DEIM Forum 2012 F6-5

GPUによる不確実データベースからの確率的頻出アイテム集合マイニングの高速化

小澤　佑介†　　天笠　俊之†,††　　北川　博之††

† 筑波大学大学院システム情報工学研究科　〒305–8573 茨城県つくば市天王台 1–1–1
†† 筑波大学システム情報系　〒305–8573 茨城県つくば市天王台 1–1–1
††† 宇宙航空研究開発機構 宇宙科学研究所　〒305–8505 茨城県つくば市千現 2–1–1
E-mail: †kyusuke@kde.cs.tsukuba.ac.jp, ††{amagasa,kitagawa}@cs.tsukuba.ac.jp

あらまし　不確実性を含む大量のデータの処理のために，不確実データベースの研究が広く行なわれている．不確実データベースに対して，頻出アイテム集合マイニングを行なう手法がいくつか提案されているが，処理速度が遅いという問題がある．一方，GPU (Graphics Processing Unit) を用いた GPGPU (General Purpose computation on GPU) という手法が，高性能計算の分野で注目されている．GPGPU は，元々はグラフィック処理のための演算装置である GPU を，その高い並列度をいかして汎用的な計算に利用するものである．本研究では，GPGPU を用いた不確実データベースに対する頻出アイテム集合マイニングの高速化のための手法を提案する．

キーワード　GPGPU，頻出アイテム集合マイニング，不確実データベース

Yusuke KOZAWA†, Toshiyuki AMAGASA††,†††, and Hiroyuki KITAGAWA††

† Graduate School of Systems and Information Engineering, University of Tsukuba
1–1–1 Tennoundai, Tsukuba, Ibaraki 305–8573, Japan
†† Faculty of Engineering, Information and Systems, University of Tsukuba
1–1–1 Tennoundai, Tsukuba, Ibaraki 305–8573, Japan
††† Institute of Space and Astronautical Science, Japan Aerospace Exploration Agency
2–1–1 Sengen, Tsukuba, Ibaraki 305–8505, Japan
E-mail: †kyusuke@kde.cs.tsukuba.ac.jp, ††{amagasa,kitagawa}@cs.tsukuba.ac.jp

1. はじめに

近年の情報通信技術の発達により，取得可能な情報の量は爆発的に増えている．しかし，情報の量が膨大なため，有用な情報を発見するのは難しい．そこで，効率的に有用な情報を抽出するために，データマイニングの技術が重要となる．

相関ルールマイニング [2] は，データマイニングの中でもよく知られた主要な手法の一つである．これは，消費者の購買行動の分析のために，マーケットバスケットデータの分析を目的として開発されたマイニング手法である．アルゴリズムに対する入力は，トランザクションの集合からなり，各トランザクションは消費者が購入したアイテム（商品）からなる．このとき，与えられた入力に対し，互いに相関性の高いアイテム集合を発見することが，相関ルールマイニングの目的である．

相関ルールマイニングは以下の二つのステップに分けられる．

（1） 全ての頻出アイテム集合をマイニングする．

（2） ステップ 1 で求めた頻出アイテム集合から相関ルールを求める．

ここで，頻出アイテム集合とは，あるしきい値以上の頻度でデータベースに現れるアイテムの集合のことである．ステップ 2 はステップ 1 で求めた頻出アイテム集合から相関ルールを発見する処理である．ステップ 1 で求めた頻出アイテム集合のみを用いるため，ステップ 2 はステップ 1 に比べて比較的計算コストは少ない．一方，ステップ 1 はデータベースを繰り返しスキャンする必要があるため，処理時間の大半を占めることになる．そのため，ステップ 1 の効率化が重要となる．

ところで，一般的には，データベース中のデータの存在は確定しており，データは正確であると仮定される．しかし，実世界ではこの仮定が当てはまらない場合も多い．例えば，RFID センサを利用した購買行動の分析の際，センサの誤読をする可能性があるため，不正確なデータを扱う必要がある．近年，これらの大量の不正確なデータを扱うために，不確実データベー

スの研究が進められている．このとき，確率的データを対象に相関ルールマイニングを適用することも可能であり，いくつかの研究が見られる [4,5,13].

不確実データベースからの頻出アイテム集合マイニングの手法は，データの不確実性を考慮しなければならない点において，従来手法 [2,8] とは異なる．現在までに，いくつかの手法 [4,5,13] が提案されているが，不確実性を考慮した処理が新たに必要となるため，処理速度が遅いという問題がある．

一方，GPU (Graphics Processing Unit) を用いた GPGPU (General Purpose computation on GPU) という手法が，高性能計算の分野をはじめ，様々な分野で注目されている [3,6,9,10,12,14]．GPGPU は，元々はグラフィック処理のための演算装置である GPU を，グラフィック処理以外の一般的な計算に利用する手法である．

GPU は数百以上の SIMD (Single Instruction, Multiple Data) コアを持つため，CPU と比較して，極めて並列度の高いデータ処理を行なうことができる．この SIMD コアは，一つの命令を複数のデータに対して，並列に実行していく処理に適したアーキテクチャとなっている．そのため，GPU はその性質上，複雑な条件分岐を含む処理には向いていない．そこで，GPGPU を用いる際には，GPU に適したアルゴリズムの開発が重要な課題の一つとなる．

本研究では，不確実データベースからの頻出アイテム集合マイニングの高速化のために，GPGPU を利用した手法を提案する．提案手法は，Sun ら [13] の提案した pApriori アルゴリズムを元に，GPU 上での並列実行により処理の高速化を図る．さらに，pApriori アルゴリズムとの比較実験により，提案手法の性能を評価した．

本稿の構成は以下の通りである．2 節において，GPGPU について簡単に説明する．3 節では，本研究で扱う問題の定義と，提案手法の元となる pApriori アルゴリズムについて説明する．4 節で提案手法について説明し，5 節で評価実験の結果と考察を述べる．その後，6 節で関連研究を説明し，7 節でまとめと今後の課題について述べる．

2. GPGPU

GPU は，現在多くのコンピュータに備え付けられており，CPU の補助を行なうコプロセッサとして動作する．その内部には，ストリーミングマルチプロセッサ (Streaming Multiprocessor, SM) と呼ばれる SIMD プロセッサが複数存在する．また，一つの SM は，ストリーミングプロセッサ (Streaming Processor, SP) と呼ばれる SIMD コアを多数含んでいる．そのため，多くのコアを持つメニーコアプロセッサとして，GPU が注目されている．

GPGPU のアプリケーションを開発する際に，最も利用されているフレームワークとして，NVIDIA により提供されている CUDA(注1)がある．CUDA では，CUDA スレッドと呼ばれる非常に軽量なスレッドを数千から数万個作成し，並列度の高い処理を実現する [15].

CUDA では，GPU 上の一つの処理に対し，その処理を行なうスレッドの集まりであるグリッドが作成される．また，グリッドは多数のスレッドブロック（またはブロック）を含み，各ブロックは数百スレッドからなる．ブロック中のスレッドは全て同じ SM に割り当てられ，メモリを共有しながら動作する．

また，GPU 上には様々なタイプのメモリが存在する．まず，全てのスレッドがアクセス可能で，大容量・高レイテンシであるグローバルメモリがある．他には，共有メモリという，ブロック中のスレッド間で共有可能な小容量・低レイテンシのメモリがある．各スレッドは非常に高速なレジスタを利用することも可能である．さらに，テクスチャメモリという，6–8KB 程度のキャッシュを持つメモリも存在する．GPU の性能を引き出すためには，これらの種々のメモリを有効活用することが，極めて重要である．

3. 確率的頻出アイテム集合マイニング

本節では，確率的頻出アイテム集合の定義とマイニングの手法について述べる．まず，従来のデータベースに対する問題の定義を 3.1 節で述べ，不確実データベースに対する定義を 3.2 節で述べる．3.3 節では Sun ら [13] の提案した pApriori アルゴリズムについて述べる．

3.1 頻出アイテム集合

アイテム全体の集合を I とする．アイテム $i \in I$ の集合をアイテム集合 X といい，ID とアイテム集合のペアをトランザクション T という．また，トランザクションの集合をトランザクションデータベース $\mathcal{T}$ という．サイズ n のトランザクションデータベース $\mathcal{T}$ とアイテム集合 X が与えられたとき，$\mathcal{T}$ における X の支持度を $sup(X)$ と表す．ここで，アイテム集合 X の支持度とは，X を含むトランザクションの数を指す．ユーザが与えた最小支持度 `minsup` 以上の支持度を持つアイテム集合を頻出アイテム集合という．

従来の頻出アイテム集合マイニングの問題は，以下のように定義できる．トランザクションデータベース $\mathcal{T}$ と最小支持度 `minsup` が与えられたとき，全ての頻出アイテム集合を抽出する．

3.2 確率的頻出アイテム集合

トランザクションデータベース $\mathcal{T}$ の各トランザクションに存在確率を与えたものを，不確実トランザクションデータベース $\mathcal{U}$ という．存在確率は，トランザクションがデータベース中に存在する確率を表す．図 1 に不確実トランザクションデータベースの例を示す．これはある店の客の購買行動を表す．すなわち，game と music が同時に購入される確率は 0.9，music と video が同時に購入される確率は 0.7，などである．

不確実トランザクションデータベースでは各トランザクションが存在確率を持ち，データベース中の存在が確率的に表されるため，アイテム集合の支持度は確率変数となる．アイテム集合 X の支持度 $sup(X)$ の確率質量関数を支持度確率質量関数（Support Probability Mass Function, SPMF）と呼び，$f_X(k)$ と表す．ここで，k は $sup(X)$ が取り得る値である．すなわち，$k \in [0, |\mathcal{U}|]$

(注1)：`http://www.nvidia.com/object/cuda_home_new.html`

ID	アイテム集合	存在確率
T_1	{game, music}	0.9
T_2	{music, video}	0.7
T_3	{game}	0.8
T_4	{game, music, video}	0.5

図 1　不確実トランザクションデータベースの例

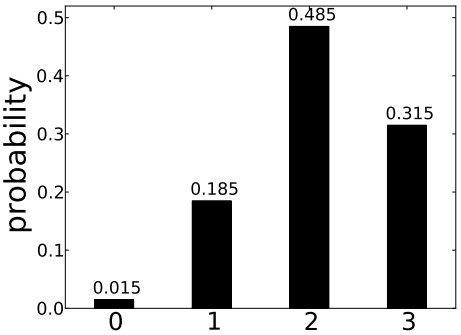

図 2　アイテム集合 {music} の SPMF $f_{\{\text{music}\}}$

となる．また，$f_X(k)$ は $sup(X)$ が k となる確率を表す．図 2 は図 1 の不確実トランザクションデータベースにおける，アイテム集合 {music} の SPMF $f_{\{\text{music}\}}$ である．例えば，$f_{\{\text{music}\}}(0)$ は全てのトランザクションに {music} が現れない確率を表すので，$f_{\{\text{music}\}}(0) = (1-0.9)\cdot(1-0.7)\cdot(1-0.5) = 0.015$ となる．

$$P(sup(X) \geqq \texttt{minsup}) = \sum_{k=\texttt{minsup}}^{|\mathcal{U}|} f_X(k) \geqq \texttt{minprob} \quad (1)$$

を満たすアイテム集合 X を確率的頻出アイテム集合という．ここで，minprob はユーザに与えられる確率のしきい値である．例えば，$\texttt{minsup} = 2$, $\texttt{minprob} = 0.5$ としたとき，図 1 の不確実トランザクションデータベースにおいて，

$$\begin{aligned} P(sup(\{\text{music}\}) \geqq \texttt{minsup}) &= f_{\{\text{music}\}}(2) + f_{\{\text{music}\}}(3) \\ &= 0.485 + 0.315 \\ &= 0.8 \geqq \texttt{minprob} \end{aligned}$$

となるため，アイテム集合 {music} は確率的頻出アイテム集合となる．

本研究で扱う問題は，以下のように定義できる．

［問題 ］（確率的頻出アイテム集合マイニング） 不確定トランザクションデータベース $\mathcal{U}$ と，しきい値 $minsup$, $minprob$ が与えられたとき，$\mathcal{U}$ から全ての確率的頻出アイテム集合を抽出する．

3.3　pApriori アルゴリズム

pApriori アルゴリズム [13] は，Agrawal ら [2] によって提案された Apriori アルゴリズムを不確実データベースに適用したものである．pApriori アルゴリズムでは，まず，要素数 1 のアイテム集合が確率的頻出アイテム集合であるかを，SPMF を計算することによって判定する．次に，$k = 2$ として処理を行っていく．要素数 $k-1$ の確率的頻出アイテム集合から，要素数 k の候補確率的頻出アイテム集合を生成し，これらの候補の中から要素数 k の確率的頻出アイテム集合を生成する．そして k を 1 増やす．これらの処理を，確率的頻出アイテム集合が見つからなくなるまで続ける．最終的に，不確実トランザクションデータベースから得られる全ての確率的頻出アイテム集合を結果として返す．

3.3.1　候補確率的頻出アイテム集合の生成

候補確率的頻出アイテム集合の生成は二つのフェイズに分けられる．一つ目は結合フェイズ，二つ目は枝刈りフェイズである．

結合フェイズでは，要素数 k の確率的頻出アイテム集合から，二つの要素数 k の確率的頻出アイテム集合 X, Y が結合可能であるかを判定する．ここで二つのアイテム集合 X, Y が結合可能とは，$X.item_i$ を X 中の i 番目のアイテムを表すとすると，$X.item_1 = Y.item_1 \wedge \cdots \wedge X.item_{k-1} = Y.item_{k-1} \wedge X.item_k < Y.item_k$ が成り立つことをいう．結合可能ならば，X と Y から，要素数 $k+1$ の候補確率的頻出アイテム集合を作り，その候補を，要素数 $k+1$ の候補確率的頻出アイテム集合の集合 $\mathcal{C}_{k+1}$ に格納する．

枝刈りフェイズでは，以下の補題を利用して枝刈りを行なう [13].

［補題 1］（逆単調性）

アイテム集合 X が確率的頻出アイテム集合ならば，X の部分集合も確率的頻出アイテム集合である．

この補題の対偶から，アイテム集合 X の部分集合が確率的頻出アイテム集合でなければ，X も確率的頻出アイテム集合で無いことが分かる．すなわち，結合フェイズで作った各候補確率的頻出アイテム集合について，その要素数 k の部分集合が確率的頻出アイテム集合であるかを確認し，そうでなければ $\mathcal{C}_{k+1}$ からその候補を削除する．

3.3.2　確率的頻出アイテム集合の抽出

アイテム集合 X が確率的頻出アイテム集合であるかは，X の SPMF を計算し，式 1 を調べなければならない．そのため，SPMF を効率的に計算することが重要となる．Sun らは動的計画法による方法と，分割統治法による方法の二種類の計算法を提案している．ここではより計算量の少なく，GPU 上での処理に適している分割統治法による方法のみを説明する．

このアルゴリズムは，入力として不確実トランザクションデータベース $\mathcal{U}$ とアイテム集合 X を受け取り，アイテム集合 X の SPMF f_X を返す．

$\mathcal{U}$ が一つのトランザクションのみを含む場合は，SPMF を直接計算する．$\mathcal{U}$ がトランザクションを二つ以上含んでいる場合は $\mathcal{U}$ を二つに水平分割し，それぞれのデータベースに対して再帰的に SPMF を計算していき，計算された二つの SPMF は畳み込みにより一つにする．畳み込みをそのまま計算すると $O(n^2)$ の計算量となるが，高速フーリエ変換を利用することで $O(n \log n)$ に抑えることができる．

3.3.3　枝　狩　り

SPMF の計算は計算量が多いため，実際に計算する前に確率的頻出アイテム集合でないものを判定したい．そのために以下

$$(1)\ \begin{bmatrix}0 & 1 & 3\\ 1 & 2 & 3\\ 0 & 3 & 3\\ 0 & 1 & 2\end{bmatrix} \qquad (2)\ \begin{bmatrix}0.9\\ 0.7\\ 0.8\\ 0.5\end{bmatrix}$$

図 3　GPU 上の不確実トランザクションデータベースの例

の二つの指標を用いる．不確実トランザクションデータベース $\mathcal{U}$ 中の，アイテム集合 X を含むトランザクションの数を表す $cnt(X)$ と，$sup(X)$ の期待値を表す $esup(X)$ である．

このとき，以下の二つの補題が成り立つ [13].

［補題 2］ $cnt(X) < \texttt{minsup}$ ならば，アイテム集合 X は確率的頻出アイテム集合ではない．

［補題 3］ $\mu = esup(X)$, $\sigma = (\texttt{minsup} - \mu - 1) \ / \ \mu$ とする．このとき，

- $\sigma \geqq 2e - 1$ かつ $2^{-\sigma\mu} < \texttt{minprob}$
- $0 < \sigma < 2e - 1$ かつ $\exp\left(\frac{-\sigma^2\mu}{4}\right) < \texttt{minprob}$

のどちらかが成り立つならば，アイテム集合 X は確率的頻出アイテム集合ではない．

4. 提 案 手 法

本節では，GPGPU を用いた確率的頻出アイテム集合マイニングの手法について説明する．まず，4.1 節で概要を述べる．その後，4.2 節，4.3 節でアルゴリズムの詳細について述べる．

4.1 手 法 概 要

pApriori アルゴリズムを元に，GPU 上での並列実行により処理の高速化を図る．各アイテムは 0 から $|I| - 1$ までの整数によって表されると仮定する．ここで，I はアイテムの集合を表す．

GPU 上では，不確実トランザクションデータベースを，

（1） 各トランザクションのアイテム集合を格納する配列

（2） 各トランザクションの存在確率を格納する配列

の二つの配列によって表す．一つめの配列には，疎行列のデータ格納方式であり，GPU 上での処理に適している ELL 形式 [3] を用いる．ELL 形式は一つの行列を表すために，各要素の値を格納する配列と，各行において要素が存在する列のインデックスを格納する配列の二つの配列を利用する．しかし，不確実トランザクションデータベースを考える場合，各要素の値は 1 か 0 のみとなるため，後者の列のインデックスの配列のみで十分となる．

例えば，図 1 の不確実トランザクションデータベースは，図 3 のように表される．ここでは，game, music, video をそれぞれ 0, 1, 2 としている．各行の長さは，最も大きいアイテム集合の要素数となるため，この例では 3 となる．この長さに満たない行では，$|I|$ を，すなわち 3 をパディングする．

また，アイテム集合は整数の配列を用いて表す．要素数 k のアイテム集合は要素数 k の配列に格納する．候補確率的頻出アイテム集合の集合や確率的頻出アイテム集合の集合などの，要素数 k のアイテム集合の集合は，以下のように格納する．0 から $k-1$ までの要素で 1 番目のアイテム集合を表し，k から $2 \cdot k - 1$ までの要素で 2 番目のアイテム集合を表す．すなわち，i 番目のアイテム集合は $(i-1) \cdot k$ から $i \cdot k - 1$ までの要素で表す．

4.2 節では，候補確率的頻出アイテム集合の生成のアルゴリズムについて説明する．4.3 節では，確率的頻出アイテム集合の抽出のアルゴリズムについて説明する．

4.2 GPU を用いた候補確率的頻出アイテム集合の生成

GPU 上の候補確率的頻出アイテム集合の生成のアルゴリズムについて説明する．基本的な構造は，pApriori アルゴリズムのものと同様である．

あらかじめ，要素数 $k \cdot \binom{|\mathcal{C}_{k-1}|}{2}$ の配列 $\mathcal{C}_k$ を用意しておく．要素数 $k-1$ の確率的頻出アイテム集合の集合 $\mathcal{L}_{k-1}$ 内の全てのペアについて，それらが結合可能かどうかを調べ，結合可能ならば i, j の値に応じた位置に，結合した候補確率的頻出アイテム集合を格納する．結合可能でないならば，候補確率的頻出アイテム集合でないことを表すため，配列中の該当する箇所に -1 を代入する．枝刈りフェイズで枝刈りの対象となった場合は，先程と同様に，候補でないことを表すため，-1 を代入する．

生成され得る候補確率的頻出アイテム集合すべてを格納できる配列を用意しておくことで，メモリの使用量は大きくなるが，候補確率的頻出アイテム集合の生成を並列に行なうことが可能となる．

4.3 GPU を用いた確率的頻出アイテム集合の抽出

GPU を用いた確率的頻出アイテム集合の抽出は，

（1） 各トランザクションに対する候補の包含判定

（2） 枝刈り

（3） SPMF の計算

の 3 ステップからなる．以下の節で順に説明していく．

4.3.1 候補の包含判定

まず，各候補確率的頻出アイテム集合 X に対し，不確実トランザクションデータベース $\mathcal{U}$ の各トランザクションが X を含んでいるかどうかを判定する必要がある．ここでは，この処理を，不確実トランザクションデータベース $\mathcal{U}$ に対する候補 X の包含判定，または候補 X の包含判定，と呼ぶ．

前準備として，各候補の包含判定結果を格納する配列を用意する必要がある．しかし，各トランザクションに対する包含判定結果は 1 か 0 のみであるため，単純に一つの要素を整数値とすると，無駄が多くなってしまう．また，候補数が多くなると，この配列は巨大になるため，そのままでは GPU 上に乗らなくなる可能性がある．そこで，各要素を 4 バイトの整数値としてビット列を用いて格納することで圧縮を行なう．これにより，必要な要素数は $\lceil |\mathcal{U}| \ / \ 32 \rceil \cdot |\mathcal{C}_k|$ となる．その他の前準備として，候補確率的頻出アイテム集合をテクスチャメモリ上に置き，テクスチャキャッシュを有効活用できるようにしておく．

GPU の各ブロックが 32 個（4 バイト分）のトランザクションを担当し，ブロック中の各スレッドが各候補に対する処理を行なっていく．すなわち，各スレッドは，32 個のトランザクションに対して候補が含まれているかを判定していく．各ブロックが必要とするトランザクションは，処理の前に共有メモ

リに転送しておく．

各スレッドは，担当する候補のアイテムがトランザクションに含まれているかを，線形探索によって判定していく．線形探索を用いることで，一つのブロック中のスレッドが，共有メモリの同じデータを同時にアクセスする．この時，このメモリアクセスはブロードキャストされ，一回分のメモリアクセス時間で済むため，非常に高速である．候補中の全てのアイテムがトランザクションに含まれていれば，そのトランザクションに対応するビットを 1 にセットし，そうでなければ 0 とする．

4.3.2 枝 狩 り

先ほど計算した，候補の包含判定結果を格納する配列と存在確率の配列を用いて，枝刈りに必要な値 cnt と $esup$ を計算していく．

$cnt(X)$ は，不確実トランザクションデータベース中の，アイテム集合 X を含むトランザクションの数を示している．つまり，候補の包含判定結果の配列の総和を取ることで，cnt の値が求められる．この処理の GPU 上での効率的な実装には，parallel reduction [17] を用いる．

また，$esup(X)$ の値は，アイテム集合 X を含むトランザクションの存在確率の総和となる．よって，先ほどと同様に，parallel reduction を用いて，候補を含むトランザクションの存在確率の総和を取ることで，$esup$ の値が求められる．

これらの値と補題 2, 3 を用いて，枝刈り可能かどうかを判定する．枝刈り可能ならば次の候補に進み，そうでなければ SPMF の計算を行なう．

4.3.3 支持度確率質量関数の計算

pApriori アルゴリズムの中で，最も計算量が大きいのが SPMF の計算である．このため，この計算を並列化して効率的に実行することが重要となる．

はじめに，SPMF を格納するための，要素数 $4 \cdot 2^{\lceil \log_2 |\mathcal{U}| \rceil}$ の配列 f_X を用意する．そして，GPU の各スレッドごとに一つのトランザクションに対する SPMF を求め，これらを f_X に格納する．f_X の要素数を $4 \cdot 2^{\lceil \log_2 |\mathcal{U}| \rceil}$ とするのは，以下のような理由からである．

（1） 畳み込みをしていき SPMF を一つにするには，配列中の SPMF の数が 2 のべき乗でなければならない．

（2） 各 SPMF のサイズを 2 のべき乗とすることで，FFT の高速化を図る [16].

各 SPMF のサイズを 2 のべき乗として扱うために，ゼロパディング（空き要素にゼロを埋める）をする．次に，配列中の隣り合う二つの SPMF に対して畳み込みを行なう．畳み込みに必要な FFT は，今回開発に利用した CUDA の FFT ライブラリである CUFFT を用いる．SPMF を一つの配列に同じサイズで格納することで，CUFFT の機能であるバッチ処理を利用することができ，全ての SPMF に対して高速に FFT を計算できる．また，f_X の要素数を固定することで，FFT を実行する際に必要となる $plan$ を再利用可能となり，$plan$ 作成時のオーバヘッドが削減される．この二つの処理を，SPMF が一つになるまで繰り返す．求める SPMF は，$f_X[0]$ から $f_X[|\mathcal{U}|]$ までとなる．

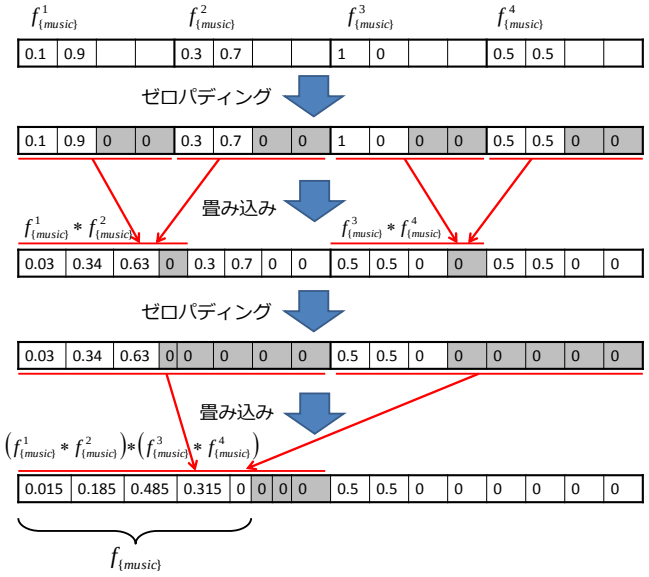

図 4 $f_{\{\text{music}\}}$ の計算過程

図 4 は，図 1 の不確実トランザクションデータベースに対する $f_{\{\text{music}\}}$ の計算過程を示したものである．$f^1_{\{music\}}$, $f^2_{\{music\}}$, ... はそれぞれ $T_1, T_2, \ldots$ に対する {music} の SPMF を表す．また，灰色になっている箇所はゼロパディング処理によってパディングされた要素を示している．

5. 評 価 実 験

本実験では，CPU 上で実装した pApriori アルゴリズムと GPU 上で実装した提案手法の性能の比較を行なう．5.1 節で実験環境について説明し，5.2 節で実験結果と考察を述べる．

5.1 実 験 環 境

実験で使用した CPU は Intel(R) Xeon(R) CPU (2.40GHz), GPU は Tesla C2050 である．Tesla C2050 は 14 個の SM を持ち，各 SM は 32 個の SP (1.15GHz) を持つ．

三つのデータセットに対して実験を行った．各データセットの特徴を表 1 に示す．

一つめは，FIMI Repository(注2)にある Accidents という実データである．トランザクションの平均要素数は 34，アイテム数は 468，データセットのサイズは約 34 万となっている．トランザクションの存在確率は平均 0.5，分散 0.02 の正規分布により与える．明記していない場合，`minsup` はデータセットのサイズの 35%とする．

二つめは，IBM data generator(注3)によって生成された T40I10D100K という合成データである．トランザクションの平均要素数は 40，アイテム数は 942，データセットのサイズは 10 万となっている．トランザクションの存在確率は $(0, 1]$ の間の一様分布により与える．明記していない場合，`minsup` はデータセットのサイズの 1.5%とする．

三つめは，先ほどと同様に IBM data generator によって生

(注2)：`http://fimi.cs.helsinki.fi/`

(注3)：`http://miles.cnuce.cnr.it/~palmeri/datam/DCI/datasets.php`

表 1　実験に用いたデータセット

データセット	アイテム数	トランザクションの平均要素数	トランザクション数
Accidents	468	33.8	340,183
T40I10D100K	942	39.6	100,000
T25I10D500K	7558	25.0	499,960

表 2　SPMF の計算時間

データセット	要素数	CPU (ms)	GPU (ms)	Speedup
Accidents	1	1990	27.5	72.4
	5	1341	27.5	48.8
T40I10D100K	1	22.7	6.27	3.62
	2	9.15	6.27	1.46
T25I10D500K	1	21.7	27.6	0.79
	5	15.9	27.6	0.58

成された，T25I10D500K という合成データである．トランザクションの平均要素数は 25，アイテム数は 7558，データセットのサイズは約 50 万となっている．　トランザクションの存在確率は $(0, 1]$ の間の一様分布により与える．`minsup` はデータセットのサイズの 1.5%とする．明記していない場合，`minsup` はデータセットのサイズの 0.7%とする．

三つのデータセットに対して `minprob` は 0.5 とする．

5.2 実験結果

5.2.1 実験 1：支持度確率質量関数の計算

各データセットに対して，ある要素数の確率的頻出アイテム集合の SPMF の計算時間を測定し，その平均値を計算時間とする．表 2 に実験結果を示す．

GPU 上の SPMF の計算は，CPU 上のものと比較して，データセットとして Accidents を用いた場合，49 倍から 72 倍程度の速度となり，T40I10D100K を用いた場合，1.5 倍から 3.6 倍程度の速度となった．一方，T25I10D500K を用いた場合，0.58 倍から 0.79 倍の速度と，GPU 上のものが遅くなっている．このような結果になった理由として以下のようなものが考えられる．

pApriori アルゴリズムでは，アイテム集合 X を含むトランザクションのみを用いて，SPMF の計算を行なう．すなわち，$cnt(X)$ 個のトランザクションを利用すればよく，SPMF の計算コストは $O(cnt(X)\log^2(cnt(X)))$ となる [13]．しかし，4.3.3 節で述べたように，提案手法では全ての SPMF の計算に対し $2^{\lceil \log_2 |\mathcal{U}| \rceil}$ 個のトランザクションを利用している．そのため，cnt の値が小さい場合，無駄な計算を大量に行ってしまう．

Accidents では，確率的頻出アイテム集合の cnt の値が $200,000$ から $340,000$ とデータセットのトランザクション数に近い大きな値となる．そのため，CPU と GPU で畳み込みの回数にあまり差がなくなり，GPU 上での並列化が効果を発揮する．

T40I10D100K と T25I10D500K では，確率的頻出アイテム集合の cnt の値が最大でも 15,000 程度と，それほど大きな値にならない．そのため，CPU 上では畳み込みの回数はそれほど多くならず，CPU と GPU であまり違いが見られなくなる．

5.2.2 実験 2：確率的頻出アイテム集合マイニング

各データセットに対して，`minsup` を変化させた時の実行時間の測定を行った．実験結果を図 5，6，7 に示す．

図 5 は Accidents を用いた場合の実験結果である．Accidents を用いた場合，5.2.1 節で示した通り，提案手法では SPMF の計算時間が大幅に短縮される．そのため，CPU と比較すると，最大 79 倍の高速化に成功している．

図 6, 7 はそれぞれ T40I10D100K，T25I10D500K を用いた場合の実験結果である．T40I10D100K を用いた場合，1.6 倍から 7.0 倍となり，T25I10D500K を用いた場合，1.01 倍から 1.48 倍となっている．Accidents を用いた場合と異なり，GPU 上の提案手法はそれほど大きくは高速化に成功していない．これは以下のような理由のためと考えられる．

pApriori アルゴリズムでは，候補の包含判定の高速化のために，トライ木という木構造を用いている．一方，提案手法では，このようなデータ構造を利用していないため，この処理がそれほど速くない．また，T40I10D100K を用いた場合，候補確率的頻出アイテム集合の数はきわめて多くなるため，包含判定の必要回数も増加する．さらに，5.2.1 節で示した通り，SPMF の計算時間に大きな違いが見られない．そのため，GPU 上の提案手法がそれほど高速化できていないという結果になった．

以上より，cnt の値が大きく，SPMF の計算時間が支配的となるデータセットを用いた場合には，提案手法が非常に有効であるが，そうでないデータセットの場合には，それほど高速化できていないと言える．

6. 関連研究

不確実データベースは，近年データベース分野において注目を集め，広く研究が行われている．研究対象は，データモデリングや問い合わせ処理，データマイニングなど多岐に渡っている [1, 4, 5, 13]．不確実データベースに関する研究については Aggarwal らによるサーベイ [1] に詳しい．不確実データベースの重要な研究対象の一つに，不確実データベースからの相関ルールマイニングがある．

相関ルールマイニングは，Agrawal ら [2] によって提案された．同時に，彼らは Apriori アルゴリズムを提案している．Han ら [8] は，頻出アイテム集合マイニングのための，Apriori アルゴリズムよりも高速な FP-growth アルゴリズムを提案した．一方，Kuok ら [11] はファジィ集合に対する相関ルールマイニングの研究を行なっている．

不確実データベースからの相関ルールマイニングの研究には，Chui ら [5] のものがある．彼らは，Apriori アルゴリズムを，不確実データベースに対して拡張した，U-Apriori アルゴリズムを提案した．U-Apriori アルゴリズムでは，アイテム集

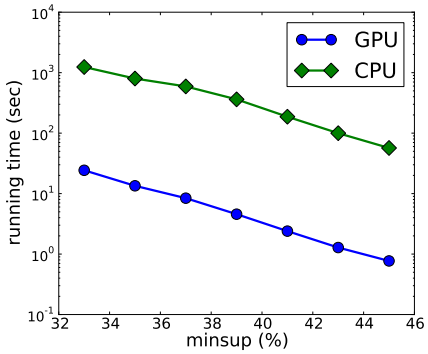

図 5　Accidents に対する実験結果

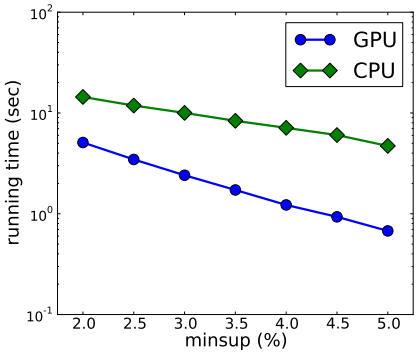

図 6　T40I10D100K に対する実験結果

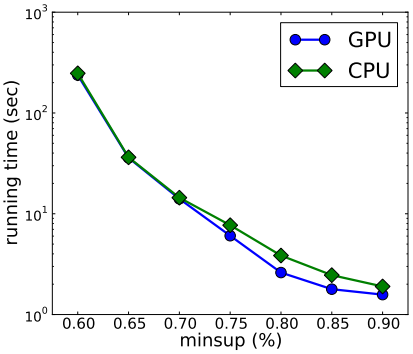

図 7　T25I10D500K に対する実験結果

合の支持度の期待値を用いて，不確実性を扱っている．後に，Bernecker ら [4] は，支持度の期待値を用いると，重要なアイテム集合を見逃す可能性があることを指摘した．彼らは，支持度の期待値の代わりに，アイテム集合が頻出となる確率を定義して用いている．Chui らと Bernecker らは，トランザクションデータベース中の各アイテムに確率が付随しているモデルを扱っている．一方，Sun ら [13] は，各トランザクションが確率を持つモデルを対象とし，動的計画法と分割統治法を利用した二つのアルゴリズムを提案した．

また，高速化のための手法として，GPU を用いた GPGPU が，高性能計算の分野だけでなく他の様々な分野でも注目されている．例えば，Govindaraju ら [7] は，GPU をコプロセッサとして利用し，巨大なデータベースを高速にソートする，GPUTeraSort という手法を提案している．また，He ら [10] は，GPGPU を用いた関係結合のアルゴリズムを考案した．その後，He らは，様々な関係問合せを，CPU と GPU のハイブリッドな手法によって高速化する研究を行なっている [9]．Fang ら [6] は，GPGPU を利用した頻出アイテム集合マイニングの手法を提案した．この研究は，不確実性を含まないデータを対象としている点が，本研究と異なる．他にも，多数の研究が行われており [3, 12, 14]，今後も様々な発展をしていくことが予想される．

7.　まとめと今後の課題

本研究では，Sun らによって提案された pApriori アルゴリズムを元に，GPGPU を用いた確率的頻出アイテム集合マイニングの手法を提案した．そして，pApriori アルゴリズムとの比較実験により，支持度確率質量関数の計算時間が支配的となるデータセットを用いた場合には，提案手法が最大 86 倍の速度となることを示した．しかし，それでないデータセットを用いた場合には，最大でも 7.4 倍程しか高速化できていないということが分かった．

今後の課題として，複数 GPU を用いるマルチ GPU への対応が考えられる．また，GPU だけでなく，CPU も有効活用する CPU と GPU のハイブリッドな手法の提案がある．そのためには，コストモデルを設計し，常に最適なプロセッサを選択できるようにする必要がある．

謝辞　本研究の一部は，科学研究費補助金基盤研究 (A)(#21240005)，科学研究費補助金若手研究 (B)(#23700102) による．

文　献

[1] Charu C. Aggarwal and Philip S. Yu. A Survey of Uncertain Data Algorithms and Applications. *IEEE Transactions on Knowledge and Data Engineering*, 21:609–623, 2009.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules. In *Proceedings of the 20th Interantional Conference on Very Large Data Bases*, pp. 487–499, 1994.

[3] Nathan Bell and Michael Garland. Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pp. 1–11, 2009.

[4] Thomas Bernecker, Hans-Peter Kriegel, Matthias Renz, Florian Verhein, and Andreas Zuefle. Probabilistic Frequent Itemset Mining in Uncertain Databases. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge Discovery and Data mining*, pp. 119–128, 2009.

[5] Chun-Kit Chui, Ben Kao, and Edward Hung. Mining Frequent Itemsets from Uncertain Data. In *Proceedings of the 11th Pacific-Asia conference on Advances in Knowledge Discovery and Data mining*, pp. 47–58, 2007.

[6] Wenbin Fang, Mian Lu, Xiangye Xiao, Bingsheng He, and Qiong Luo. Frequent Itemset Mining on Graphics Processors. In *Proceedings of the Fifth International Workshop on Data Management on New Hardware*, pp. 34–42, 2009.

[7] Naga K. Govindaraju, Jim Gray, Ritesh Kumar, and Dinesh Manocha. GPUTeraSort: High Performance Graphics Co-processor Sorting for Large Database Management. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of Data*, pp. 325–336, 2006.

[8] Jiawei Han, Jian Pei, and Yiwen Yin. Mining Frequent Patterns without Candidate Generation. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of Data*, pp. 1–12, 2000.

[9] Bingsheng He, Mian Lu, Ke Yang, Rui Fang, Naga K. Govindaraju, Qiong Luo, and Pedro V. Sander. Relational Query Coprocessing on Graphics Processors. *ACM Transactions on Database Systems*, 34:21:1–21:39, 2009.

[10] Bingsheng He, Ke Yang, Mian Lu, Naga K. Govindaraju, Qiong Luo, and Pedro V. Sander. Relational Joins on Graphics Processors. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of Data*, pp. 511–524, 2008.

[11] Chan Man Kuok, Ada Fu, and Man Hon Wong. Mining Fuzzy Association Rules in Databases. *SIGMOD Record*,

27:41–46, 1998.

[12] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E. Lefohn, and Timothy J. Purcell. A Survey of General-Purpose Computation on Graphics Hardware. In *Computer Graphics Forum*, volume 26, pp. 80–113, 2007.

[13] Liwen Sun, Reynold Cheng, David W. Cheung, and Jiefeng Cheng. Mining Uncertain Data with Probabilistic Guarantees. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge Discovery and Data mining*, pp. 273–282, 2010.

[14] Xintian Yang, Srinivasan Parthasarathy, and P. Sadayappan. Fast Sparse Matrix-Vector Multiplication on GPUs: Implications for Graph Mining, In *Proceedings of the VLDB Endowment*, Vol. 4, No. 4, pp. 231–242, 2011.

[15] NVIDIA. CUDA C Programming Guide. `http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf`, 2012.

[16] NVIDIA. CUFFT Library User Guide. `http://developer.download.nvidia.com/compute/DevZone/docs/html/CUDALibraries/doc/CUFFT_Library.pdf`, 2012.

[17] Mark Harris. Optimizing Parallel Reduction in CUDA. `http://developer.download.nvidia.com/compute/cuda/2_2/sdk/website/projects/reduction/doc/reduction.pdf`.