Vol.2016-HPC-155 No.5
2016/8/8

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]

概要：機械学習による画像認識において Convolutional Neural Network (CNN) と大規模なデータセットを用いた高い認識結果が報告されている．CNN の学習にはミニバッチ Stochastic Gradient Descent (SGD) と呼ばれる最適化手法が広く用いられるが，不適切なミニバッチサイズ下では認識性能が悪化することが知られている．SGD を高速化するために GPU での CNN の計算とパラメータの更新を非同期に行う非同期 SGD が提案されているが，ミニバッチサイズが動的に定まることからノード数等の学習条件の最適値は明らかではない．本論文では非同期 SGD で CNN の学習を行うシステム SPRINT の性能モデルを提案する．この性能モデルは CNN の構造とマシン性能・構成を入力とし，データセット全体を学習に使用する時間と平均ミニバッチサイズを予測する．TSUBAME-KFC/DL の 1〜16 ノードを用いた評価では複数の CNN 構造について学習時間と平均ミニバッチサイズの平均予測誤差は 8%以下だった．また，2 つの異なるマシン上である平均ミニバッチサイズの範囲内で学習時間が最短となる学習条件を探索したところ，モデルが予測した順位は実測での順位と一致した．

1. 背景

近年，Deep Learning (DL) と呼ばれる Deep Neural Network (DNN) を用いた機械学習手法が画像認識 [1] や音声認識 [2] の分野で優れた認識性能を達成している．DNN の教師あり学習では勾配降下法のような最適化手法がよく用いられるが，データセットが巨大であるために収束に多数の反復が必要となり，結果的に学習に非常に長い時間がかかることが多い．

最適化手法の一つであるミニバッチ Stochastic Gradient Descent (SGD, 確率的勾配降下法) は一定個数のデータサンプルを用いて一回の重みの更新を行う手法であり，収束速度と認識精度双方の良さから DL で最も一般的に用いられている (式 1).

$$W^{(t+1)} = W^{(t)} - \eta \sum_{i=1}^{N_{Minibatch}} \nabla E_i(x_0^{r_i}; W^{(t)}) \qquad (1)$$

ここで $W^{(t)}$ はイテレーション t における DNN の重み，η は学習係数，$\nabla E_i(x_0^{r_i}; W)$ はランダムに選択したサンプル r_i についての重み W を用いた場合のコスト関数の勾配であり，サンプル数 $N_{Minibatch}$ はミニバッチサイズと呼ばれる．ミニバッチサイズは学習時間と認識精度の両方に影響することが知られれており，小さいミニバッチサイズでは収束に多数の反復が必要となる一方で大きいミニバッチサイズでは認識精度が悪化することが指摘されている [2], [3], [4], [5].

Asynchronous SGD(ASGD)[3], [6] とは勾配計算と重みの更新を複数のプロセスで非同期的に行うことで従来の SGD[1], [2], [7] をより高速化する手法である (図 **1**).

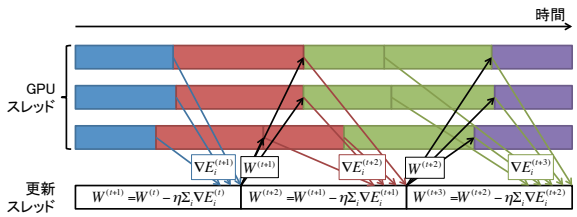

図 1　ASGD のタイムライン．GPU スレッドが勾配 $\nabla E_i^{(t)}$ (t は勾配が更新に使用されるイテレーション数) を計算する一方で更新スレッドが非同期的に重みを更新し，GPU スレッドが次回のイテレーションで更新された重みを勾配計算に使用する．

ASGD では勾配が計算されてから学習に使用されるまでに多少の遅延があり，これが学習結果に与える影響は自明ではないが，認識精度を損なわずに学習速度が向上した例も報告されている [6]．また，ASGD ではミニバッチサイズがスレッド同士の計算速度によって確率的に定まることから，DNN の構造やノード数等の実行条件に対するミニ

[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

バッチサイズの分布は自明ではなく，また認識精度を上げるためには DNN の構造をトライアンドエラーで調整する必要があることから学習条件を考慮した ASGD の学習は容易ではない．

本研究では ASGD を用いた DL システム SPRINT の性能モデルを提案する．SPRINT は GPU スパコン上で Convolutional Neural Network (CNN) の教師あり学習を行う．本モデルではミニバッチサイズの平均値を平均ミニバッチサイズと呼び，これを認識精度の指標と位置づけることで学習条件を考慮した学習時間の予測を行う．

2. 対象とする DL システム

“SPRINT” は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非同期 DL システムである．SPRINT は ILSVRC[8] 2012 データセットのアンサンブル学習において 13.67% の top-5 validation error を達成しており，TSUBAME 2.5 や TSUBAME-KFC/DL(表 **6**) で最大 96 GPU を用いた学習実績がある．

2.1 システム構成

SPRINT は各ノードが高速なインターコネクトで接続された GPU スパコン上で動作する．各ノードでは GPU 数分の「GPU スレッド」が個別のデータサンプルについてのコスト関数の勾配を計算し (データ並列)，同時に「更新スレッド」が他のノードと All-reduce 通信を行うことで重みを更新する (図 **2**)．

モデル並列やパラメータサーバを用いた更新手法は SPRINT で用いられないことから本論文では対象としない．

SPRINT では計算や通信を高速に行うために全ての小数計算を単精度浮動小数点数で行う．

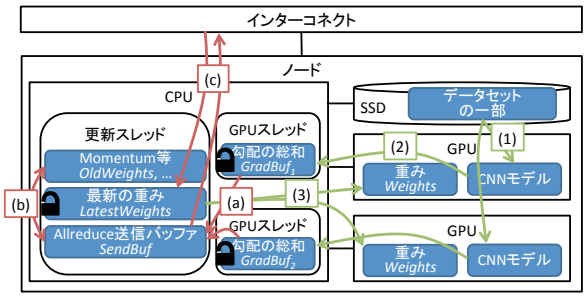

図 **2** SPRINT の構成: 緑色 (赤色) の矢印は GPU スレッド (更新スレッド) によるデータ移動を表し，錠はアクセスが mutex により制御されていることを表す．斜体の変数名は Algorithm 1，2 に対応する．

2.1.1 GPU スレッドの動作

データセットは事前にノード数で等分割され，ノード内の SSD に格納されているものとする．GPU スレッドは (1) 初めに SSD からお互いに重複しないインデックスで一定数 ($N_{Subbatch}$) のデータサンプルを読み込み，デバイスメモリに転送する．(2) 次に認識精度を上げ過学習を防ぐためにサンプルを変形し，デバイスメモリ上にある重みを用いて勾配を計算する．勾配は $-\eta$ を乗算したうえでベクトルとしてホストメモリに積算する．(3) GPU スレッドは更新スレッドから重みを更新されたことが通知された場合は次回のイテレーション開始時にホストメモリから最新の重みをデバイスメモリに転送する．

2.1.2 更新スレッドの動作

(a) 更新スレッドは複数の GPU スレッドが計算した勾配の総和を計算し (前回の計算から勾配が更新されていない場合は無視する)，勾配を計算に使用したことを GPU スレッドに通知する．(b) 次に重みをノード数で等分したうちの自ノードの担当箇所について momentum 等の計算を行う．(c) この計算で得られたベクトルに対して MPI 通信で加算 All-reduce を実行し，更新された重みを得る．この後 GPU スレッドに重みを更新したことを通知する．

3. 性能モデル

本研究では 1 章で述べた目的を達成する上でハードウェア性能を用いた理論性能よりも実際の実装・ハードウェアに即した性能の予測を重視することから，実際にアプリを実行した際の実行時間の実測値を用いる Empirical な性能モデルを構築した．

本性能モデルでは CNN の構造とマシン性能・構成を入力として GPU スレッドと更新スレッドの 1 イテレーションの実行時間を予測し，この実行時間を用いてデータセット全体を学習に使用する時間 (Epoch 時間と呼ぶ) と平均ミニバッチサイズを予測する．

各スレッドの実行時間は行うタスクに応じて複数の部分モデルに分割し，その総和で表現する．各部分モデルは最小二乗法を用いて決定するが，一部についてはスレッド間の確率的な挙動を考慮したモデル化も行った．また GPU 上での勾配計算のモデルでは更に細かいレイヤー単位のモデルを構成することで任意の CNN 構造に対してより一般性の高い予測を行えるようにした．

モデル構築に用いた実測値の計測条件は 4 で述べる．

3.1 入力パラメータ

本性能モデルの入力パラメータを表 **1** に示す．

本性能モデルでは簡単のために畳み込みフィルタサイズ $c \times c (c = 3)$ が全レイヤーで同じであると仮定する．また，$p_l = 2$ の場合は l 番目の畳み込みレイヤーの直後に $p_l \times p_l = 2 \times 2$ の Pooling を行い，$p_l = 1$ の場合は何も行わないものとする．

ILSVRC 2012 データセットを用いた場合，$m_0 = 3$ (RGB), $m_L = 1000$ (出力クラス数), $N_{File} = 1,281,167$ (総画像枚数) となる．

表 1　性能モデルの入力パラメータ

パラメータ	意味	制限
L	全レイヤー数	正の整数
L_c	畳み込みレイヤー数	L 以下の正の整数
x_l	レイヤー l サイズ	式 2 で決定する
	$(l = 0, 1, \cdots, L)$	
m_0	入力レイヤーのマップ数	データセットに依存
m_l	レイヤー l のマップ数	正の整数
	$(l = 1, 2, \cdots, L-1)$	
m_L	出力クラス数	データセットに依存
c	畳み込みフィルタサイズ	3
p_l	Pooling サイズ	1 または 2
	$(l = 1, 2, \cdots, L_c)$	
$N_{Subbatch}$	1 回のイテレーションで	デバイスメモリを
	計算されるサンプル数	超過しない数
N_{Node}	ノード数	正の整数
N_{GPU}	ノードあたりの GPU 数	正の整数
N_{File}	データセットのサンプル数	データセットに依存

3.2　CNN 構造の制限

本研究の性能モデルでは CNN のうち正方形の畳み込みレイヤー，Max-pooling，全結合レイヤー，Softmax 計算で構成されるものを対象とし (図 **3**)，レイヤーを飛び越えた接続 [9] や inception module[10] を含む特殊な CNN は対象としない．本研究の性能モデルが要求する条件を満たす代表的な CNN には VGG[11] がある．

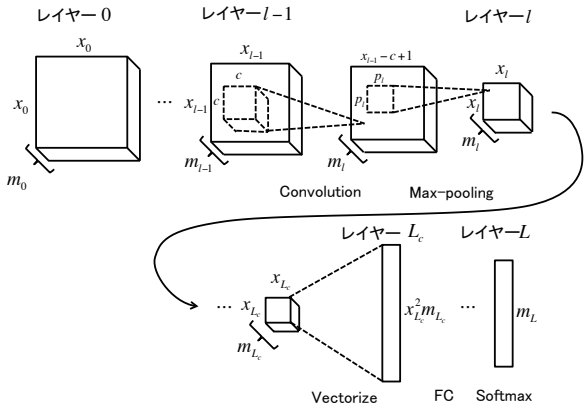

図 3　本研究の性能モデルが対象とする CNN の構造

各レイヤーのサイズ x_l $(l = 1, 2, \cdots, L)$ は以下の式で計算される．

$$x_l = \begin{cases} \frac{x_{l-1}-c+1}{p_l} & (l = 1, 2, \cdots, L_c) \\ 1 & (l = L_c+1, L_c+2, \cdots, L) \end{cases} \quad (2)$$

また，式の簡略化のために畳み込み直後 (Pooling 前) のレイヤーサイズ $x'_l = x_{l-1} - c + 1$ $(l = 1, 2, \cdots, L_c), 1(l = L_c+1, L_c+2, \cdots, L)$ をモデルに用いる．

CNN の重み (バイアスを含む) の数 N_{Param} は以下の式で計算される．

$$N_{Param} = \sum_{l=1}^{L_c} m_l(c^2 m_{l-1}+1) + \sum_{l=L_c+1}^{L} m_l({x_{l-1}}^2 m_{l-1}+1) \quad (3)$$

3.3　勾配計算の性能モデル

SPRINT はコスト関数の勾配の計算において独自の CUDA カーネルと cuBLAS ライブラリの密行列積計算関数である *cublasSgemm* を逐次的に組み合わせて用いる．例えば，畳み込み演算ではレイヤーの畳み込まれる一部分を行とした行列形式に並び替える *im2col* とその結果を用いて行列積演算を行う *convolution* を用いることで複数サンプルの畳み込み演算を同時に行う．本研究では前者を CUDA カーネル，後者を SGEMM カーネルと呼ぶ．

勾配計算の実行時間は以下の節で述べる部分式を用いて以下のモデルで推定する．

$$T_{ComputeGradient} = \sum_{l=1}^{L} \left\{ \sum_{cuda \in \{im2col, \cdots\}} T_{cuda}(l) + \sum_{sgemm \in \{convolution, \cdots\}} T_{sgemm}(l) \right\} \quad (4)$$

3.3.1　CUDA カーネルの性能モデル

CUDA カーネルはすべてメモリ律速なカーネルであることから，メモリアクセス量のオーダーの線形モデルで表す．例えばレイヤー l における *im2col* の実行時間は次の式でモデル化する．なお，レイヤー $l+1 (\leq L)$ 以降は全結合レイヤーの計算となるため実行時間は 0 とする．

$$T_{im2col}(l) = \begin{cases} \alpha {x'_l}^2 c^2 m_{l-1} N_{Subbatch} + \beta & (l \leq L_c) \\ 0 & (\text{otherwise}) \end{cases} \quad (5)$$

係数 α, β は最小二乗法により実測した実行時間から推定する．ただしメモリ配置の関係上カーネルごとに異なるメモリアクセスパターンが起こりうることから，係数はカーネルごとに独立した値を用いた．

3.3.2　SGEMM カーネルの性能モデル

SGEMM カーネルで計算される行列積の行列サイズを表 **2** に示す．なお，$m \times n \times k$ は $m \times k$ 行列と $k \times n$ 行列についての行列積計算であることを表す．

cublasSgemm を指数的な行列サイズ m, n, k で実行した際に内部で呼び出されるカーネル (プロファイラにより取得) の組み合わせと実行時間を図 **4** に示す．結果より，カーネルの組み合わせにより実行時間が不連続に変化する箇所が存在した．これは特定の行列サイズに対して適した条件で計算を行うよう最適化されているためであると考えられるが，その分岐条件は自明ではない．

本研究ではこの実行時間の不連続性を考慮し，次の方法で実行時間の予測を行う．まず行列サイズ (a^x, a^y, a^z) (a は

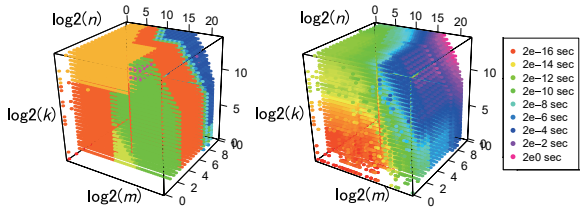

図 4　*cublasSgemm* のカーネルの組み合わせ (左) と実行時間 (右): $m \times k$, $k \times n$ は行列積を $C = AB$ とした時のそれぞれ A と B の行列サイズを表す．この行列サイズの範囲でのカーネルの組み合わせは 15 通りあった．実行時間は NVIDIA Tesla K80 で実行した際の 5 回の平均値とした．

正の定数, x, y, z は非負整数) について事前に *cublasSgemm* の実行時間を計測する．次に実行時間を予測したい行列サイズについて，行列サイズの対数の空間でその点を内包するような実測値 8 点を用いて $\{mnk, mn, mk, nk, m, n, k\}$ についての線形モデルを作成し，そのモデルから実行時間を予測する (式 6).

$$\begin{aligned} T_{convolution}(l) &= \alpha_{mnk} c^2 m_{l-1} m_l {x'_l}^2 N_{Subbatch} \\ &+ \alpha_{mn} m_l {x'_l}^2 N_{Subbatch} \\ &+ \alpha_{mk} c^2 m_{l-1} {x'_l}^2 N_{Subbatch} \\ &+ \alpha_{nk} c^2 m_{l-1} m_l + \alpha_m {x'_l}^2 N_{Subbatch} \\ &+ \alpha_n m_l + \alpha_k c^2 m_{l-1} + \beta \end{aligned} \quad (6)$$

これにより，全体で一つのモデルを用いる場合よりも実行時間の局所的な特徴をモデルに含めることができる．

また，入力行列の片方または両方が転置される場合についても呼び出されるカーネルが異なることから個別に計測を行いモデルを作成する．

表 2　SGEMM カーネルの行列サイズ

カーネル	レイヤー l	行列サイズ
convolution	$1, 2, \cdots, L_c$	${x'_l}^2 N_{Subbatch} \times m_l \times c^2 m_{l-1}$
fc	$L_c+1, L_c+2, \cdots, L$	$N_{Subbatch} \times m_l \times {x_{l-1}}^2 m_{l-1}$
dedw	$L, L-1, \cdots, 1$	${c_{l-1}}^2 m_{l-1} \times m_l \times {x'_l}^2 N_{Subbatch}$
dedb	$L-1, L-2, \cdots, 1$	$1 \times m_l \times {x'_l}^2 N_{Subbatch}$
dedx_fc	$L-1, L-2, \cdots, L_c$	$N_{Subbatch} \times {x_l}^2 m_l \times m_{l+1}$
dedx_conv	$L_c-1, L_c-2, \cdots, 1$	${x'_{l+1}}^2 N_{Subbatch} \times c^2 m_l \times m_{l+1}$

3.4　GPU スレッドの性能モデル

GPU スレッド t ($t = 1, 2, \cdots, N_{GPU}$ はノード内のインデックス) の擬似コードを Algorithm 1 に示す．提案するモデルではこのコードを 8 つの要素に分割する (表 **3**)．なお，表の α, β は要素ごとに異なる係数であり，実測値を元に最小二乗法により推定する．

排他制御の獲得にかかる時間 (*LockWeights_G*, *LockGradient_G*) は，クリティカルセクション (CS) の実行時間がイテレーション全体よりも十分に短く FCFS で獲得できる

Algorithm 1 GPU スレッド t の擬似コード

```
1: repeat
2:     LatestWeights の排他制御を獲得する
3:     if LatestWeights が前回から更新されている then
4:         Weights ← LatestWeights
5:     end if
6:     LatestWeights の排他制御を解放する
7:     画像を SSD から読み込みデバイスメモリに転送する
8:     GPU 上で画像に変形を施す
9:     Grad ← (Weights を用いて計算した勾配)
10:    Grad ← −ηGrad
11:    GradBuf_t の排他制御を獲得する
12:    if GradBuf_t が前回から使用されている then
13:        GradBuf_t ← Grad
14:    else
15:        GradBuf_t ← GradBuf_t + Grad
16:    end if
17:    GradBuf_t の排他制御を解放する
18: until プロセス終了する
```

表 3　GPU スレッドの構成要素

要素	行番号	モデル
LockWeights_G	2	$\frac{{T_{UpdateWeights}}^2}{2 \times T_{Update}} + (N_{GPU}-1)\frac{{T_{FetchWeights}}^2}{2 \times T_{GPU}}$
FetchWeights	3 - 5	$\alpha N_{Param} \times \min(T_{GPU}/T_{Update}, 1)$
LoadImage	7	$\alpha N_{Subbatch} + \beta$
DeformImage	8	$\alpha N_{Subbatch} + \beta$
ComputeGradient	9	Equation 4
ComputeUpdateVal	10	αN_{Param}
LockGradient_G	11	$\frac{(T_{SumGradient}/N_{GPU})^2}{2 \times T_{Update}}$
UpdateGradient	12 - 16	αN_{Param}

と仮定し，次の式でモデル化する．

$$\begin{aligned} T_{waiting} &= \sum_{t \in T} E(T_{waiting\_for\_t}) \\ &= \sum_{t \in T} E(T_{waiting\_for\_t} | t \text{ が CS を実行中}) \\ &\quad \times P(t \text{ が CS を実行中}) \\ &= \sum_{t \in T} \frac{T_{t\_critical\_section}}{2} \times \frac{T_{t\_critical\_section}}{T_{t\_iteration}} \\ &= \sum_{t \in T} \frac{{T_{t\_critical\_section}}^2}{2 \times T_{t\_iteration}} \end{aligned} \quad (7)$$

ここで T は競合するスレッドの集合，$T_{t\_critical\_section}$ はスレッド t のクリティカルセクションの実行時間，$T_{t\_iteration}$ は t のイテレーション全体の実行時間である．

FetchWeights では，*LatestWeights* が更新されている確率は GPU スレッドと更新スレッドのイテレーション実行時間に依存する．これを考慮してこの確率を T_{GPU}/T_{Update} ($T_{GPU} < T_{Update}$) または 1(それ以外) として決定する．

3.5　更新スレッドの性能モデル

更新スレッドの擬似コードを Algorithm 2 に示す．なお，$(x)_n$ はベクトル x のうちノード n ($n = 1, 2, \cdots, N_{Node}$)

の更新スレッドが担当する $1/N_{Node}$ の部分を表す．提案するモデルではこのコードを GPU スレッドと同様に 8 つの要素に分割する (表 **4**).

Algorithm 2 更新スレッドの擬似コード

```
repeat
  for t = 1 to N_GPU do
    GradBuf_t の排他制御を獲得する
    if GradBuf_t が前回から更新されている then
      if t = 1 then
        SendBuf ← GradBuf_t
      else
        SendBuf ← SendBuf + GradBuf_t
      end if
    else if t = 1 then
      SendBuf ← 0
    end if
    GradBuf_t の排他制御を解放する
  end for
  OldWeights ← (LatestWeights)_n
  SendBuf ← SendBuf+(OldWeights)_n+ν(DeltaWeights)_n

  RecvBuf ← MPI_Allreduce(SendBuf)
  DeltaWeights ← (RecvBuf)_n − OldWeights
  LatestWeights の排他制御を獲得する
  LatestWeights ← RecvBuf
  LatestWeights の排他制御を解放する
until プロセスが終了する
```

表 **4** 更新スレッドの構成要素

要素	行番号	モデル
LockGradient_U	3	$N_{GPU} \times \frac{T_{UpdateGradient}{}^2}{2 \times T_{GPU}}$
SumGradient	4 - 12	$\alpha N_{GPU} N_{Param} \times \min(T_{Update}/T_{GPU}, 1)$
UpdateOldWeights	15	$\alpha N_{Param}/N_{Node}$
AddMomentum	16	$\alpha N_{Param}/N_{Node}$
Allreduce	17	$T_{Barrier} + (\alpha \log_2(N_{Node}) + \beta) \times N_{Param}$
UpdateMomentum	18	$\alpha N_{Param}/N_{Node}$
LockWeights_U	19	$N_{GPU} \times \frac{T_{FetchWeights}{}^2}{2 \times T_{GPU}}$
UpdateWeights	20	αN_{Param}

SumGradient の実行時間は前回のイテレーションから新たに勾配を計算した GPU スレッドの数に依存するため，*UpdateWeights* が終了してから *Allreduce* が開始するまでの時間は更新スレッドによって異なることが予想される．これにより，*Allreduce* にはこの不均衡によって最も遅い更新スレッドを待つ時間が含まれる可能性がある．

本研究ではこの待ち時間を考慮した予測を行うために，1 イテレーションで $GradBuf_t$ が *SendBuf* に加算される回数 X_i が二項分布 $B(N_{GPU}, p)(p = \min(T_{Update}/T_{GPU}, 1))$ に従うと仮定し，最多回数 $X_M = \max(X_1, X_2, \cdots, X_{N_{Node}})$ との差の期待値を用いて待ち時間をモデル化する (式 8).

$$\begin{aligned} T_{Barrier} &= \alpha' E(X_M - X_1) \\ &= \alpha' \left\{ N_{GPU}(1-p) - \sum_{i=0}^{N_{GPU}-1} F(i)^{N_{Node}} \right\} \end{aligned} \tag{8}$$

ここで $F(i)$ は $B(N_{GPU}, p)$ の分布関数である．なお，$p = 1$ または $N_{Node} = 1$ である場合は加算回数が更新スレッド間で一定となるため $T_{Barrier} = 0$ となる．

3.6 システム全体の性能モデル

平均ミニバッチサイズは単位時間あたりに GPU スレッド全体で使用されるサンプル数が $(N_{Node} \times N_{GPU} \times N_{Subbatch})/T_{GPU}$ であることを考慮して以下の式でモデル化する．

$$N_{Minibatch\_avg} = \frac{N_{Node} \times N_{GPU} \times N_{Subbatch} \times T_{Update}}{T_{GPU}} \tag{9}$$

Epoch 時間は以下の式でモデル化する．

$$\begin{aligned} T_{Epoch} &= \frac{N_{File} \times T_{Update}}{N_{Minibatch\_avg}} \\ &= \frac{N_{File} \times T_{GPU}}{N_{Node} \times N_{GPU} \times N_{Subbatch}} \end{aligned} \tag{10}$$

4. 評価

提案するモデルを評価するために SPRINT を TSUBAME 2.5，TSUBAME-KFC/DL 上で実行して実測値との比較を行った．学習に使用するデータセットは ILSVRC 2012 データセットを用いた．評価する CNN は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設計した CNN-A, CNN-B, CNN-C を用いた (表 **5**).

表 **5** 評価に使用する CNN

	x_0	L_c	L	$\#\{l \mid p_l > 1\}$	$N_{Param}(10^6)$
CNN-A	396	15	15	5	16.1
CNN-B	396	15	15	5	12.1
CNN-C	346	17	17	5	12.5

実行環境を表 6 に示す．なお，FLOP/s は各設定下の単精度浮動小数点数の理論計算性能を表す．また，Tesla K80 は 2 つの GK210 チップを持ち，個別の 2 GPU として動作するため，本研究では TSUBAME-KFC/DL のノードあたりの GPU 数を $N_{GPU} = 8$ として扱う．

4.1 勾配計算の性能モデル

CUDA カーネルのモデルを決定するために CNN-A(と全結合レイヤーを含む類似したモデル) の各カーネルの実行時間を TSUBAME 2.5 と TSUBAME-KFC/DL で 5 分間計測した．サンプル数 $N_{Subbatch}$ は TSUBAME 2.5 で $N_{Subbatch} = 1, 2, \cdots, 5$，TSUBAME-KFC/DL で

表 6　実行環境

	TSUBAME 2.5	TSUBAME-KFC/DL
ノード数	1408	42
CPU	Intel Xeon CPU X5670 × 2 2.93 GHz, 6 コア 54 GB DDR3 メモリ	Intel Xeon E5-2620 v2 × 2 2.1 GHz, 6 コア 64GB DDR3 メモリ
GPU	NVIDIA Tesla K20X × 3 3.95 TFLOP/s 6 GB GDDR5 メモリ ECC 有効	NVIDIA Tesla K80 × 4 8.74 TFLOP/s 24 GB GDDR5 メモリ ECC 無効 Auto boost 有効
SSD	HP 572071-B21 × 2 60GB SATA3	Intel SSDSC2BB480G4 × 2 120GB SATA3
インター コネクト	4X QDR InfiniBand × 2 4 GB/s	4X FDR InfiniBand 7 GB/s
OS	SUSE Linux Enterprise Server 11 SP3	CentOS 6.4
コンパイラ	icpc 14.02	icpc 14.0.0
CUDA	CUDA 7.0	CUDA 7.0
MPI	MVAPICH2 2.0rc1	MVAPICH2 2.0rc1
pthread	NPTL 2.11.3	NPTL 2.12

$N_{Subbatch} = 1, 2, \cdots, 6$ とした．計測結果と決定されたモデルを図 **5** に示す．CUDA カーネルは静的かつ大量のメモリアクセスを行うことから実測時間とモデルがよく一致した．

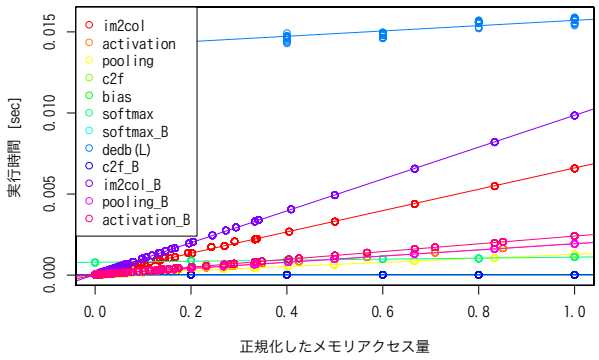

図 5　TSUBAME-KFC/DL 上の CUDA カーネルの実行時間 (点) と決定されたモデル (直線): 可読性のために初回 5 サンプルを除いた 5%のランダムに抽出した点をプロットした．x 軸は各 CUDA カーネルの最大のメモリアクセス量 (*im2col* は 1013 MB，*softmax* は 23.4 KB 等) で正規化した．

さらに，SGEMM カーネルのモデルを決定するために行列サイズを $(m, n, k) = (2^{\frac{x}{2}}, 2^{\frac{y}{2}}, 2^{\frac{z}{2}})(x, y, z$ は非負整数で推定する行列サイズより十分大きい範囲) としたときの実行時間 (5 回の実行の平均値) を測定した．この実測値を用いた時の行列サイズ $(m, n, k) = (2^{\frac{2x+1}{4}}, 2^{\frac{2y+1}{4}}, 2^{\frac{2z+1}{4}})$ についての実行時間の予測誤差を図 **6** に示す．予測誤差の第 3 四分位数は 13.6%だったが，図 4 で示したようなカーネルの組み合わせの境界ではモデルが実際に実行される組み合わせを明確に判断できないために予測誤差が他の箇所と比較して大きくなった．

決定したモデルを用いて CNN-A, CNN-B, CNN-C についての勾配計算の実行時間の予測結果を図 **7** に示す．なお，実行時間は 5 分間 SPRINT を実行した際の平均値であり，また最大の $N_{Subbatch}$ は CNN の構造とデバイスメモリ容量に依存する．

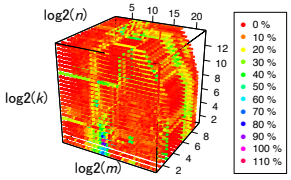

図 6　TSUBAME-KFC/DL での *cublasSgemm* の実行時間の予測誤差

$N_{Subbatch} = 6$ から 7 にかけて CNN-A の実行時間が不連続的に変化しており予測と乖離しているが，これは最も計算時間が大きいカーネルである *dedw*(2) で $N_{Subbatch} = 6$ のときに *scal_kernel* と *sgemm_largek_lds64* という (*cublasSgemm* が内部的に呼び出す) カーネルが実行されるのに対し，$N_{Subbatch} = 7$ では 2 つの *sgemm_sm35_ldg_tn_32x16x64x8x16* というカーネルが実行され，これらの実行時間の差が 0.15 秒にもおよぶためである．一方で $N_{Subbatch}$ が十分大きくなると SGEMM カーネルのモデルが後者のカーネルの実測値を予測に用いるようになるため再度予測誤差が減少する．

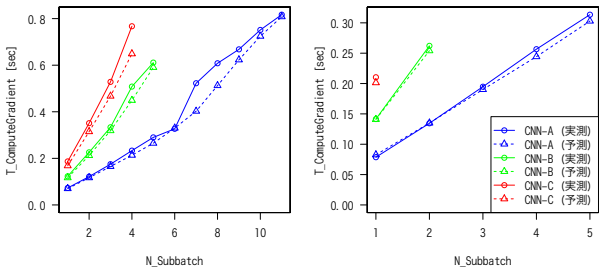

図 7　TSUBAME-KFC/DL(左) と TSUBAME 2.5(右) での勾配計算の実行時間の実測値 (実線) と予測値 (点線)

CNN-C についての $N_{Subbatch} = 4$ の際の実行時間の内訳を図 **8** に示す．SGEMM カーネルの実行時間が CUDA カーネルの実行時間と比べて大きいために，予測誤差の大部分は *cublasSgemm* の実行時間の予測誤差に由来している．

勾配計算実行時間の予測誤差を表 **7** に示す．全ての組み合わせで平均予測誤差は 12%以下だった．

表 7　勾配計算実行時間の予測誤差

	TSUBAME 2.5		TSUBAME-KFC/DL	
	最大 [%]	平均 [%]	最大 [%]	平均 [%]
CNN-A	5.02	3.17	23.0	7.42
CNN-B	3.09	1.75	11.6	6.11
CNN-C	4.29		15.4	11.8

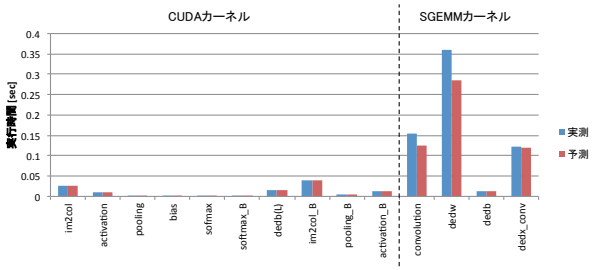

図 8　TSUBAME-KFC/DL での CNN-C の $N_{Subbatch} = 4$ の際の勾配計算の実行時間の内訳

4.2　システム全体のモデルの評価

システム全体のモデルを決定するために，CNN-A について以下の設定で各要素の実行時間を計測した．

- TSUBAME 2.5: $N_{GPU} = 3$, $N_{Node} = 2, 4, 8, \cdots, 32$, $N_{Subbatch} = 1, 2, \cdots, 5, 10$ 分間の実行 ×5 回
- TSUBAME-KFC/DL: $N_{GPU} = 8$, $N_{Node} = 2, 4, 8$, $N_{Subbatch} = 1, 4, 8, 11, 5$ 分間の実行 ×3 回

1 回の計測における各要素の実行時間の代表値は全スレッドの実行時間の平均値を用いた．さらに，特に TSUBAME 2.5 では他のジョブの実行により通信性能が実行により変化することが予想されることから，複数回の実行のうち T_{Update} が中央値を取る計測をそのパラメータ $(N_{Node}, N_{Subbatch})$ における代表値とし，これを用いてモデルの係数を決定した．

$N_{GPU} = 8$, $N_{Node} = 1, 2, 4, \cdots, 16$, $N_{Subbatch} = 1, 4, 8, 11$ で CNN-A を用いた際の GPU スレッドと更新スレッドの実行時間の予測を図 **9**，図 **10** に示す．図 9 より，T_{GPU} はほぼ $N_{Subbatch}$ にのみ依存するが，図 10 より T_{Update} は N_{Node} だけでなく $N_{Subbatch}$ にも影響されている．

$N_{Node} = 8$, $N_{Subbatch} = 4$ での実行時間の内訳を図 **11** に示す．更新スレッドでは MPI 通信を行う $T_{Allreduce}$ だけでなく $T_{SumGradient}$ もイテレーション全体の大きな割合を占めているが，これは現在の SPRINT の実装では更新スレッドが 1CPU スレッドで勾配の加算を行うためである．

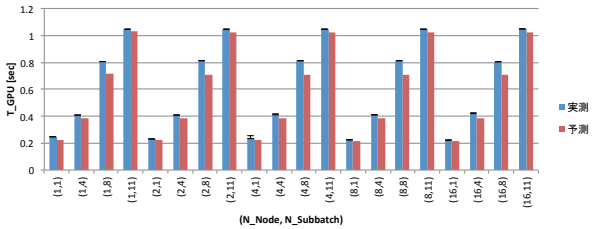

図 9　TSUBAME-KFC/DL での GPU スレッドの実行時間の実測値・予測値: ひげは同じパラメータ $(N_{Node}, N_{Subbatch})$ で実行した際の最小・最大値を表す．

TSUBAME 2.5 についても $N_{GPU} = 3$, $N_{Node} = 1, 2, 4, \cdots, 64$, $N_{Subbatch} = 1, 2, \cdots, 5$ で同様の予測を行った．これらの条件での T_{GPU}, T_{Update}, T_{Epoch}, $N_{Minibatch\_avg}$ の予測誤差を表 **8** に示す．なお，T_{Epoch} は T_{GPU} のみから定まることから T_{GPU} と同じ予測誤差になる．

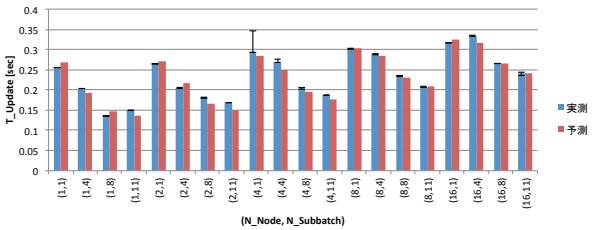

図 10　TSUBAME-KFC/DL での更新スレッドの実行時間の実測値・予測値

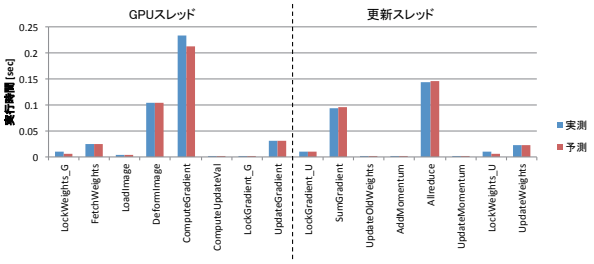

図 11　TSUBAME-KFC/DL での $N_{Node} = 8$, $N_{Subbatch} = 4$ の実行時間の内訳

表 8　CNN-A についての予測誤差

	TSUBAME 2.5		TSUBAME-KFC/DL	
	最大 [%]	平均 [%]	最大 [%]	平均 [%]
T_{GPU}	5.94	2.29	12.3	6.28
T_{Update}	33.1	11.0	11.3	4.52
T_{Epoch}	5.94	2.29	12.3	6.28
$N_{Minibatch\_avg}$	33.0	11.0	24.3	7.27

さらに，前述の実測で決定したモデルを用いて CNN-B ($N_{Node} = 1, 2, 4, 8, 16$, $N_{Subbatch} = 1, 3, 5$) と CNN-C ($N_{Node} = 1, 2, 4, 8, 16, N_{Subbatch} = 1, 4$) についても同様の予測を行った (表 **9**)．

表 9　TSUBAME-KFC/DL での CNN-B，CNN-C についての予測誤差

	CNN-B		CNN-C	
	最大 [%]	平均 [%]	最大 [%]	平均 [%]
T_{GPU}	7.68	5.77	15.7	6.83
T_{Update}	11.1	5.84	18.3	10.0
T_{Epoch}	7.68	5.77	15.7	6.83
$N_{Minibatch\_avg}$	18.7	6.24	29.1	11.6

TSUBAME-KFC/DL では CNN-A, CNN-B, CNN-C についての $T_{Epoch}, N_{Minibatch\_avg}$ の予測誤差の平均はそれぞれ 6%，8%だった．

4.3 性能モデルを用いたパラメータ探索

ASGD を用いる DL システムを実行する場合，平均ミニバッチサイズを一定範囲に保ちつつ学習時間を最小にする $N_{Node}, N_{Subbatch}$ が事前に分かっていることが望ましい．本研究で提案するモデルがこのような実行条件の探索に用いることができることを示すために，TSUBAME 2.5 と TSUBAME-KFC/DL で CNN-A の学習を行う場合の T_{Epoch} と $N_{Minibatch\_avg}$ を網羅的に予測し (図 **12**, 図 **13**)，最適な実行条件を推定した．なお，認識性能の悪化を防ぐために，学習に適した平均ミニバッチサイズの範囲を $138 \pm 25\%$ と仮定した．なお，$N_{Minibatch\_avg} = 138$ は TSUBAME 2.5 で $N_{Node} = 16, N_{Subbatch} = 5$ の条件で実際に学習を行った際の実測値である．

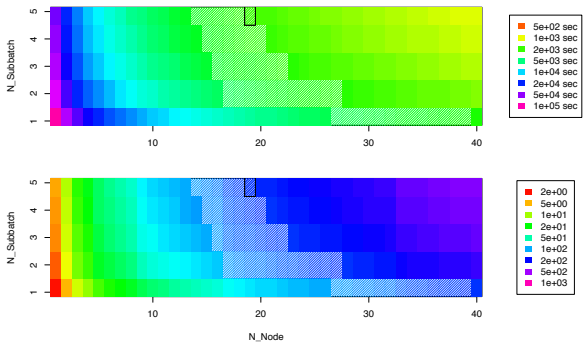

図 **12** TSUBAME 2.5 での T_{Epoch}(上) と $N_{Minibatch\_avg}$(下) の予測値: 斜線部は $N_{Minibatch\_avg}$ が $138 \pm 25\%$ の範囲であり，黒い枠はそのうちで T_{Epoch} が最小となる実行条件を表す．

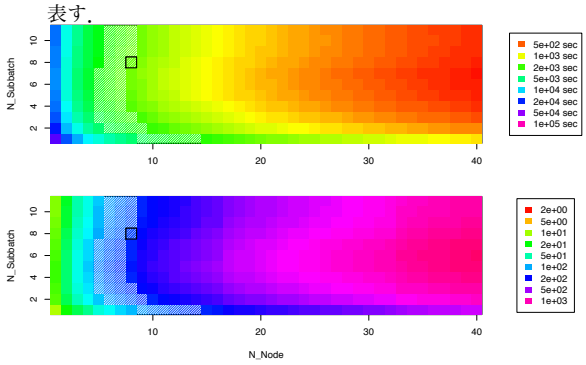

図 **13** TSUBAME-KFC/DL での T_{Epoch}(上) と $N_{Minibatch\_avg}$(下) の予測値

図 12，図 13 の斜線部の実行条件のうち，4.2 節で実測した条件について T_{Epoch} と $N_{Minibatch\_avg}$ と予測誤差を表 **10** に示す．この条件においては予測した条件の順位は実測と一致した．

5. 関連研究

[12] では CPU クラスタ上でのパラメータサーバを用いる DL システムの性能モデルが提案されている．このモデルではモデル並列・データ並列・パラメータサーバ並列の並列度を入力として Epoch 時間を予測する．しかしこのモデルでは非同期性による学習の質を考慮しておらず，また SPRINT のようなパラメータサーバを持たないようなシステムには直接適用できない．

[5] ではパラメータサーバを用いる構造を基本とした DL システム Rudra が提案されている．Rudra では勾配計算を行う learner が (階層化された) パラメータサーバと非同期的に通信し学習を行う．著者らは勾配計算で使用した重みのタイムスタンプとその勾配が更新に使用される際のタイムスタンプの差を staleness と定義し，ミニバッチサイズだけでなく staleness も学習精度に影響することをこれらのパラメータについての網羅的な学習実験で示した．本研究の性能モデルでは学習精度自体については理論的な予測が困難であることから予測しないものの，このような実験的な結果と統合することでより適切な実行条件の予測を行うことが可能になると考えられる．

表 **10** 予測された実行条件についての $T_{Epoch}, N_{Minibatch\_avg}$ と予測誤差: T2 は TSUBAME 2.5 を，KFC は TSUBAME-KFC/DL を表す．行はマシンに関わらず T_{Epoch} の昇順で表示した．

	N_{Node}	$N_{Subbatch}$	T_{Epoch} [sec] 実測	予測	誤差 [%]	$N_{Minibatch\_avg}$ 実測	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	16	5	2725	2614	4.06	128	125	2.29
T2	16	4	2910	2840	2.40	116	118	1.71
T2	32	1	3178	3227	1.54	121	125	3.17
T2	16	3	3276	3257	0.605	97.7	105	7.87

6. まとめと今後の課題

本論文では非同期 SGD で CNN の学習を行うシステム SPRINT の性能モデルを提案し，TSUBAME-KFC/DL の 1〜16 ノードを用いた評価で複数の CNN 構造について学習時間と平均ミニバッチサイズを平均予測誤差 8%以下で予測することができた．また，2 つの異なるマシン上である平均ミニバッチサイズの範囲内で学習時間が最短となる学習条件を探索したところ，モデルが予測した順位は実測での順位と一致した．

今後の課題としては，現在のデータ並列を用いた実装に加えてより並列度を上げるためにモデル並列やパラメータサーバを導入した場合の性能予測を行う必要があると考えられる．また，現在のモデルに実際の学習結果を組み合わせてより信頼性の高い予測を行う必要があると考えられる．

謝辞 本研究の一部は科学技術振興機構戦略的創造研究推進事業「EBD：次世代の年ヨッタバイト処理に向けたエクストリームビッグデータの基盤技術」による．

参考文献

[1] Wu, R., Yan, S., Shan, Y., Dang, Q. and Sun, G.: Deep Image: Scaling up Image Recognition, *CoRR*, Vol. abs/1501.02876 (online), available from ⟨http://arxiv.org/abs/1501.02876⟩ (2015).

[2] Amodei, D., Anubhai, R., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Chen, J., Chrzanowski, M., Coates, A., Diamos, G., Elsen, E., Engel, J., Fan, L., Fougner, C., Han, T., Hannun, A., Jun, B., LeGresley, P., Lin, L., Narang, S., Ng, A., Ozair, S., Prenger, R., Raiman, J., Satheesh, S., Seetapun, D., Sengupta, S., Wang, Y., Wang, Z., Wang, C., Xiao, B., Yogatama, D., Zhan, J. and Zhu, Z.: Deep Speech 2: End-to-End Speech Recognition in English and Mandarin, *ArXiv e-prints* (2015).

[3] Zhang, S., Zhang, C., You, Z., Zheng, R. and Xu, B.: Asynchronous stochastic gradient descent for DNN training, *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 6660–6663 (online), DOI: 10.1109/ICASSP.2013.6638950 (2013).

[4] Krizhevsky, A.: One weird trick for parallelizing convolutional neural networks, *CoRR*, Vol. abs/1404.5997 (online), available from ⟨http://arxiv.org/abs/1404.5997⟩ (2014).

[5] Gupta, S., Zhang, W. and Milthorpe, J.: Model Accuracy and Runtime Tradeoff in Distributed Deep Learning, *ArXiv e-prints* (2015).

[6] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M., Senior, A., Tucker, P., Yang, K., Le, Q. V. and Ng, A. Y.: Large Scale Distributed Deep Networks, *Advances in Neural Information Processing Systems 25* (Bartlett, P., Pereira, F., Burges, C., Bottou, L. and Weinberger, K., eds.), pp. 1232–1240 (online), available from ⟨http://books.nips.cc/papers/files/nips25/NIPS2012_0598.pdf⟩ (2012).

[7] Iandola, F. N., Ashraf, K., Moskewicz, M. W. and Keutzer, K.: FireCaffe: near-linear acceleration of deep neural network training on compute clusters, *CoRR*, Vol. abs/1511.00175 (online), available from ⟨http://arxiv.org/abs/1511.00175⟩ (2015).

[8] Stanford Vision Lab, Stanford University, P. U.: ImageNet. available from ⟨http://image-net.org/⟩.

[9] Srivastava, R. K., Greff, K. and Schmidhuber, J.: Highway Networks, *CoRR*, Vol. abs/1505.00387 (online), available from ⟨http://arxiv.org/abs/1505.00387⟩ (2015).

[10] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A.: Going Deeper With Convolutions, *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015).

[11] Simonyan, K. and Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition, *CoRR*, Vol. abs/1409.1556 (online), available from ⟨http://arxiv.org/abs/1409.1556⟩ (2014).

[12] Yan, F., Ruwase, O., He, Y. and Chilimbi, T.: Performance Modeling and Scalability Optimization of Distributed Deep Learning Systems, *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, New York, NY, USA, ACM, pp. 1355–1364 (online), DOI: 10.1145/2783258.2783270 (2015).