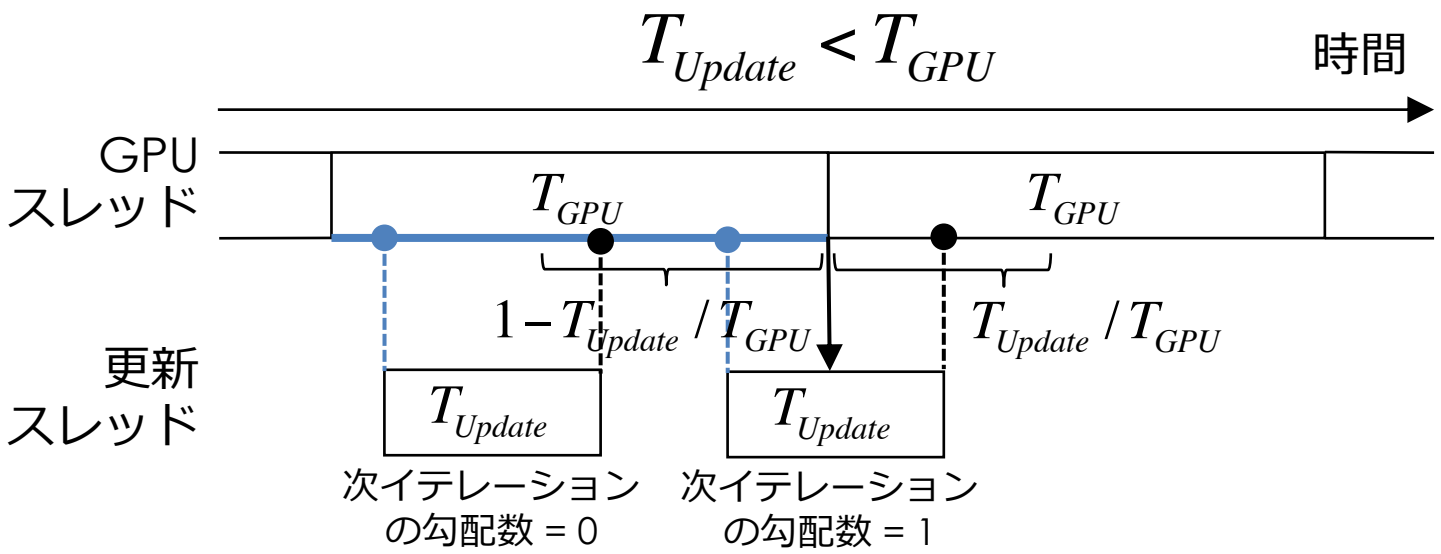
提案する性能モデル
更新スレッド・勾配加算時間($T_{SumGradient}$)

- GPUスレッドが勾配を計算し終わっている場合のみ加算する
  → 更新スレッドとの処理速度比で実行時間が動的に変化
  - スレッドの位相差が一様であると仮定し計算済み確率の期待値をモデルに使用

$$T_{SumGradient} = \alpha \times N_{Param} \times N_{GPU} \times \min(T_{Update} / T_{GPU}, 1)$$

重み(パラメータ)数
(CNN構造から決定)

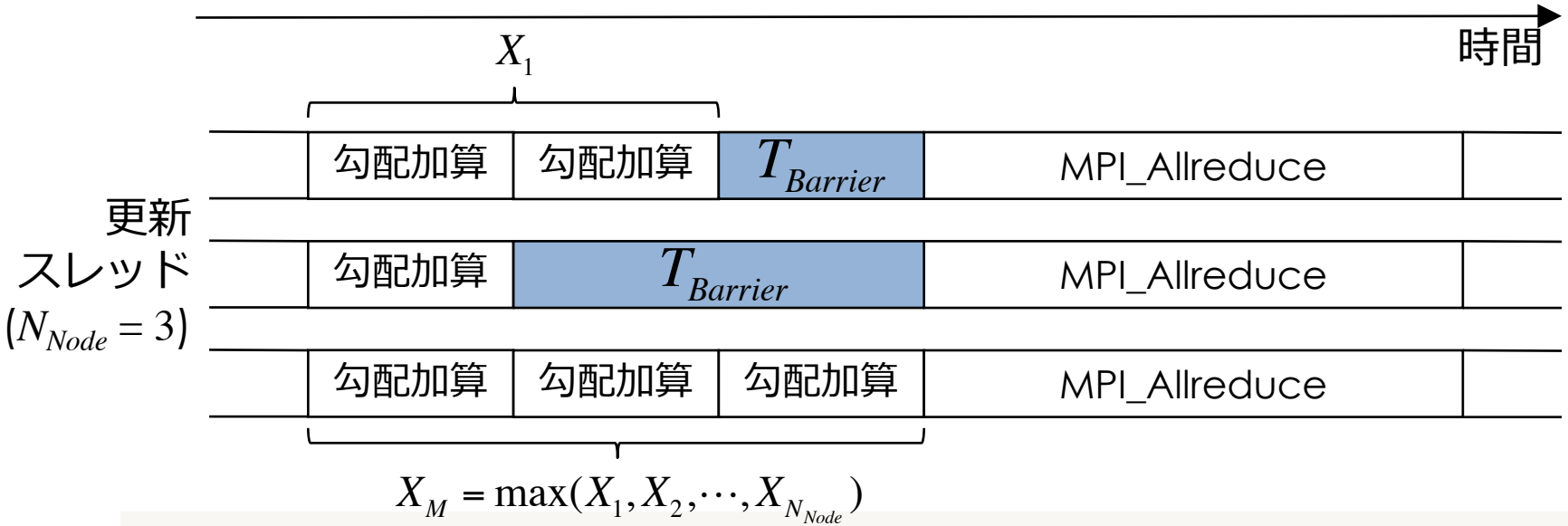
勾配が計算済みである確率の期待値



提案する性能モデル
更新スレッド・Allreduce実行時間

- 勾配加算の計算量がノードごとに一定ではない
  → Allreduce実行時に(暗黙的な)バリア同期時間が発生
  - 計算量を二項分布でモデル化し同期時間を考慮
    - $X_1 \sim B(N_{GPU}, p=\min(T_{Update}/T_{GPU}, 1))$, $F(i)$ は分布関数

$$T_{Barrier} = \alpha E(X_M - X_1) = \alpha\{N_{GPU}(1-p) - \Sigma_{i=1}^{N_{GPU}} F(i)^{N_{Node}}\}$$



提案する性能モデル
出力

- スレッドの実行時間を用いて平均ミニバッチサイズ $N_{MiniBatch\_avg}$, Epoch時間 T_{Epoch} をモデル化

1ミニバッチの処理時間

$$N_{Minibatch\_avg} = \frac{N_{Node} \times N_{GPU} \times N_{Subbatch}}{T_{GPU}} \times T_{Update}$$

単位時間あたりに処理されるサンプル数

データセットのサンプル数

$$T_{Epoch} = \frac{N_{File}}{N_{Minibatch\_avg}} \times T_{Update} = \frac{N_{File} \times T_{GPU}}{N_{Node} \times N_{GPU} \times N_{Subbatch}}$$

1サンプルあたりの処理時間

性能モデルの評価

性能モデルの評価

- TSUBAME 2.5とTSUBAME-KFC/DLでSPRINTを実行
  - 15～17レイヤーのCNN(CNN-A, CNN-B, CNN-C)を使用
  - ILSVRC2012データセットを使用
  - CNN-Aを使用した場合の実測値とSGEMM実行時間の実測値よりモデル係数を決定
  - CNN-A,B,Cを使用した場合の実測値とモデルによる予測値を比較

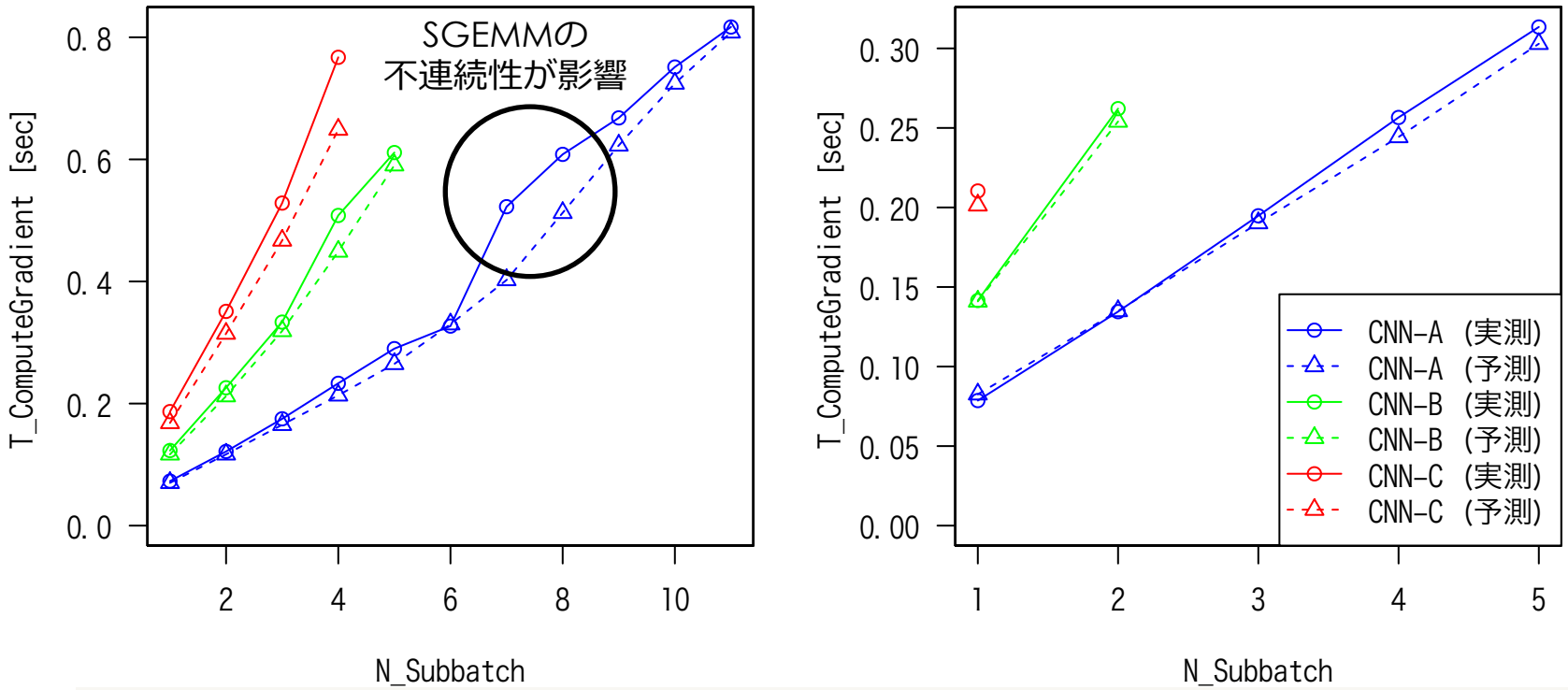
- **TSUBAME 2.5** ($N_{Node} = 1, 2, 4, \ldots, 64$)
  - Intel Xeon X5670 × 2 (計 281 単精度GFlop/s)
  - NVIDIA Tesla K20x × 3 (計 11.8 単精度TFlop/s) → $N_{GPU} = 3$
  - 4xQDR InfiniBand × 2 (計 8 GB/s)

- **TSBUAME-KFC/DL** ($N_{Node} = 1, 2, 4, \ldots, 16$)
  - Intel Xeon E5-2620 v2 × 2 (計 403 単精度GFlop/s)
  - NVIDIA Tesla K80 × 4 (計 34.9 単精度TFlop/s) → $N_{GPU} = 8$
  - 4xFDR InfiniBand (7 GB/s)

性能モデルの評価
勾配計算時間 ($T_{ComputeGradient}$)

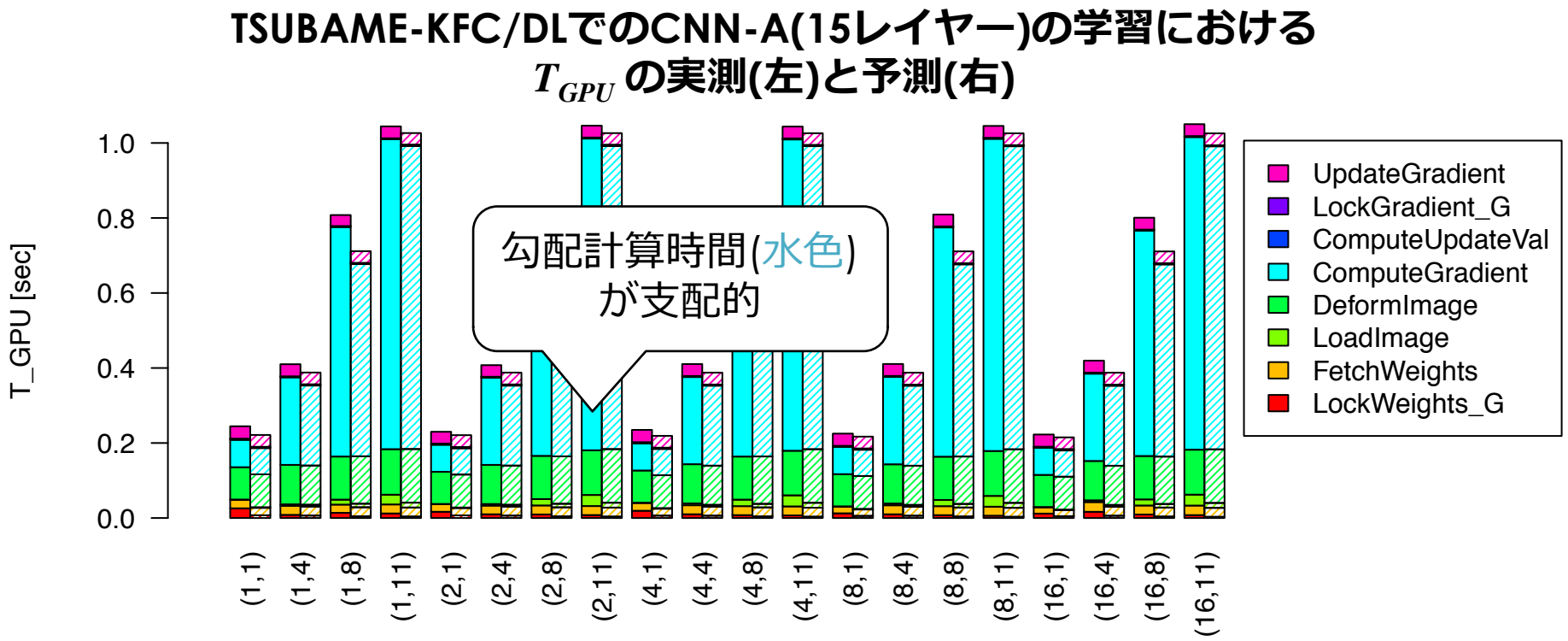
- TSUBAME 2.5とTSUBAME-KFC/DLの両方で15～17レイヤーからなるCNNの勾配計算時間の予測誤差は平均12%以下

TSUBAME-KFC/DL(左)，TSUBAME 2.5(右)での勾配計算時間($T_{ComputeGradient}$) の実測(実線)と予測(点線)



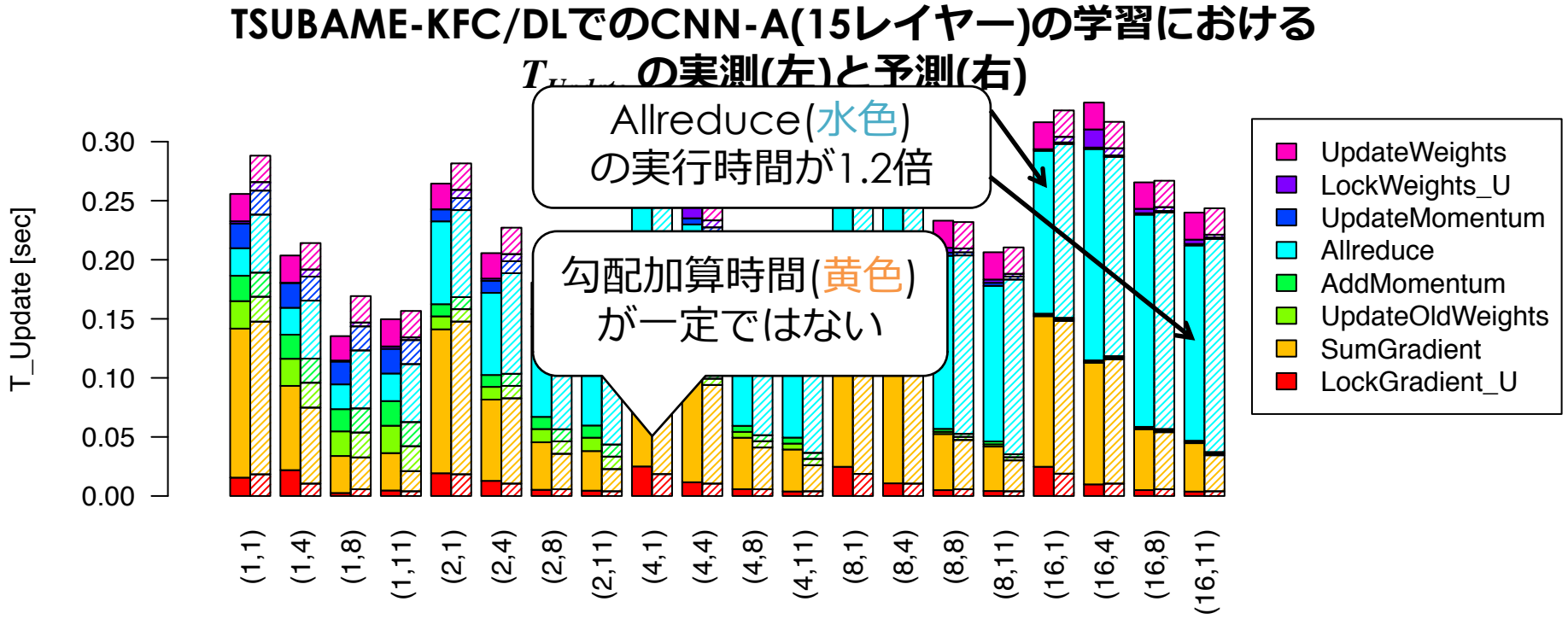
性能モデルの評価
GPUスレッド実行時間 (T_{GPU})

- TSUBAME 2.5とTSUBAME-KFC/DLの両方で最大16%以下，平均7%以下の予測誤差
  - ほぼ$N_{Subbatch}$にのみ依存



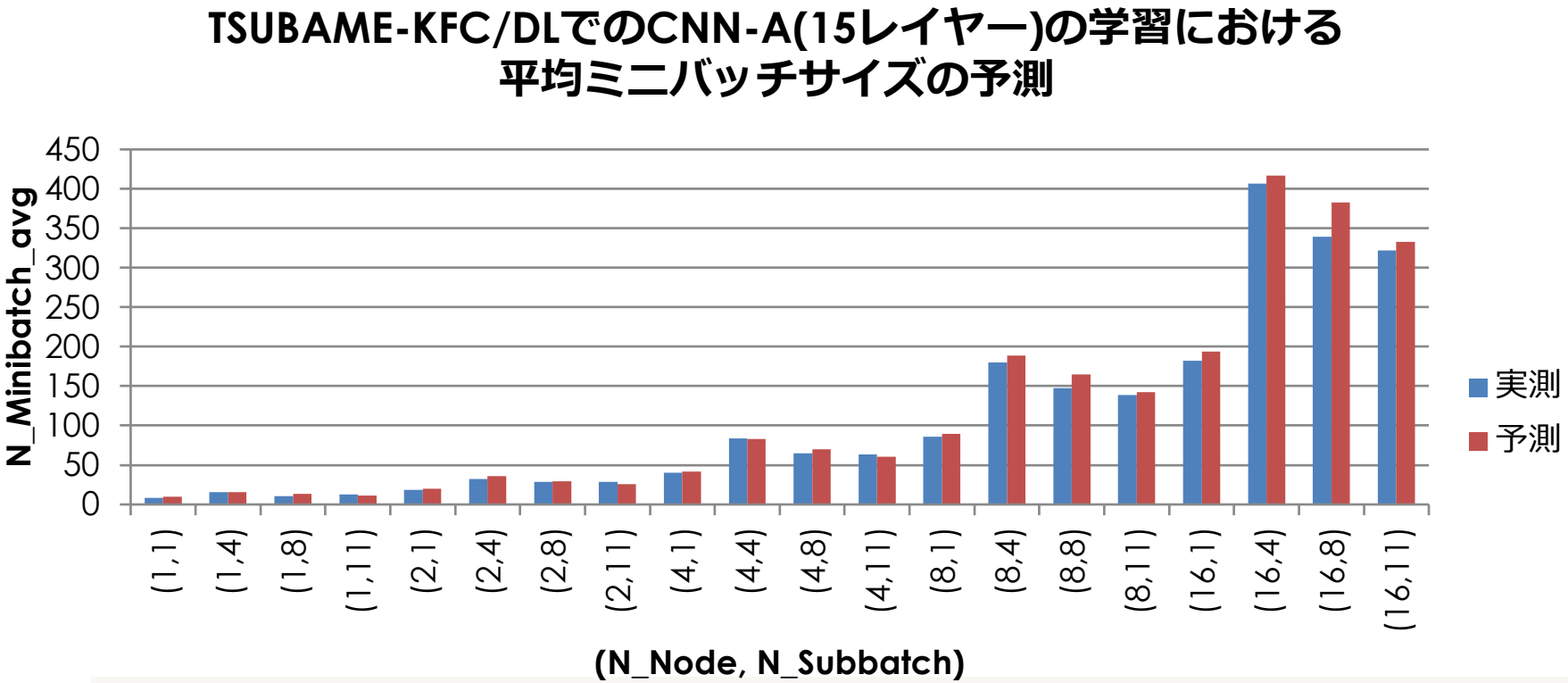
性能モデルの評価
更新スレッド実行時間 (T_{Update})

- TSUBAME 2.5とTSUBAME-KFC/DLの両方で最大33%以下，平均11%以下の予測誤差
  - N_{Node}だけでなく$N_{Subbatch}$にも依存 → モデルでよく予測されている



性能モデルの評価
Epoch時間，平均ミニバッチサイズの予測

- Epoch時間の予測誤差は平均6%
- 平均ミニバッチサイズの予測誤差は平均8%



性能モデルの評価
パラメータ探索

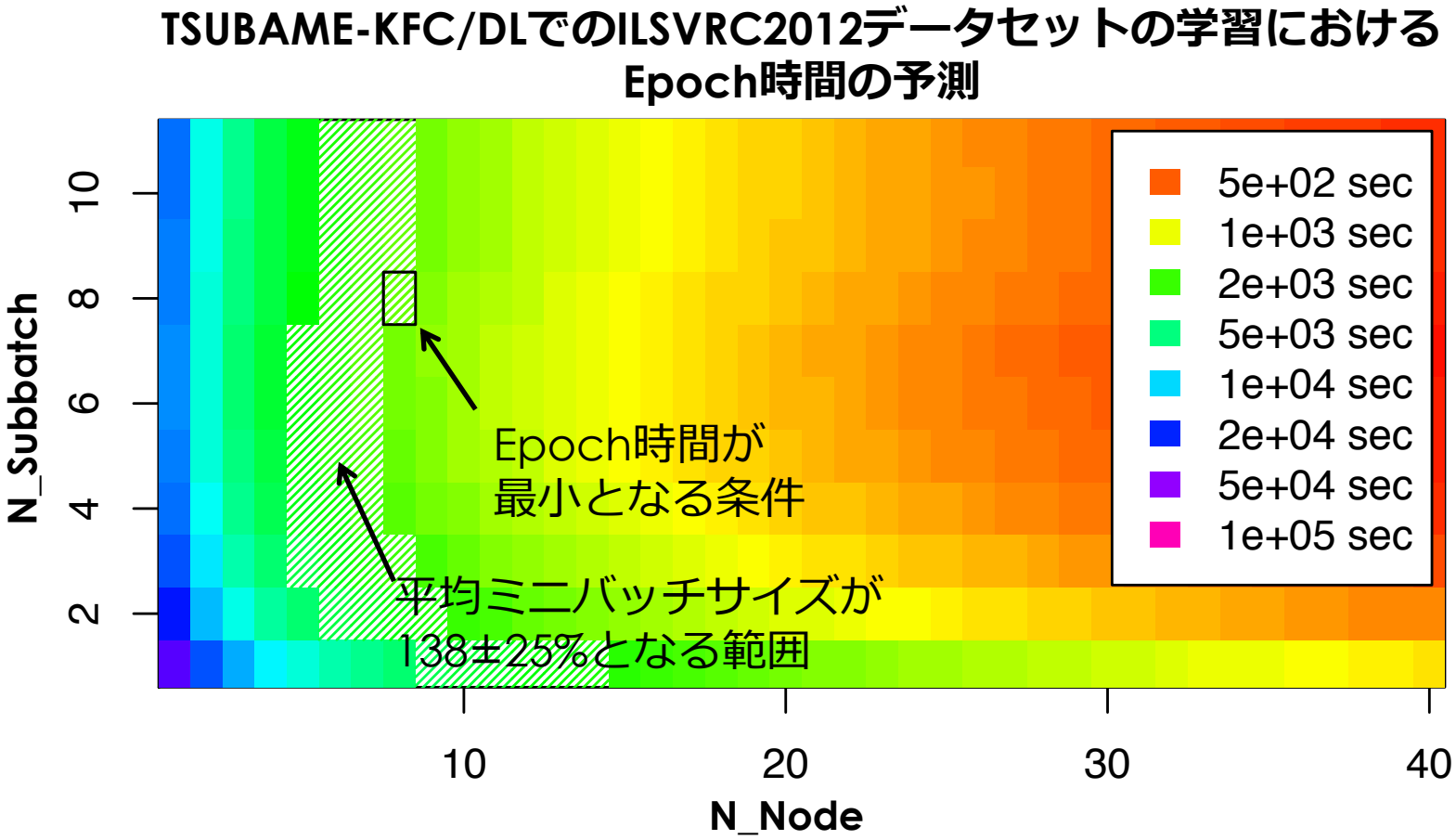
- 良い認識性能となったTSUBAME2.5 16ノードでの学習の平均ミニバッチサイズ138(実測値)の±25%以内となるパラメータを探索
  - TSUBAME-KFC/DLの8ノードで同等な学習が1.47倍の高速化
  - 実測したパラメータ範囲においてEpoch時間昇順順位が予測と一致

ILSVRC2012データセットの学習における実行性能の予測
(実測した範囲のみEpoch時間昇順で表示)

	N_{Node}	$N_{Subbatch}$	Epoch時間 [sec]			平均ミニバッチサイズ		
			実測 [sec]	予測 [sec]	予測誤差 [%]	実測	予測	予測誤差 [%]
KFC	8	8	2025	1779	12.1	147	165	12.2
KFC	8	11	2316	2226	3.90	173	171	1.28
T2.5	16	5	2725	2614	4.06	128	125	2.29
T2.5	16	4	2910	2840	2.40	116	118	1.71
			1.35倍	1.47倍		138±26%	138±25%	

性能モデルの評価
パラメータ探索

- 性能モデルを用いることで実行可能範囲の定量的な予測が可能に



関連研究・まとめ

関連研究

- パラメータサーバを用いたDLシステムの性能モデリング [Yan et al, SIGKDD, 2015]
  - パラメータサーバ(NNの最新の重みを保持するサーバ)を用いるDLシステムの性能モデルを提案
  - モデル並列・データ並列・パラメータサーバ並列の並列数とNN構造からEpoch時間を予測
  - 非同期性による学習の質(ミニバッチサイズ)は考慮されていない
- 学習時間と精度のトレードオフに関する研究 [Gupta et al, 2015]
  - パラメータサーバの構造を基本とした非同期並列DLシステムRudraを提案
  - Rudraを用いた学習実験でNNの認識性能がミニバッチサイズと平均staleness (勾配を計算する間に重みが更新される回数)の両方に影響されることを示した
  - 一般にミニバッチサイズ・stalenessと認識性能の関係は明らかではない

まとめと今後の課題

- まとめ
  - TSUBAME 2.5とTSUBAME-KFC/DLでEpoch時間，平均ミニバッチサイズを平均誤差6%, 8%で予測
  - 複数のCNNについて勾配計算時間を平均誤差12%以下で予測
  - 平均ミニバッチサイズを基準として異なる実行環境での学習時間を比較する手法を提案
- 今後の課題
  - モデル並列等の並列化手法を用いた場合の性能予測
  - 実際の学習結果を統合したより高度な性能予測