Vol.2020-HPC-173 No.15
2020/3/17

SX-Aurora TSUBASAによるSLIMの高速化

井手口 裕太[1,a)]　大野 善之[1,b)]　石坂 一久[1,c)]

概要：
Top-N 推薦のための高精度かつ高速なアルゴリズムである SLIM のベクトル演算を用いた高速化手法を提案する．Top-N 推薦は，過去の購入履歴などを学習することでユーザーに推薦するアイテムを決定する問題であるが，膨大なデータを利用する学習時間の短縮が求めらている．SLIM はスレッド並列化が考慮されたアルゴリズムであるが，高性能ベクトルコンピュータ SX-Aurora TSUBASA で高速化するには，効率的なベクトル演算手法を開発する必要があった．本稿では，SLIM の主要処理に対するベクトル演算手法を提案し，SX-Aurora を用いた高速化を可能とする．Top-N 推薦でよく利用される MovieLens データセットを用いた評価では，提案する SX-Aurora を用いた SLIM は，2 ソケット Xeon に比べて 3.3 倍の高速であることを確認した．

キーワード：ベクトルプロセッサ，SX-Aurora TSUBASA，Top-N 推薦

1. はじめに

Top-N 推薦は，過去の購入履歴などを学習することで，未購入のアイテムなどからなる推薦候補アイテム集合に対する推薦スコアを計算し，各ユーザに対して，その中から推薦スコアが上位の N 個のアイテムを推薦することである．近年では Top-N 推薦においても，DL（Deep Learning）に関する研究が盛んに行われ，従来の DL 以外の手法（以降，従来推薦手法）との精度の比較が行われている．Top-N を推薦する問題において，Dacrema ら [1] は，いくつかの DL を用いた手法と従来推薦手法を実装し，精度の比較を行った．その結果，多くの問題において従来推薦手法の方が精度が高いことが示された．一方，学習速度の観点では，一般的には大量の演算を必要とする DL ベースの手法に比べて，従来推薦手法のほうが高速であり，従来推薦手法が見直されてきている．本稿では，従来推薦手法の中でも精度と学習速度の両面で優れたアルゴリズムとして知られている SLIM（Sparse LInear Method）[2] に注目する．

Top-N 推薦は，過去の購入履歴などを学習するため，購入アイテム数とユーザ数が増加すると学習時間も増加する．そのため，ベクトルコンピューターなどの高性能コンピューターを用いた高速化が重要である．

[1] NEC データサイエンス研究所
神奈川県川崎市中原区下沼部 1753, 211–8666
a) ideguchi@nec.com
b) ohno.yoshiyuki@nec.com
c) ishizaka@nec.com

本稿では，高性能ベクトルコンピューターである SX-Aurora TSUBASA を用いて精度の高い統計的な手法である SLIM の高速化を目指す．SX-Aurora TSUBASA に搭載されている VE（Vector Engine）は，高い演算性能とメモリ帯域幅を備えるが，VE の性能を引き出すためには，主要処理をベクトル化する必要がある．本稿では，SLIM の学習処理をベクトル化することで SLIM を VE で実装し，実行した際の処理時間を CPU との比較を行った結果について説明する．

2. VE（ベクトルエンジン）

本章では，SX-Aurora TSUBASA に搭載されている VE（ベクトルエンジン）について説明する．VE は，ベクトルプロセッサと高速メモリにより構成されている．ベクトルプロセッサは，複数のデータをレジスタに格納し，同時に計算を行なうことで，複数のデータに対して高速に数値演算を実行できるプロセッサである．そのため，レジスタの数が多く，これらのデータをメモリから多く読み書きするため，メモリバンド幅も広い．ベクトルプロセッサは，一つのベクトル命令で複数のデータを一括処理できるため，大規模・複雑な計算を高速処理することができ，気象，航空宇宙，環境，流体解析，物性計算などに適している．従来，ベクトルプロセッサはスーパーコンピュータ用に開発されていた．近年，発売された SX-Aurora TSUBASA[3] では，タワー型のモデルも発売され，より使用用途の拡大が見込まれる．本稿では，SLIM を VE で実装することで高速化

© 2020 Information Processing Society of Japan

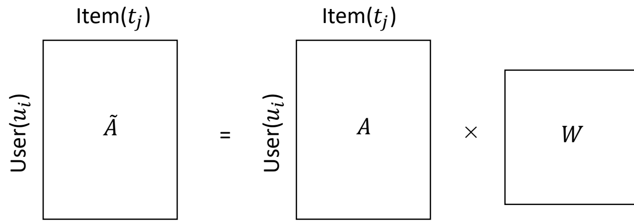

図 1　推薦スコア行列 $\tilde{A} = AW$ の計算

を目指す．この VE は，複数要素を同時に処理（ベクトル演算）することができるコアが複数個あるため，SLIM の処理を複数要素でベクトル化および複数コアで並列化する必要がある．本稿では，SLIM の学習処理のベクトル化および複数コアでの並列化について説明する．

3. SLIM（Sparse LInear Method）[2]

本章では，SLIM の推薦モデルおよび学習方法について説明する．

3.1 SLIM の推薦モデル

あるユーザが各アイテムを購入したかどうかが，0 または 1 で各行に表現されているユーザ・アイテム行列 A を考える．行番号を i，列番号を j とするとき，ユーザ u_i がアイテム t_j を購入していれば，a_{ij} は 1，購入していなければ 0 とする．SLIM による推薦モデルでは，ユーザ u_i へのアイテム t_j の推薦スコア $\tilde{a}_{ij}$ は，疎な係数ベクトル $\mathbf{w}_j$ を用いて，以下の式より計算される．

$$\tilde{a}_{ij} = \mathbf{a}_i^{\mathrm{T}} \mathbf{w}_j \tag{1}$$

$\mathbf{a}_i^T$ は，ユーザ u_i の各アイテムの購入履歴である．したがって，ユーザ・アイテム推薦スコア行列 $\tilde{A}$ は，図 1 に示すように，推薦モデルの重みとして疎な係数行列 W を用いて以下のように表現される．

$$\tilde{A} = AW \tag{2}$$

3.2 重み W の計算

SLIM では，A と AW の各要素が近い値になるように，以下の最適化問題を解くことで重み W の計算を行う．

$$\begin{gathered} \underset{W}{\text{minimize}} \frac{1}{2}||A - AW||_2^2 + \frac{\beta}{2}||W||_2^2 + \lambda||W||_1, \\ \text{diag}(W) = 0 \end{gathered} \tag{3}$$

β および λ は係数である．この問題は W の各列で独立に解くことが可能なので以下のように，$\mathbf{w}_j$ ごとに計算することができるため，並列化が容易である．

$$\begin{gathered} \underset{\mathbf{w}_j}{\text{minimize}} \frac{1}{2}||\mathbf{a}_j - A\mathbf{w}_j||_2^2 + \frac{\beta}{2}||\mathbf{w}_j||_2^2 + \lambda||\mathbf{w}_j||_1, \\ w_{jj} = 0 \end{gathered} \tag{4}$$

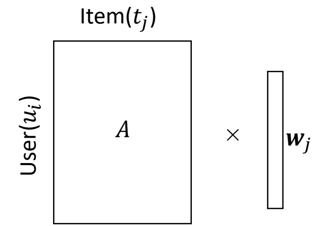

図 2　$A\mathbf{w}_j$ の計算

この最適化問題では，以下のように $\mathbf{w}_j$ を更新することで学習を行う．

$$\mathbf{w}_j \Leftarrow \mathbf{w}_j - \kappa \frac{\partial G(\mathbf{w}_j)}{\partial \mathbf{w}_j} \tag{5}$$

$$\begin{gathered} G(\mathbf{w}_j) = \frac{1}{2}||\mathbf{d}_j||_2^2 + \frac{\beta}{2}||\mathbf{w}_j||_2^2 + \lambda||\mathbf{w}_j||_1, \\ \mathbf{d}_j = \mathbf{a}_j - A\mathbf{w}_j, \end{gathered}$$

κ は係数である．$\mathbf{w}_j$ を更新するために $\partial||\mathbf{d}_j||_2^2/\partial \mathbf{w}_j = 2A\mathbf{d}_j$ を計算する必要があり，$\mathbf{d}_j$ を計算するための $A\mathbf{w}_j$ と $A\mathbf{d}_j$ の計算が学習時間の大部分を占める．

3.3 アイテム間の類似度を用いた重み計算の高速化

SLIM では，重み計算の際の $A\mathbf{w}_j$ の計算時に，A のうち特定の列（アイテム）のみを用いて演算するように $\mathbf{w}_j$ の非零要素の位置を決定することで演算量を削減する．前述したように，$\tilde{\mathbf{a}}_j = A\mathbf{w}_j$ と $\mathbf{a}_j$ の各要素が近い値になるように重みを学習するため，A の各列のうち $\mathbf{a}_j$ に類似している列（$\mathbf{a}_j$ と同じ位置に 1 が入っている列）に対応する重み w_{ij} が大きくなる．したがって，$\mathbf{a}_j$ と類似していない列を無視して計算しても結果への影響は少ない．

そこで，SLIM では，前処理として行列 A の列間（アイテム間）の類似度を計算し，重み $\mathbf{w}_j$ の非零要素の位置を類似度が高い列のみからなるユーザーアイテム行列に対応する位置に設定し，重み $\mathbf{w}_j$ の非零要素の値のみを更新することで学習を行う．

3.4 SLIM の学習

SLIM の学習の流れを図 3 に示す．前節で述べたように，SLIM ではまず前処理として類似度計算を行い，その後に W の更新を複数回行う．本稿では，この繰り返しの終了を判定を行うため，重み更新を行う毎に精度検証を行い，精度がしきい値に達するまで繰り返す．

精度検証用のデータとして，購入履歴から各ユーザごとに購入済アイテム 1 個と未購入アイテム $M-1$ 個を選択することで，1 ユーザあたり M 個の推薦候補アイテムを用意する．なお，精度検証用データに各ユーザごとに購入済アイテム 1 個が含まれているので，学習用データは，精度検証用データに含まれている購入済アイテムを除外した購入履歴を利用する．

© 2020 Information Processing Society of Japan

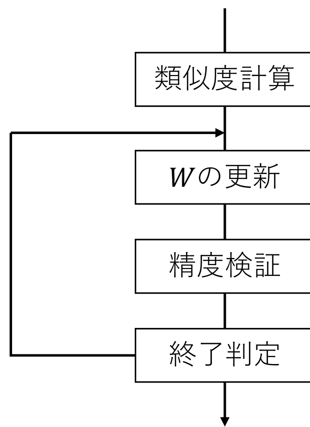

図 3　SLIM の学習の流れ

精度検証では，まず全ユーザーに対して M 個の推薦候補アイテムに対する推薦スコアを以下のように計算する（ここで $\tilde{\mathbf{a}}_i$ および $\mathbf{a}_i$ は M 次元ベクトルである）．

$$\tilde{\mathbf{a}}_i^T = \mathbf{a}_i^T W = (W^T \mathbf{a}_i)^T \qquad (6)$$

精度としては，推薦スコアの高い N 個のアイテム中に選択した購入済みアイテムが含まれるユーザーの割合を用いる．

4.　SLIM の学習の VE による実装

本章では，まず，本稿で利用する疎行列および疎ベクトルの格納方式について説明する．次に，類似しているアイテム F 個を選択するための基準となる類似度の計算時間の高速化について説明し，ベクトル化および並列化について説明する．そして，重み W の値の更新および推薦精度検証の高速化について説明し，ベクトル化および並列化について説明する．

4.1　疎行列および疎ベクトルの格納方式

まず，一般的な疎行列の保存方式である CRS（Compressed Row Storage）と CCS（Compressed Column Storage）および疎ベクトルの保存について説明する．CRS 形式の疎行列は，図 4 に示すように，非零要素の値と列番号を行方向に順々に value 配列と index 配列に格納し，各行の最初の要素の番号を offset 配列に格納する．CCS 形式の疎行列は，図 4 に示すように，CRS 形式と同様に value 配列，index 配列，offset 配列で格納するが，格納の方向が列方向であり，各列の最初の要素の番号を offset 配列に格納する．

本稿では，疎ベクトルを，密形式として格納する方法と疎形式として非零要素のみを格納する方法を考える．密形式では，図 6 に示すように，疎ベクトルの値のすべてを value 配列に格納する．粗形式では，図 7 に示すように，疎ベクトルの非零要素のみを value 配列と index 配列で格納する．

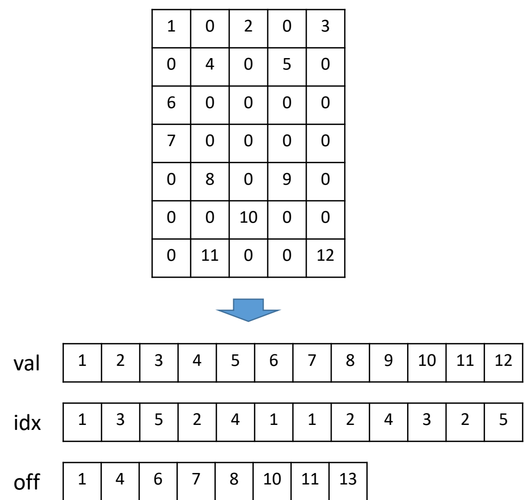

図 4　CRS 形式の疎行列

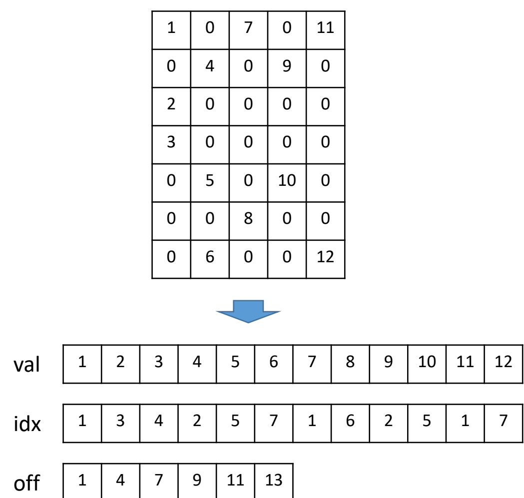

図 5　CCS 形式の疎行列

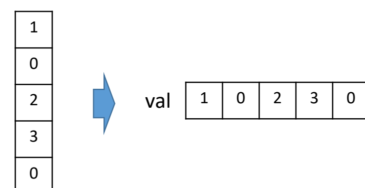

図 6　疎ベクトルの密形式での格納

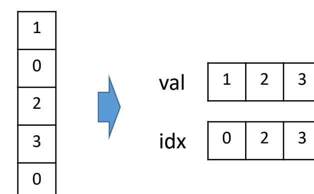

図 7　疎ベクトルの疎形式での格納

4.2　類似度計算の高速化

本稿では，アイテム同士の類似度として，cos 類似度を利用する．アイテム t_j の各ユーザの購入履歴を $\mathbf{a}_j$，アイテム $t_{j'}$ の各ユーザの購入履歴を $\mathbf{a}_{j'}$ とするとき，アイテム t_j とアイテム $t_{j'}$ の cos 類似度は以下のようになる．

© 2020 Information Processing Society of Japan

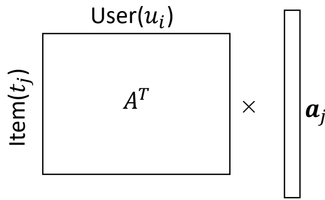

図 8 $A^T\mathbf{a}_j$ の計算

$$\cos\theta_{j'j} = \frac{\mathbf{a}_{j'} \cdot \mathbf{a}_j}{|\mathbf{a}_{j'}||\mathbf{a}_j|} \quad (7)$$

SLIM では，各アイテム t_j に対してすべてのアイテムの類似度を計算し，t_j 以外の類似度の高いアイテムを特徴アイテムとして F 個選択する．この計算では，$\mathbf{a}_{j'} \cdot \mathbf{a}_j$ が計算時間の大部分を占めることになる．類似度計算では，アイテム t_j に対してすべてのアイテムの類似度を計算するため，図 8 に示す $A^T\mathbf{a}_j$ を計算することになる．

本稿では，$\mathbf{a}_j$ の非零要素のうちの一部だけで類似度を計算することで，類似度計算の高速化を図る．実際には，$\mathbf{a}_j$ の非零要素のうちの Q %のデータをランダムに選択し，類似度を計算する．これは，類似度計算に利用される行列 A のユーザデータをサンプリングするのと等価であり，演算量を Q %に減らすことができるので，類似度計算の時間を $100/Q$ 倍に高速化することができる．さらに，このサンプリング方法は，各アイテム t_j ごとに使用するユーザデータを変えることができるので，類似度計算全体として全ユーザデータを利用することも可能である．

VE は，複数要素を同時に処理（ベクトル演算）することができるコアが複数個あるため，ベクトル化および複数コアで並列化する必要がある．類似度計算では，$A^T\mathbf{a}_j$ をすべてのアイテムに対して行うので，各アイテムごとの類似度計算を複数コアで並列化し，$A^T\mathbf{a}_j$ をベクトル化する．$A^T\mathbf{a}_j$ を高速に処理するため，疎行列 A^T および疎ベクトル $\mathbf{a}_j$ が十分に疎であることを仮定し，図 5 に示すように，疎行列は CCS 形式で格納し，疎ベクトルは疎形式で格納する．これにより，CCS 形式の疎行列と疎形式のベクトルで疎行列疎ベクトル積を行うことで密形式の疎ベクトルを出力する．この演算では，図 9 に示すように，疎行列の列方向でベクトル化する．疑似コード 1 に，密形式のベクトルを出力する CCS 形式の疎行列疎ベクトル積の処理の流れを示す．疑似コード 1 の do in pallalel は，ベクトル化を行う for 文を示している．m_{val}，m_{idx}，m_{off} はそれぞれ，疎行列の value 配列，index 配列，offset 配列であり，sv_{val}，sv_{idx}，sv_{size} はそれぞれ，疎ベクトルの value 配列と index 配列と非零要素数であり，o_{val}，o_{size} は，密形式の出力ベクトルの value 配列と全要素数である．CCS 形式の疎行列と疎形式のベクトルで疎行列疎ベクトル積は，疎ベクトルが十分に疎であるときにメモリへのアクセスは複雑であるが無駄なメモリアクセスや演算が少ないため，高速に計算することが可能である．

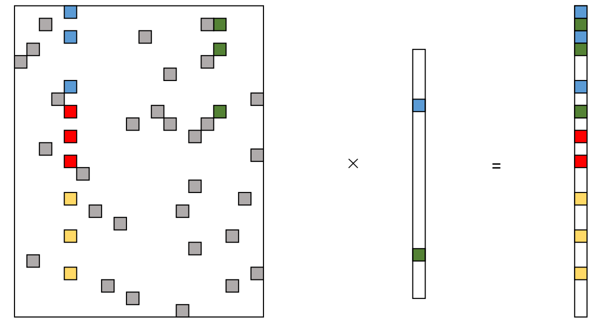

図 9 疎行列疎ベクトル積のベクトル化（ベクトル長 3 の例），疎行列は CCS 形式，処理の順番：青→赤→黄→緑，灰：計算に使われない要素

疑似コード 1 密形式のベクトルを出力する CCS 形式の疎行列疎ベクトル積

```
1: for i = 0 to o_size  do in parallel
2:   o_val[i] ⇐ 0
3: end for
4: for  i = 0 to sv_size  do
5:   c ⇐ sv_idx[i]
6:   for r = m_off[c] to m_off[c + 1]  do in parallel
7:     o_val[m_idx[r]] ⇐ o_val[m_idx[r]] + m_val[r] × sv_val[c]
8:   end for
9: end for
```

疑似コード 2 類似度計算

```
1: for i = 0 to o_size  do in parallel
2:   o_val[i] ⇐ 0
3: end for
4: for  i = 0 to sv_size  do
5:   c ⇐ sv_idx[i]
6:   for r = m_off[c] to m_off[c + 1]  do in parallel
7:     o_val[m_idx[r]] ⇐ o_val[m_idx[r]] + 1
8:   end for
9: end for
```

疎行列 A^T および疎ベクトル $\mathbf{a}_j$ の値は 0 または 1 であり，疎行列は CCS 形式で格納し，疎ベクトルは疎形式で格納しているため，m_{val} および sv_{val} はすべて 1 となる．したがって，類似度計算では，疑似コード 1 を疑似コード 2 に書き換えることができる．

4.3 W の更新の高速化

W の更新では，$A\mathbf{w}_j$ と $A\mathbf{d}_j$ をそれぞれすべての $\mathbf{w}_j$ と $\mathbf{d}_j$ で計算する必要があるため，各 $\mathbf{w}_j$ と各 $\mathbf{d}_j$ をコアで並列化し，$A\mathbf{w}_j$ と $A\mathbf{d}_j$ をベクトル化する．

本稿では，$A\mathbf{w}_j$ に関しては，疎行列および疎ベクトルは十分に疎であることを仮定し，図 5 に示すように，疎行列は CCS 形式で格納し，疎ベクトルは疎形式で格納する．これにより，CCS 形式の疎行列と疎形式のベクトルで疎行列疎ベクトル積を行うことで密形式の疎ベクトルを出力する．この演算では，図 9 および疑似コード 1 に示すよう

© 2020 Information Processing Society of Japan

図 10 $\mathbf{w}_j$ の更新

図 11 推薦候補アイテムの推薦スコアのみの計算

疑似コード 3 疎ベクトルを出力する CRS 形式の疎行列ベクトル積

```
1: so_idx ⇐ set index of output
2: for i = 0 to so_size do in parallel
3:    so_val[i] ⇐ 0
4: end for
5: for i = 0 to so_size do
6:    r ⇐ so_idx[i]
7:    for c = m_off[r] to m_off[r + 1] do in parallel
8:       so_val[i] ⇐ so_val[i] + m_val[c] × v_val[m_idx[c]]
9:    end for
10: end for
```

に，疎行列の列方向でベクトル化する．

そして，$A\mathbf{d}_j$ に関しては，疎行列は図 4 に示すような CRS 形式で格納し，$\mathbf{d}_j$ が比較的密なベクトルになるので，疎ベクトルは，密形式で格納する．$A\mathbf{d}_j$ の計算結果は，$\mathbf{w}_j$ の更新に使われるが，図 10 に示すように，$\mathbf{w}_j$ の更新は非零要素のみを更新するため，$A\mathbf{d}_j$ のすべてを計算する必要はない．したがって，$\mathbf{w}_j$ の非零要素に対応する行のみを A から抽出した疎行列を作成することで計算することで，$\mathbf{w}_j$ の非零要素が全アイテム数に比べて少ないときに無駄な計算を削減できる．しかし，$\mathbf{w}_j$ の非零要素の位置は各 $\mathbf{w}_j$ ごとに異なるため，疎行列の抽出を $\mathbf{w}_j$ 毎に行う必要がある．そこで本稿では，CRS 形式の疎行列と密形式のベクトルで疎行列ベクトル積を部分的に行い，$\mathbf{w}_j$ の非零要素の位置に対応した疎ベクトルを出力することで，疎行列の抽出を行わずに $A\mathbf{d}_j$ の必要な結果を計算する．疑似コード 3 に，疎形式の疎ベクトルを出力する CRS 形式の疎行列疎ベクトル積の処理の流れを示す．疑似コード 3 の do in pallalel は，ベクトル化を行う for 文を示している．m_{val}，m_{idx}，m_{off} はそれぞれ，疎行列の value 配列，index 配列，offset 配列であり，v_{val} は，ベクトルの value 配列であり，so_{val}，so_{idx}，so_{size} はそれぞれ，疎形式の出力ベクトルの value 配列と index 配列と非零要素数である．

4.4 精度検証

精度検証では，精度検証用のデータに対して推薦を行うことで，推薦精度の検証を行う．そのために，全ユーザーに対して M 個の推薦候補アイテムに対する推薦スコアを計算する．推薦スコア計算の計算 $(W^T\mathbf{a}_i)^T$ では，行列は図 4 に示すような CRS 形式で格納し，疎ベクトルは密形式で格納する．この計算は図 11 に示すように，$\tilde{\mathbf{a}}_i^T$ の内の M 個の推薦候補アイテムの推薦スコアのみを計算すればよいので，$A\mathbf{d}_j$ の計算と同様に，CRS 形式の疎行列と密形式のベクトルで疎行列ベクトル積を部分的に行うことで，無駄な計算を削減する．これにより，疑似コード 3 に示すように，推薦候補アイテムの推薦スコアのみの疎ベクトルを出力することで $W^T\mathbf{a}_i^T$ の必要な結果を計算する．

5. 速度計測実験

本章では，SLIM を VE で実装することによる速度性能向上を調べるため，CPU と VE で動作させた際の速度を計測した．

5.1 計測条件

実験では，SX-Aurora TSUBASA[3] 上の CPU(Xeon Gold 6126: 1.3 TFlops 単精度, 0.13 TB/s) 2 ソケット，VE：ベクトルエンジン (Type 10B: 8 Core 4.3 TFlops 単精度, 1.2 TB/s) でそれぞれ SLIM を実装し，処理速度を比較した．SLIM の実装は，統計的機械学習向けフレームワークの Frovedis[4] を用いて実装した．

データセットとしては，MovieLens 1B Synthetic Dataset[5] のデータセットを用いた．このデータセットは，ユーザ数が約 2M ユーザで，アイテム数が約 0.9M アイテムで，非零要素数が約 1G 個のデータセットである．精度検証用のデータは，1 ユーザあたりの推薦候補アイテム数 $M = 1000$ アイテムとした．推薦検証では，Top-N 推薦の $N = 10$ を考え，HR@10 を計算する．HR@10 の計算は，各ユーザの推薦候補アイテムの推薦スコアを計算し，各ユーザの上位 10 個のアイテムの中に購入済アイテムが入っていたユーザの割合を計算する．

SLIM の実行では，特徴アイテム数を $F = 100$ とし，類似度計算に用いるユーザのサンプリングレートを $Q = 0.3$ とした．また，速度計測は図 12 に示すように，W の更新と精度検証を $U = 10$ 回繰り返した際の全体の時間を計測した．

5.2 結果と考察

CPU と VE で SLIM を実行した結果，今回のデータセットにおいては HR@10 は 0.514 となった．HR@10 はデータセットや推薦候補アイテム数 M で変化するが，今回の設定において HR@10 は比較的高い精度の結果となった．図 13 に，CPU と VE でそれぞれで SLIM を実行した際の

© 2020 Information Processing Society of Japan

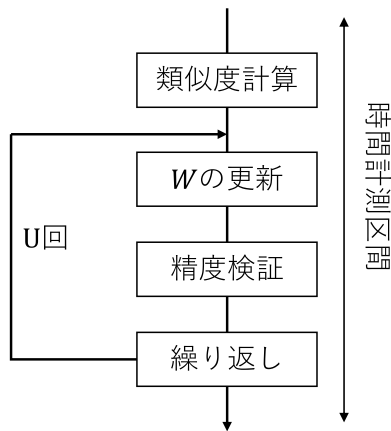

図 12　SLIM の処理の時間計測

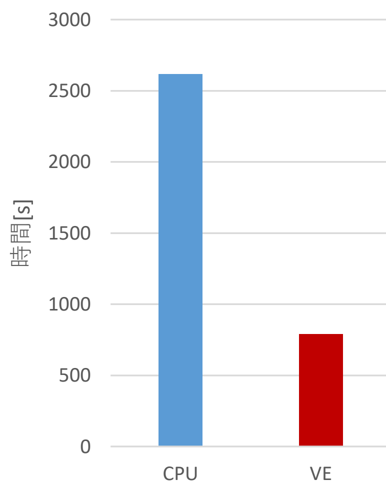

図 13　SLIM の処理時間結果

処理時間を計測した結果を示す．図 13 は，CPU と VE で実行した際の時間を示している．この結果から，CPU 2 ソケットに比べて VE は，処理全体で 3.3 倍程度高速であることが確認できる．

6. おわりに

Top-N 推薦は，過去の購入履歴などを学習することで，推薦候補アイテム集合の推薦スコアを計算し，各ユーザに対して，推薦候補アイテムの中から推薦スコアが上位の N 個のアイテムを推薦することである．この Top-N 推薦を行う精度の高い統計的な手法として SLIM（Sparse LInear Method）[2] がある．本稿では，高性能ベクトルコンピューターである SX-Aurora TSUBASA を用いて精度の高い統計的な手法である SLIM の高速化を目指す．SX-Aurora TSUBASA に搭載されている VE（Vector Engine）は，高い演算性能とメモリ帯域幅を備えるが，VE の性能を引き出すためには，主要処理をベクトル化する必要がある．本稿では，SLIM の学習処理をベクトル化することで SLIM を VE で実装し，実行した際の処理時間を CPU との比較を行った．その結果，CPU 2 ソケットに比べて約 3.3 倍程度高速であることが確認できた．

参考文献

[1] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 101–109, 2019.

[2] Xia Ning and George Karypis. Slim: Sparse linear methods for top-n recommender systems. In *2011 IEEE 11th International Conference on Data Mining*, pages 497–506. IEEE, 2011.

[3] NEC Corporation. Sx-aurora tsubasa. http://jpn.nec.com/hpc/sxauroratsubasa/.

[4] Frovedis: Framework of vectorized and distributed data analytics. https://github.com/frovedis/frovedis.

[5] Movielens 1b synthetic dataset. https://grouplens.org/datasets/movielens/movielens-1b/.

© 2020 Information Processing Society of Japan