大脳皮質モデルBESOMのGPGPUによる並列化

中田秀基[1]、井上辰彦[1,2]、一杉裕志[1]

1 産業技術総合研究所人工知能研究センター

2 株式会社創夢

この成果は、国立研究開発法人新エネルギー・産業技術総合開発機構（ＮＥＤＯ）の委託業務の結果得られたものです。

背景（１）

- 大脳皮質モデルBESOM[IJCNN15 一杉]
  - ベイジアンネットによる大脳皮質のモデル化
  - 計算量大 -> 並列化が必須
  - 並列化の2つの手法
    - 複数モデルを用いたデータ並列化 [CPSY15 黎]
    - モデル内部の並列化

背景（２）

- Deep LearningではGPGPUを用いたモデル並列化が標準的
  - Tensorflow
  - Chainer
  - Caffe
  - Cudaconvnet
  - Torch
  - Pylearn2
  - ….
- DLは比較的単純な行列演算に帰着できる

ベイジアンネットベースのBESOMに同じ手法が適用可能か？

研究の目的と成果

- 目的
  - GPGPUを用いてモデル内部を並列化することで高速化を図る
  - Deep Learningで標準的な手法がBESOMにおいても有効であることを確認

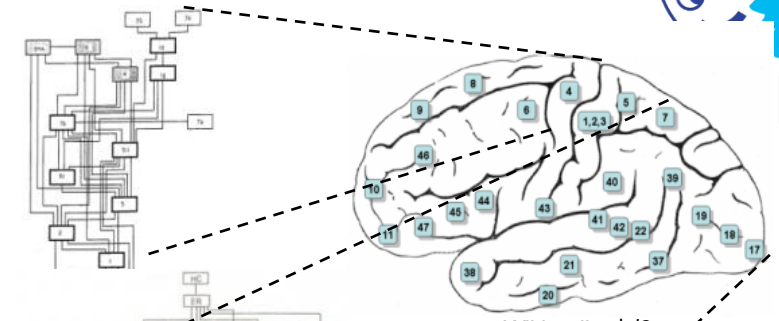
- 成果
  - およそ50倍の高速化を実現

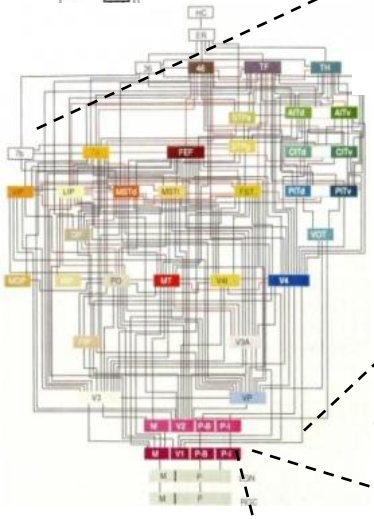
発表の流れ

- BESOMの概要
- GPGPUによる並列化
  - GPGPUの特徴
  - ミニバッチ化
- 評価
- まとめと今後の課題

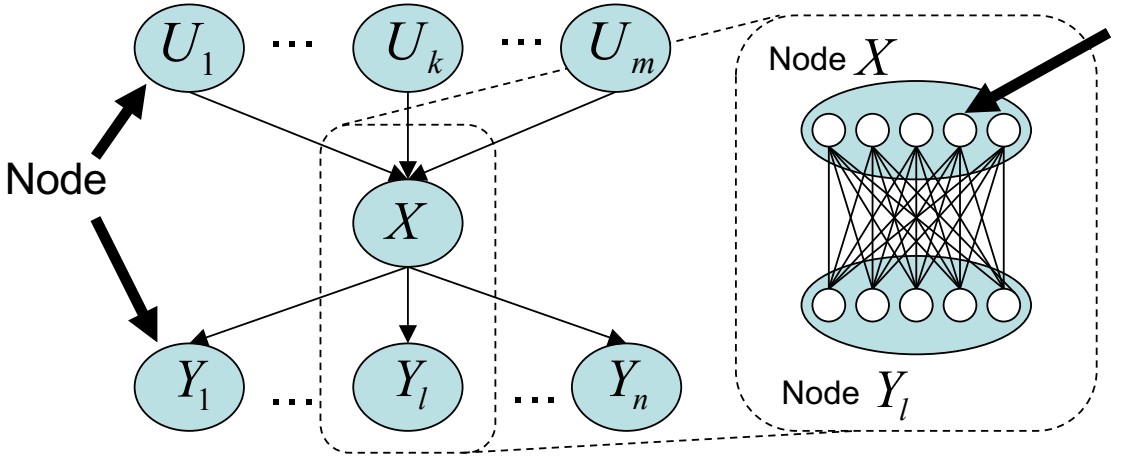
大脳皮質モデルBESOM

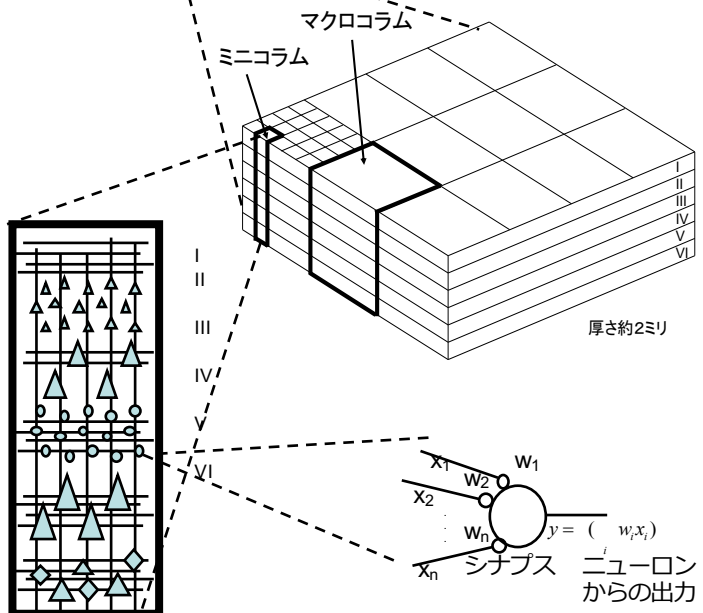
- ベイジアンネットによる
  マクロカラム間ネットワークの
  モデル化





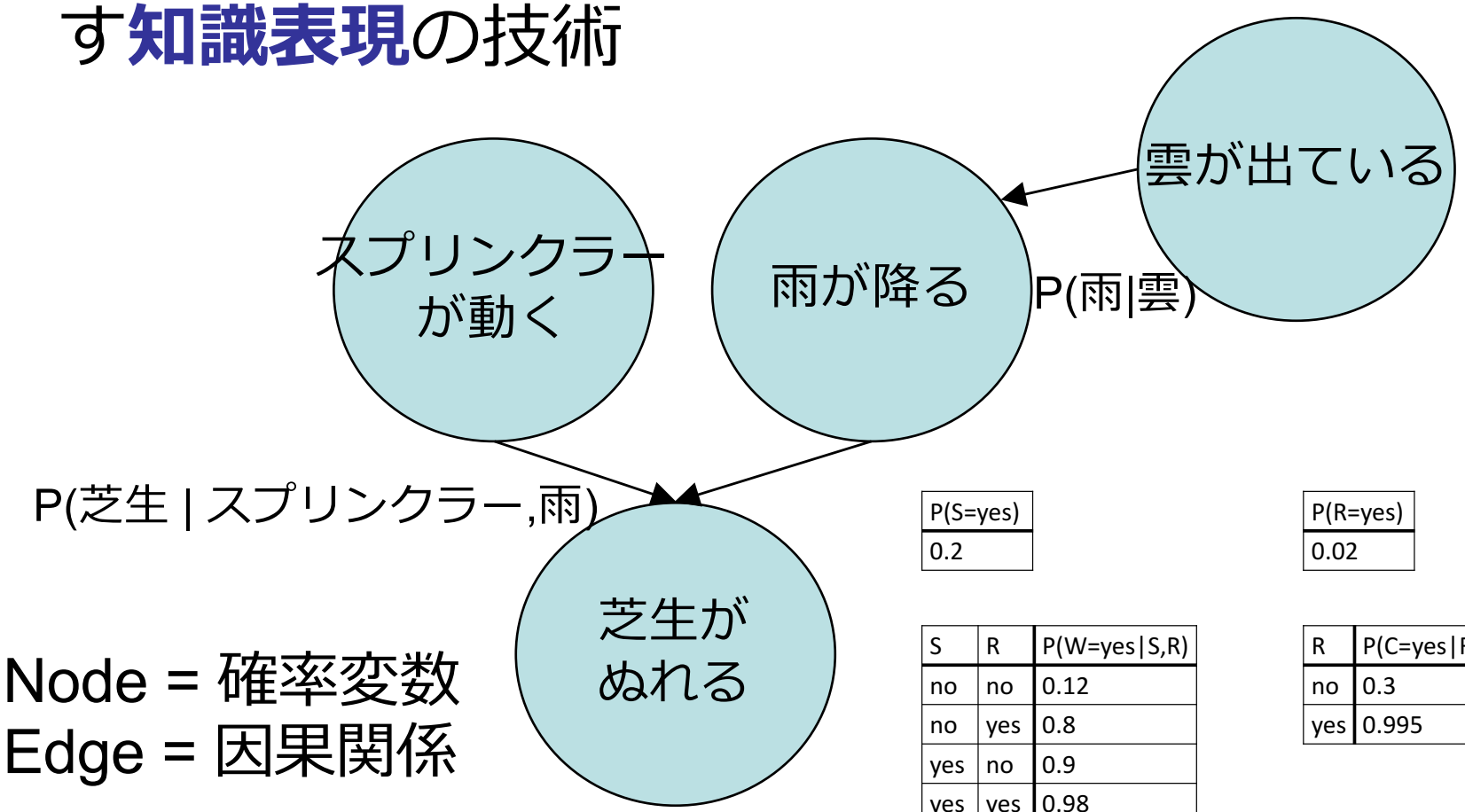
Daniel J. Felleman and David C. Van Essen
Distributed Hierarchical Processing in the
Primate Cerebral Cortex
Cerebral Cortex 1991 1: 1-47





ベイジアンネット[Pearl 1988]とは

- 脳の「直感・連想記憶」と似た働きをする
- 確率変数の間の因果関係をグラフで**効率的**に表す**知識表現**の技術



P(S=yes)
0.2

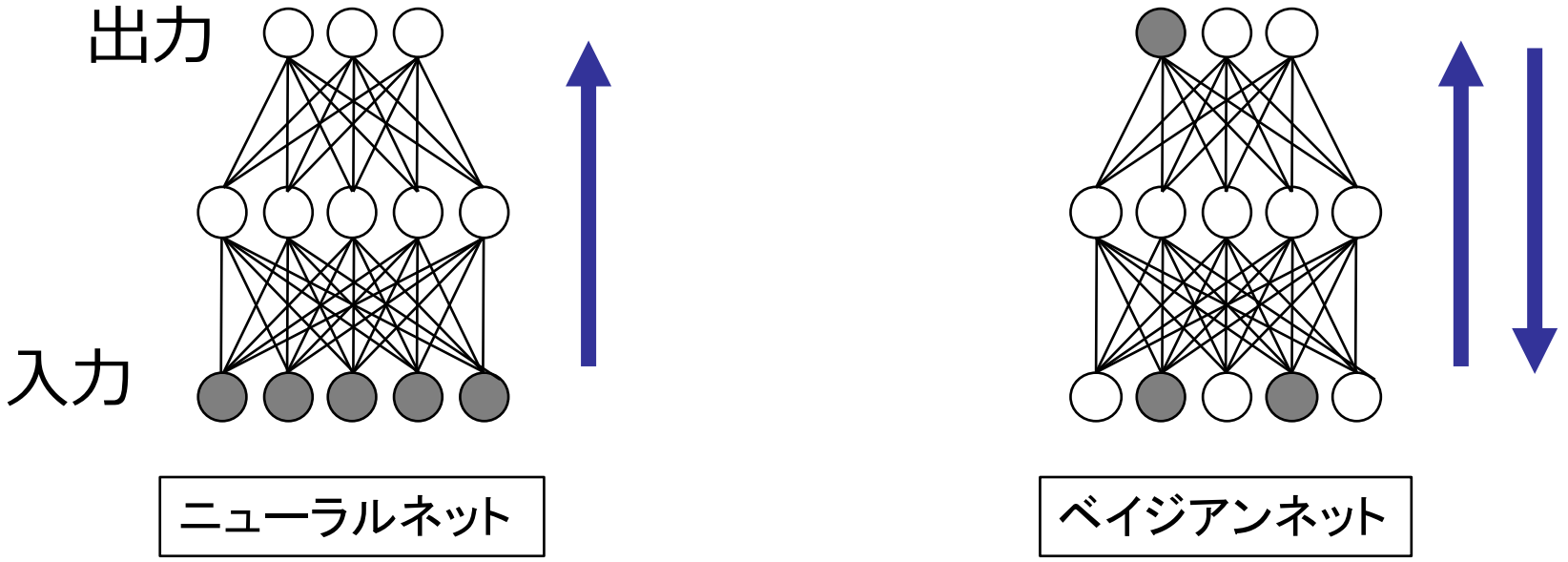
P(R=yes)
0.02

S	R	P(W=yes|S,R)
no	no	0.12
no	yes	0.8
yes	no	0.9
yes	yes	0.98

R	P(C=yes|R)
no	0.3
yes	0.995

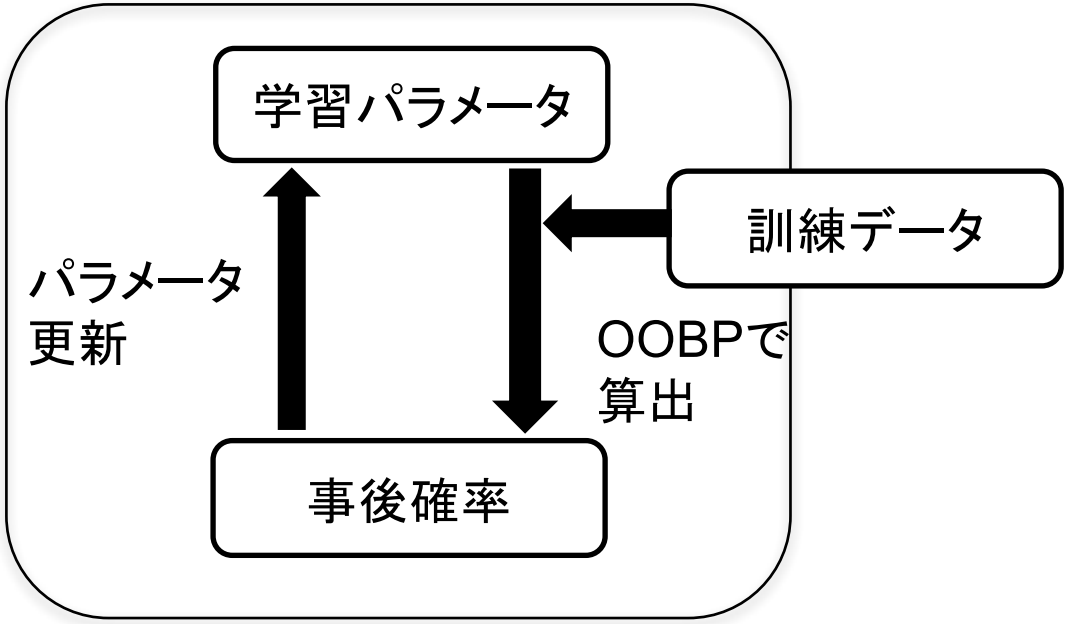
ベイジアンネットとニューラルネット

- 典型的なニューラルネット（多層パーセプトロン）は、情報が入力から出力への１方向
- ベイジアンネットは入力と出力が限定されず、双方向に情報が伝搬
  - 任意のノードが入力・出力ノードとなりうる



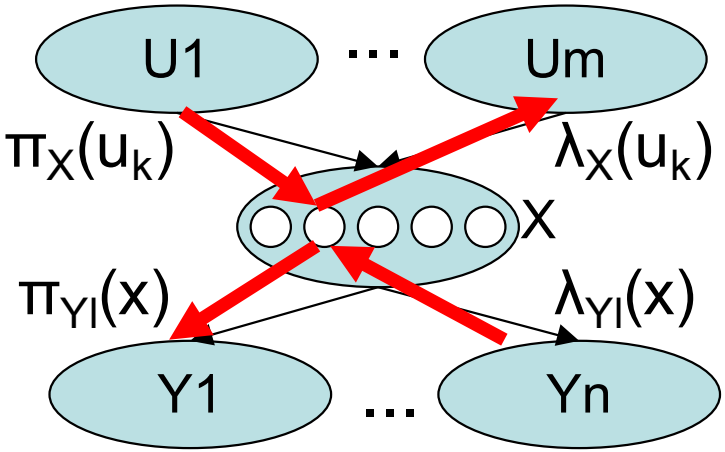
BESOMによる学習の概要

- 訓練データを与えてOOBPで事後確率を算出
- 事後確率を用いて学習パラメータを更新



近似確率伝播アルゴリズムの改良OOBP [ichisugi, takahashi 2015]

- 上から下への情報伝達と、下から上への情報伝達を交互に行う
- 層数に依存するが10回程度の反復で収束



$$\lambda_{Y_l}^{t+1}(x) = \beta_2 \sum_{y_l} \lambda^t(y_l)(\pi^t(y_l) - \kappa_X^t(y_l) + w(y_l, x))$$

$$\lambda^{t+1}(x) = \prod_{l=1}^{n} \lambda_{Y_l}^{t+1}(x)$$

$$\pi_{Y_l}^{t+1}(x) = \beta_1 \rho^{t+1}(x) / \lambda_{Y_l}^{t+1}(x)$$

$$\kappa_{U_k}^{t+1}(x) = \sum_{u_k} w(x, u_k) \pi_X^t(u_k)$$

$$\pi^{t+1}(x) = \sum_{k=1}^{m} \kappa_{U_k}^{t+1}(x)$$

$$\rho^{t+1}(x) = \lambda^{t+1}(x) \pi^{t+1}(x)$$

$$BEL^{t+1}(x) = \alpha \rho^{t+1}(x) \qquad (8)$$

機械学習の並列化手法

- モデル間並列化（Data Parallel）
  - 複数のモデルを用いて並列に学習を実行し、結果となるモデルパラメータを更新

クラスタ分散化

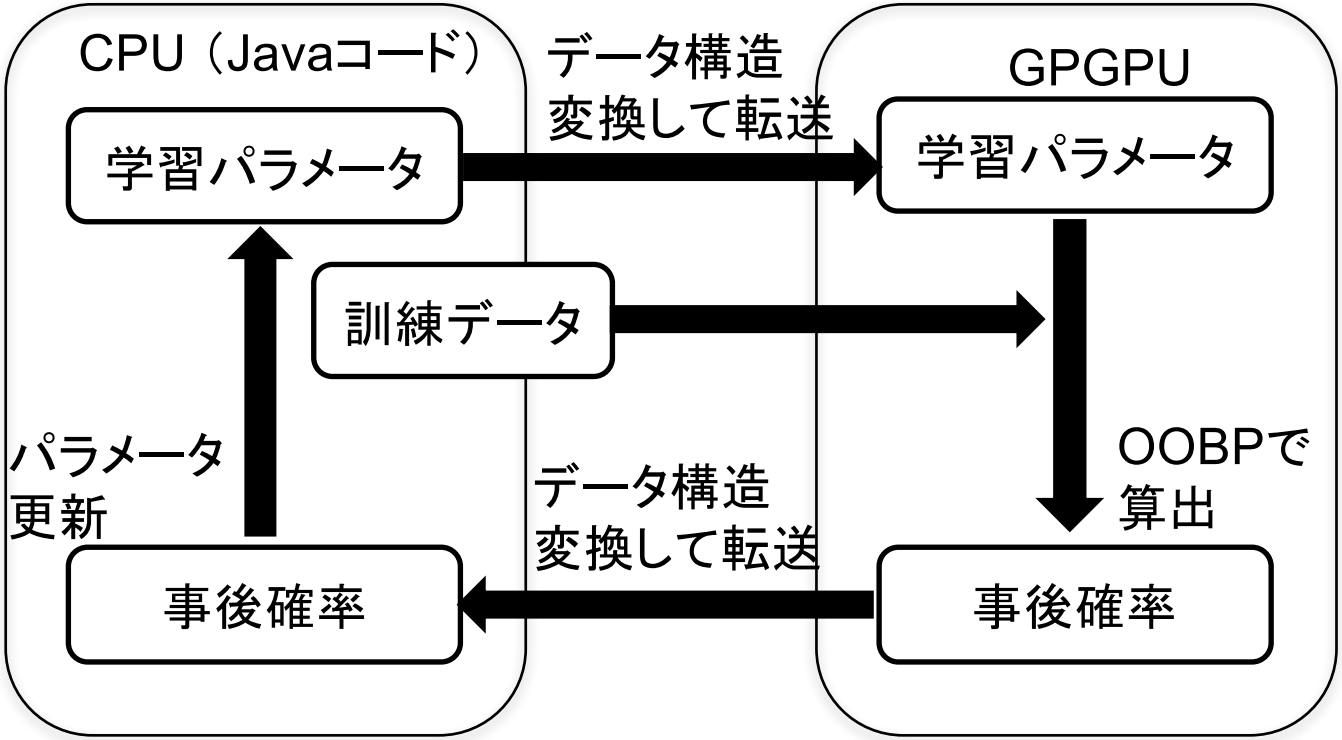
- モデル内並列化（Model Parallel）
  - 一つのモデルの学習の内部を並列化
  - ミニバッチ化による並列化

GPGPU並列化

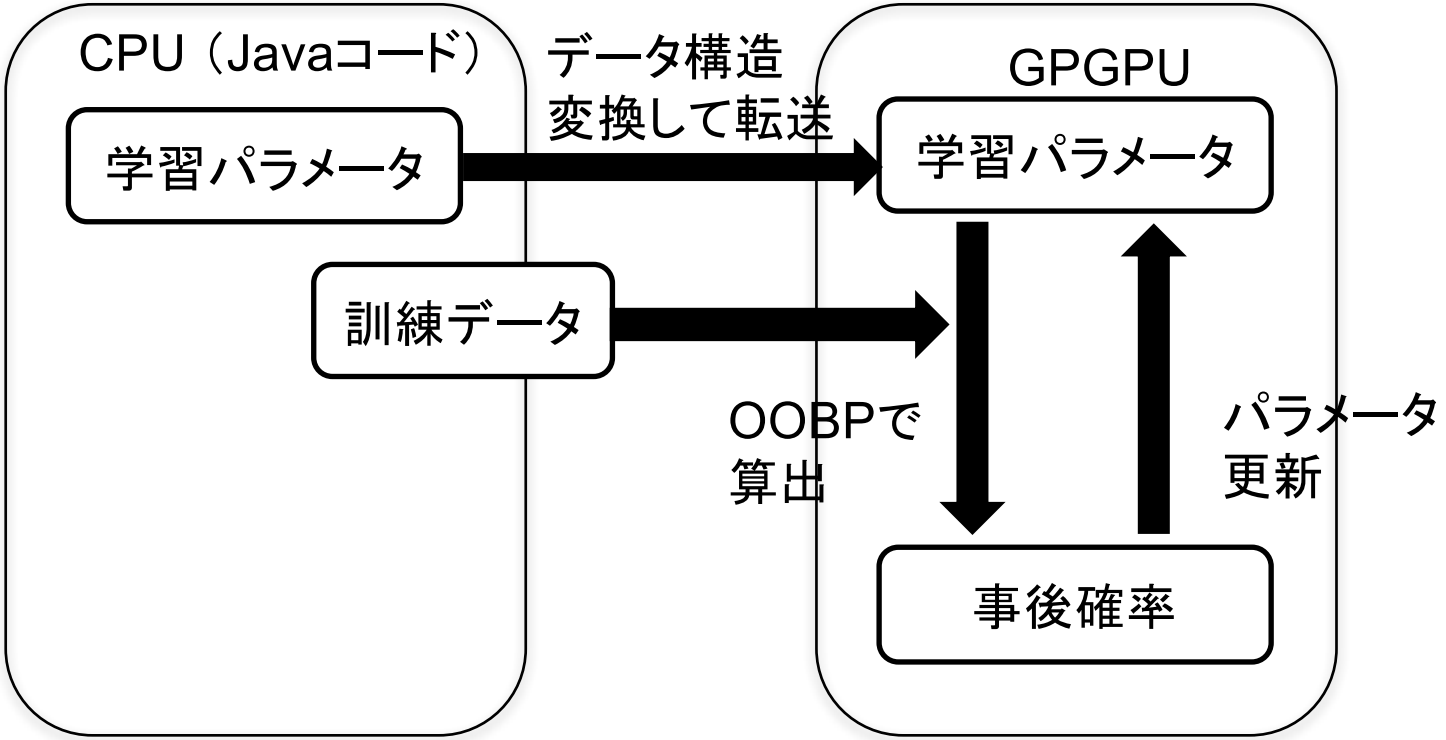
GPGPUの特徴

- 多数のコア
  - 数千以上の並列実行が可能
  - 多数の並列性がないと高速実行できない
  - メモリアクセスレイテンシを隠蔽するにはコア数以上の並列度が必要
- ユニフォームな計算に特化
  - 分岐の選択が異なると効率低下
- CPUとは独立したメモリ空間
  - CPUとのデータ通信がボトルネックになりうる

GPGPUを用いた実装



GPGPUを用いた実装(2)

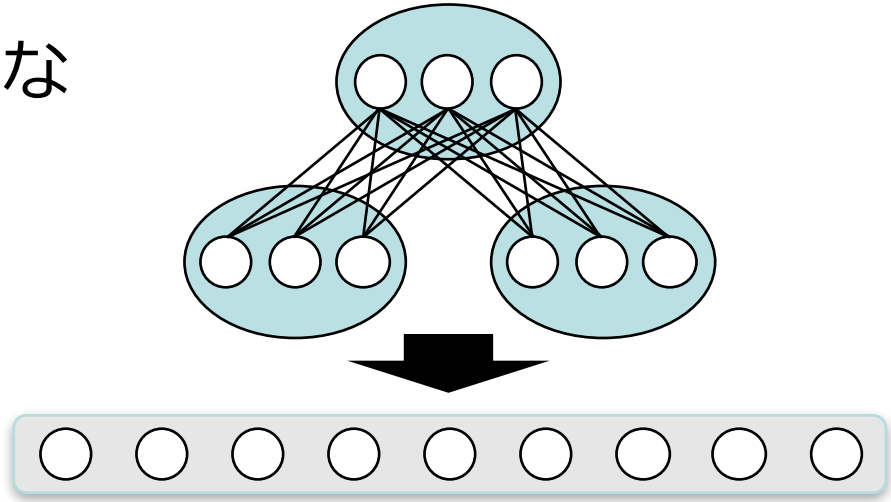


並列化方針

- OOBPのノード情報更新の並列化
  - ノード単位では負荷が不均衡
    - 負荷が上下のノードの数に依存するため
  - ノードをオブジェクトとし相互に参照するデータ構造はGPGPU化に不適
    - GPGPUは別のメモリ空間 → 参照構造を引き写すにはポインタの変換が必要
    - 不均質な計算 → GPGPUコードの並列化効率の低下

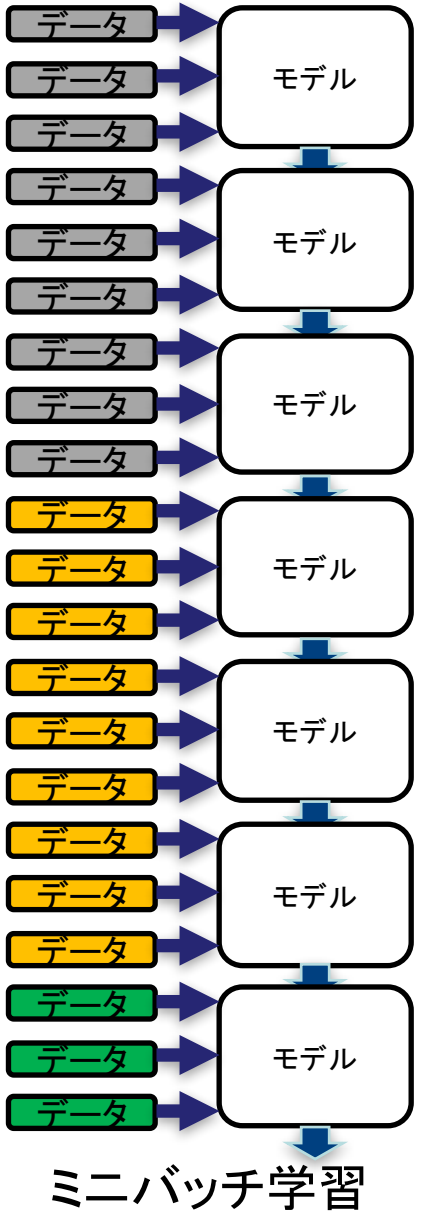
➔ユニットを基盤としたフラットなデータ構造に変更

- 多数のオブジェクトに格納された三次元配列を一本の配列に
- 負荷の均等化
- 計算の均質化

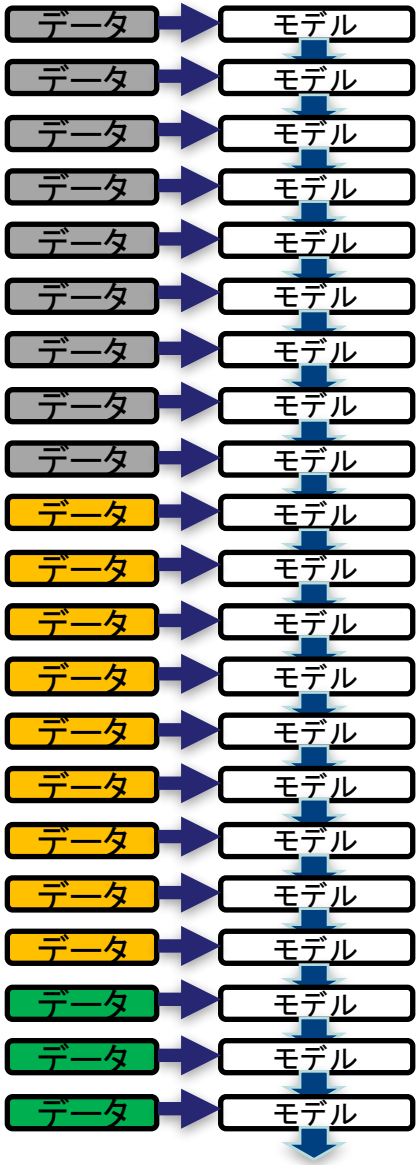


ミニバッチ学習による並列化

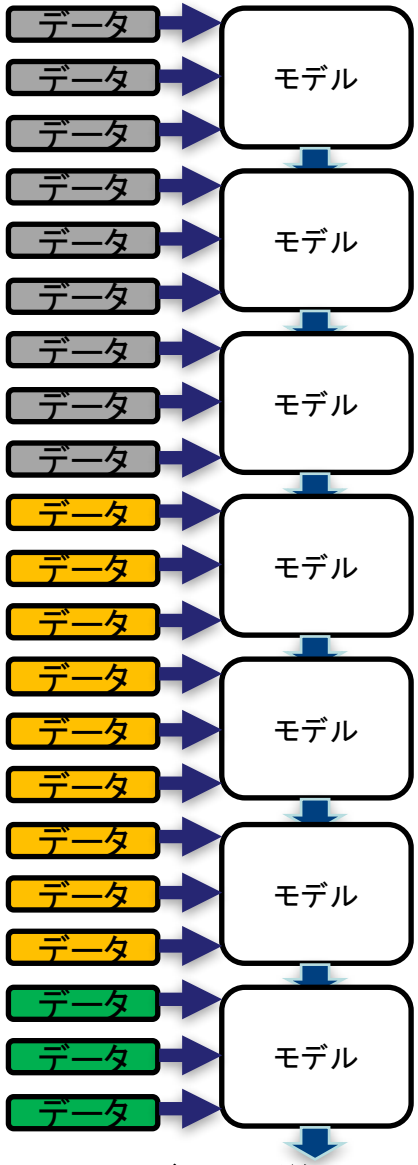
- Javaでの実装は、オンライン学習
  - 個々のデータに対してモデルパラメータを更新
  - 複数のデータに対する評価を並列に実行できない
- ミニバッチ学習に変更
  - 複数のデータを一つのモデルに適用可能 → 並列化が可能



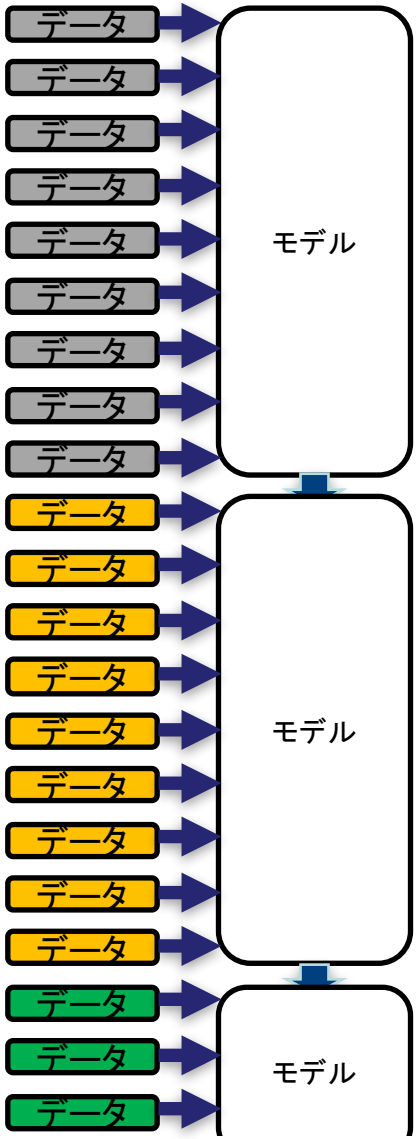
さまざまな学習のモード



オンライン学習



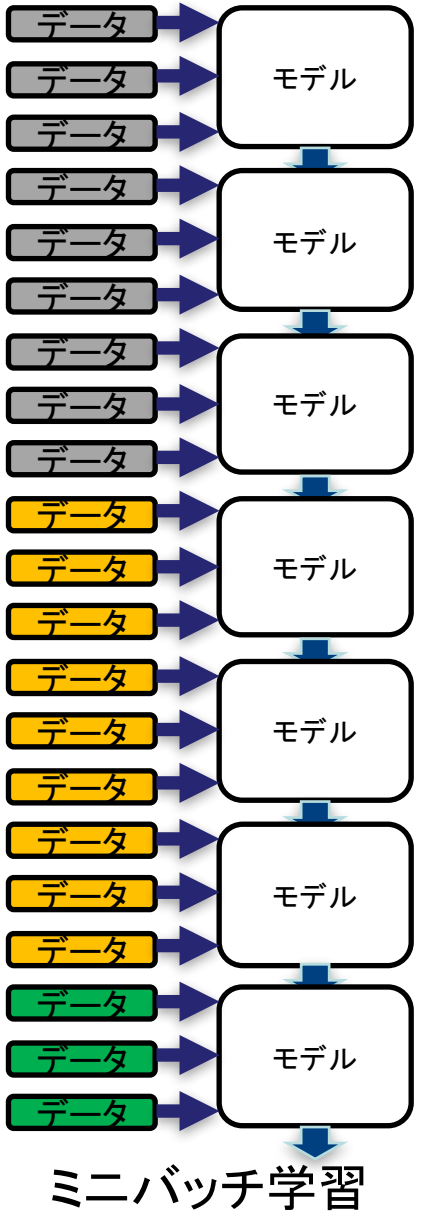
ミニバッチ学習



バッチ学習

ミニバッチ学習による並列化

- Javaでの実装は、オンライン学習
  - 個々のデータに対してモデルパラメータを更新
  - 複数のデータに対する評価を並列に実行できない
- ミニバッチ学習に変更
  - 複数のデータを一つのモデルに適用可能 → 並列化が可能



JCUDA

JavaからCUDAで書かれたカーネルを直接呼び出す機構

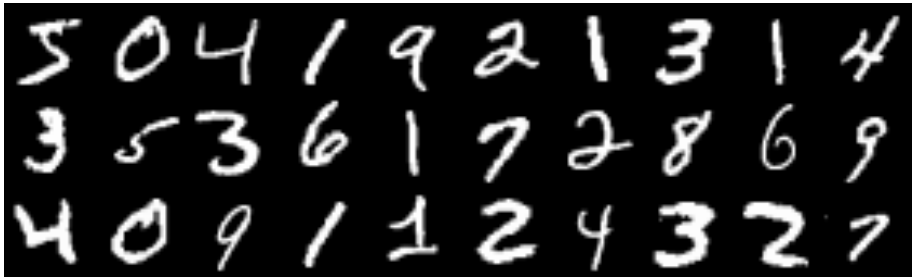
```
// Load the ptx file.
CUmodule module = new CUmodule();
cuModuleLoad(module, "JCudaVectorAddKernel.ptx");

// Obtain a function pointer to the kernel function.
CUfunction function = new CUfunction();
cuModuleGetFunction(function, module, "add");

// Call the kernel function.
cuLaunchKernel(function,
     gridSizeX,  1, 1,      // Grid dimension
     blockSizeX, 1, 1,      // Block dimension
     0, null,               // Shared memory size and stream
     kernelParameters, null // Kernel- and extra parameters
);
```

評価

- 測定項目
  - OOBP実行時間
  - 学習を含めた実行時間
- 対象問題
  - MNIST手書き文字認識
  - 4層ネットワーク、パラメータ数 約20万
- 評価環境
  - NVIDIA GeForce GTX 980
  - Intel Xeon W5590 3.33GHz 4コア x2

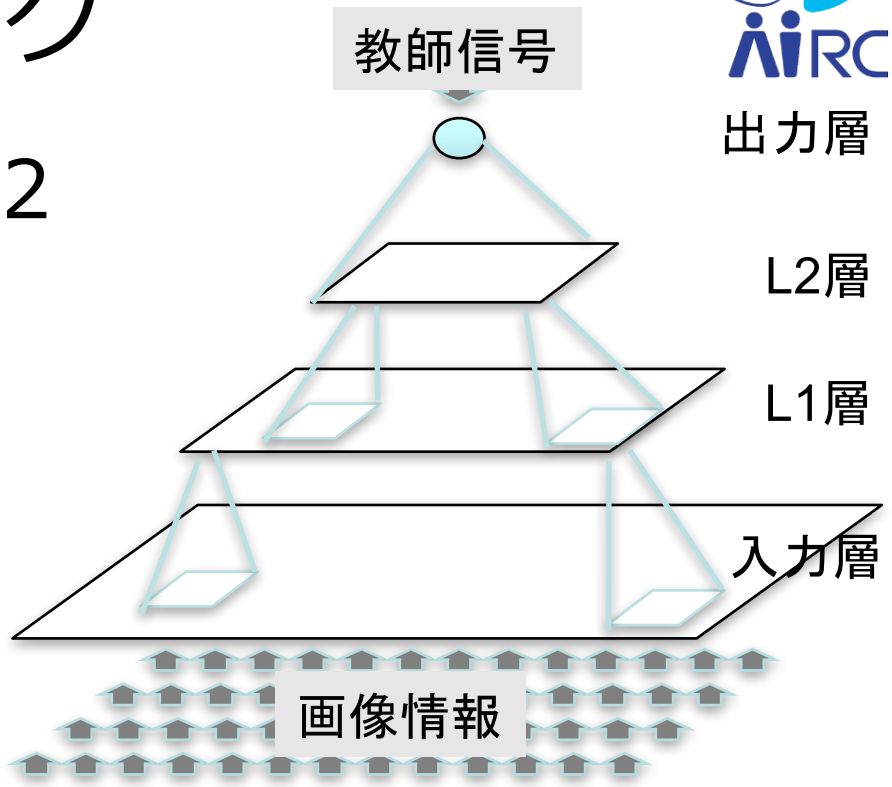
評価問題：MNIST手書き数字認識

- NIST（National Institute of Standards and Technology）によるデータセットの一つで、基本的な認識問題として広く用いられる
- 手書きの数字を識別する
  - 28x28x8bit (256階調)
  - 50000の訓練データ
  - 10000のテストデータ



評価問題：ネットワーク

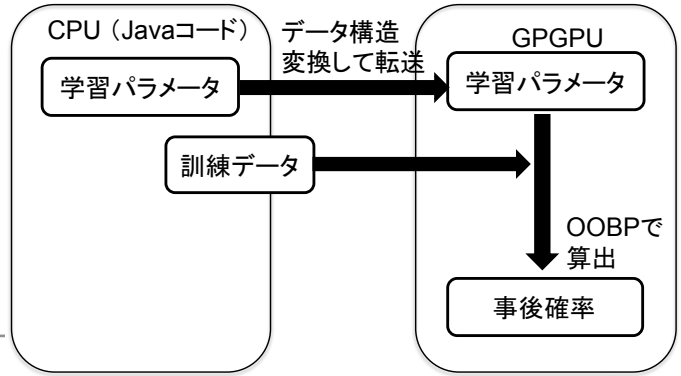
- 入力層、出力層、中間層ｘ２の４層ネットワーク



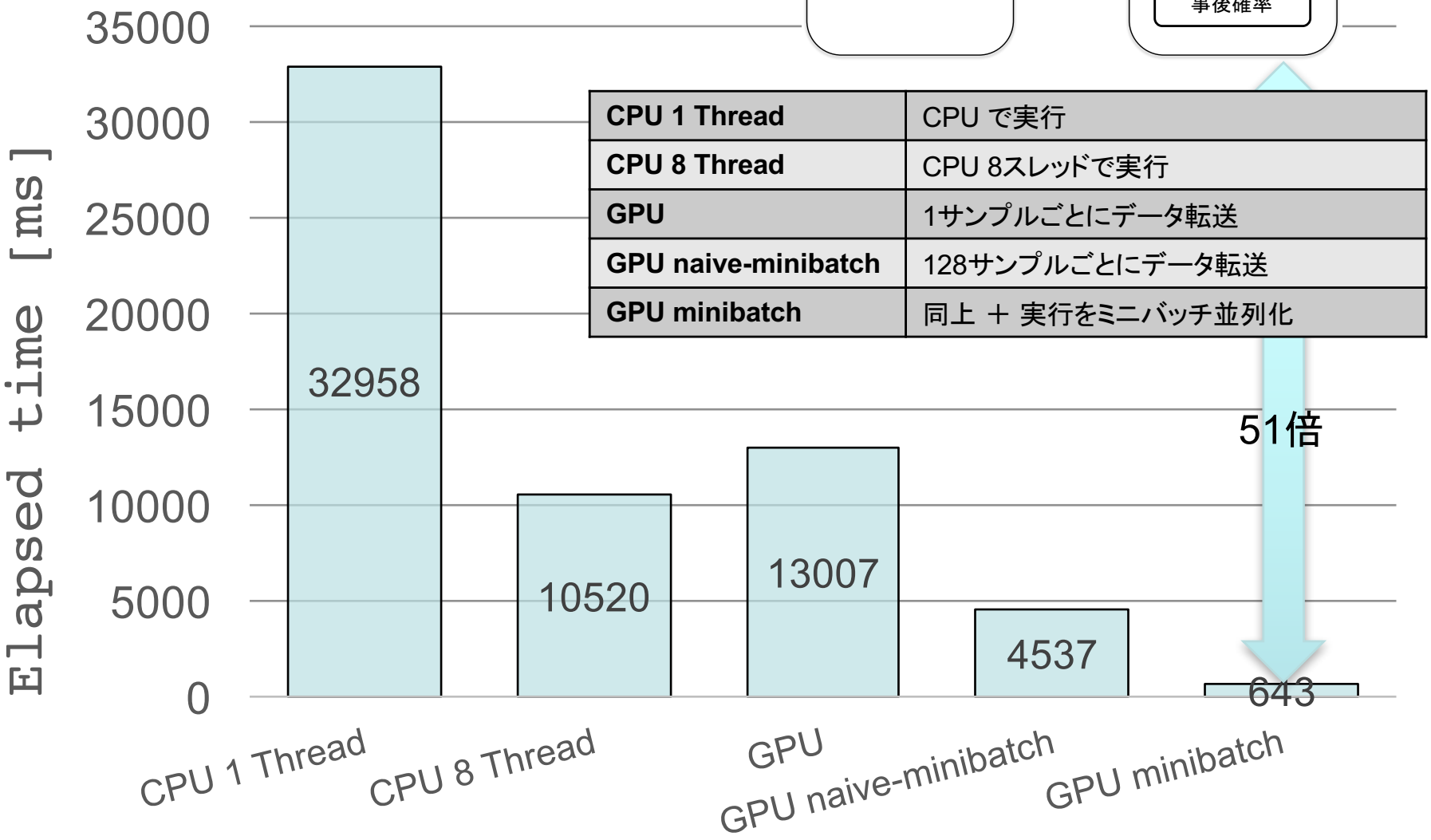
	入力層	L1	L2	出力層
ノード数	784 (28 * 28)	81 (9 * 9)	9 (3 * 3)	1
1ノードあたりのユニット数	2	20	100	11
ユニット総数	1568 (784 * 2)	1620 (81 * 20)	900 (9 * 100)	11 (1 * 11)
接続する親ノードの数	―	16 (4 * 4)	9 (3 * 3)	9 (3 * 3)
パラメータ数	―	51840 (81 * 20 * 2 *16)	162000 (9 * 100 * 20 * 9)	9900 (1 * 11 * 100 * 9)

OOBPの実行時間

1280サンプルを処理、バッチサイズ128

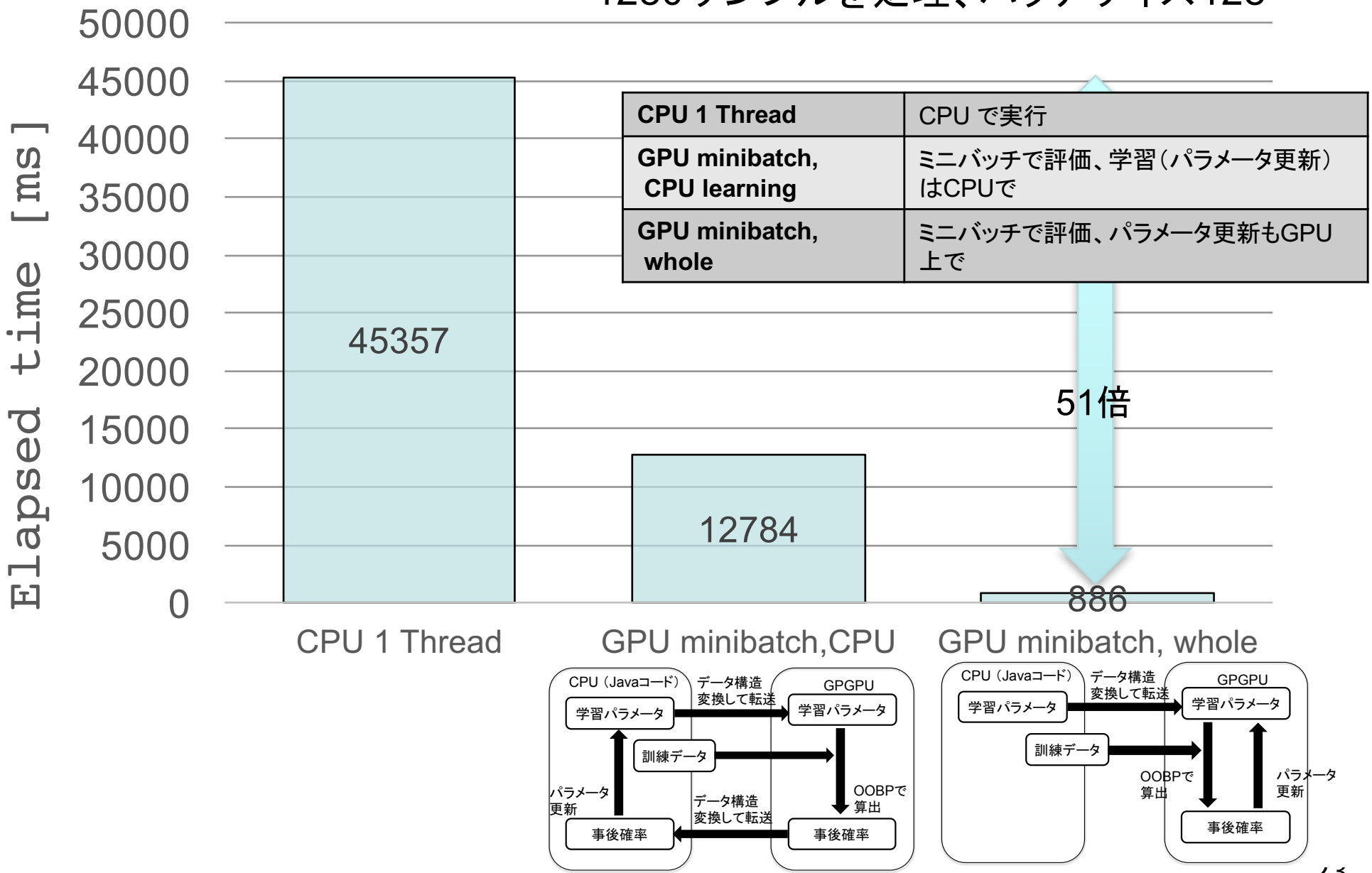


CPU 1 Thread	CPU で実行
CPU 8 Thread	CPU 8スレッドで実行
GPU	1サンプルごとにデータ転送
GPU naive-minibatch	128サンプルごとにデータ転送
GPU minibatch	同上 ＋ 実行をミニバッチ並列化



学習を含めた実行時間

1280サンプルを処理、バッチサイズ128



まとめと今後の課題

- まとめ
  - GPGPUによる並列化
  - ミニバッチ学習を導入
  - 約50倍の高速化
  - DLにおいて標準的な高速化手法がBESOMにおいても有効であることを確認
- 今後の課題
  - 学習率の調整
  - マルチGPGPU対応
  - 分散化との併用