複数GPUによる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　実世界に大量に存在する，不確実性を含むデータを，効果的かつ効率的に扱う技術が近年注目を集めている．このような不確実データに対して，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きている．しかし，不確実性を考慮した処理の計算量が多いため，巨大なデータを扱うためにはさらなる高速化が必要と考えられる．そこで，我々はGPGPUを用いた高速化の研究を行なっている．先行研究では単一GPUによる手法を提案し，その有効性を示した．本稿では，これを発展させ，複数のGPUを用いる手法を提案する．提案手法は，CPU-GPU間のデータ転送を削減し，かつGPU間の負荷分散を行なう．さらに，実験により提案手法の性能を評価する．

キーワード　GPGPU, 不確実データベース, 頻出アイテム集合マイニング

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

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

1. 序　　論

実世界では不確実性を含むデータが爆発的に増加しており，これらを扱う技術が近年非常に注目されてきている．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．これ以外にも，ノイズやプライバシ保護などの様々な理由によって不確実データが生成されうる．このようなデータを効果的かつ効率的に処理するために，不確実データベースの研究が広く行なわれてきている [1].

このとき，不確実データを対象に，主要なデータマイニング手法の一つである頻出アイテム集合マイニングを適用することも可能であり，多くの研究が見られる [5], [6], [16], [17]．ここで，データの不確実性は確率によって表現されるため，この問題は確率的頻出アイテム集合マイニングと呼ばれる．この問題に対する手法がいくつか存在するが，確率的頻出アイテム集合マイニング中の確率計算の処理が計算量の多いものであるため，大量のデータを処理するためにはさらなる高速化が必要である．

そこで，我々はGPGPU (General-Purpose computation on GPU) を用いた，確率的頻出アイテム集合マイニングの高速化を行なっている [10]．GPGPUは，元々はグラフィック処理用のプロセッサであるGPUを，グラフィック処理以外の一般的な処理の高速化に利用する手法である．GPUは数百以上の演算ユニットを有しているため，極めて並列度の高いデータ処理を行なうことができる．また，GPUを効果的に用いることで，省電力かつ安価でCPU以上の性能を出すことが可能なため，近年非常に注目されており，多くの研究がなされている [7], [9], [13], [15]．さらに，GPUはその構造が比較的単純であるため，処理性能の向上が目覚ましく，GPGPUは今後ますます重要な技術になると考えられる．

先行研究では，一つのGPUのみを用いた確率的頻出アイテム集合マイニングの高速化を行なってきた．本稿では，これを発展させ，複数GPUを用いた手法を提案する．複数GPUを用いる利点として，GPUの増加による並列度の向上がある．また，GPUのメモリはそれほど大きくないが，複数GPUにすることで利用可能なメモリ領域が増加し，巨大なデータを扱

うことも可能となる．一方，複数 GPU を利用する上で問題になる点として，GPU 間でのデータ通信がある．通常，GPU は PCI-Express によって接続されているが，ここの通信のメモリバンド幅が小さいため，ボトルネックになる可能性が高い．そのため，GPU 間でのデータの依存をできるだけ削減することが望ましい．提案手法では，GPU 間でのデータ通信の削減に加え，負荷分散を行ない処理の高速化を図る．さらに，複数 GPU を持つノードからなるクラスタにおける手法も提案する．また，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 節で，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．3 節で提案手法について説明し，4 節で評価実験の結果と考察を述べる．関連研究について 5 節で説明し，最後に 6 節でまとめと今後の課題について述べる．

2. 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説明する．まず，従来の，不確実でないデータベースに対する問題の定義を 2.1 節で述べ，2.2 節で確率的頻出アイテム集合マイニングの定義を行なう．2.3 節では Sun ら [16] の提案した pApriori アルゴリズムについて説明する．

2.1 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と言い，要素数 k のアイテム集合を k-アイテム集合と呼ぶ．ID とアイテム集合のペアをトランザクション T，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．トランザクション数 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．

2.2 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．表 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 0.8，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は一般的に，可能世界意味論によって解釈される．すなわち，不確実トランザクションデータベースは，それぞれ別々のトランザクションの組み合わせからなる可能世界の集合を生成すると考える．例えば，表 1 のデータベースは，$\emptyset, \{T_1\}, \{T_2\}, \{T_3\}, \{T_4\}, \{T_1, T_2\}$ など，全部で 16 個の可能世界を生成する．また，各可能世界はそれぞれ存在確率を持つ．この存在確率は，各世界に含まれるトランザクションをもとに計算される．例として表 1 における可能世界 $\{T_1\}$ を考えると，その存在確率は，T_1 が存在する確率と T_2, T_3, T_4 が存在しない確率から求められる．すなわち，その確率は $0.8 \cdot (1-0.7) \cdot (1-0.9) \cdot (1-0.5) = 0.012$ となる．

表 1　不確実トランザクションデータベースの例

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

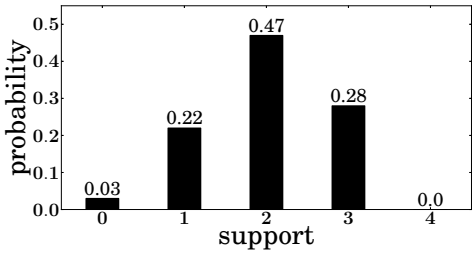


図 1　支持度確率質量関数 $f_{\{\text{music}\}}$.

このとき，アイテム集合 X の支持度 $\sup(X)$ は各可能世界ごとに異なるため，確率変数になると考えられ，支持度の分布は確率質量関数によって表される．アイテム集合 X の支持度の確率質量関数を支持度確率質量関数 (Support Probability Mass Function, SPMF) f_X と呼ぶ．特に，$f_X(k)$ は $\sup(X) = k$ となる確率を示す．ここで，k は $\sup(X)$ が取りうる値，すなわち $k \in \{0, 1, ..., |\mathcal{U}|\}$ である．例えば，表 1 における SPMF $f_{\{\text{music}\}}$ は図 1 のようになる．横軸は支持度，縦軸はその支持度となる確率を表している．

以下の条件を満たすアイテム集合を確率的頻出アイテム集合と呼ぶ：

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

ここで，$\texttt{minprob} \in (0, 1]$ はユーザに与えられるしきい値である．例えば，$\texttt{minsup} = 2, \texttt{minprob} = 0.5$ としたとき，表 1 の不確実トランザクションデータベースにおいて，

$$\begin{aligned} P(\sup(\{\text{music}\}) \geq \texttt{minsup}) &= f_{\{\text{music}\}}(2) + f_{\{\text{music}\}}(3) \\ &= 0.47 + 0.28 \\ &= 0.75 > \texttt{minprob} \end{aligned}$$

となるため，アイテム集合 {music} は確率的頻出アイテム集合である．

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）． 不確実トランザクションデータベース $\mathcal{U}$，`minsup`, `minprob` の三つが与えられたとき，$\mathcal{U}$ からすべての確率的頻出アイテム集合を抽出する．

2.3 pApriori アルゴリズム

pApriori アルゴリズム [16] は，Agrawal ら [3] 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，1-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，$k = 2$ として処理を行なってい

く．確率的頻出 $(k-1)$-アイテム集合から，候補 k-アイテム集合を生成し，これらの候補の中から確率的頻出 k-アイテム集合を抽出する．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．以下の節で，候補の生成と確率的頻出アイテム集合の抽出について説明していく．

2.3.1 候補アイテム集合の生成

候補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，二つの確率的頻出 $(k-1)$-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 $(k-1)$-アイテム集合 X, Y が結合可能とは，$X.\mathrm{item}_i$ が X 中の i 番目のアイテムを表すとすると，$X.\mathrm{item}_1 = Y.\mathrm{item}_1 \wedge \cdots \wedge X.\mathrm{item}_{k-2} = Y.\mathrm{item}_{k-2} \wedge X.\mathrm{item}_{k-1} < Y.\mathrm{item}_{k-1}$ が成り立つことをいう．結合可能ならば，X と Y の和集合から，候補 k-アイテム集合を作り，その候補を候補 k-アイテム集合の集合 $\mathcal{C}_k$ 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う [16]．

［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，結合フェイズで作成した各候補 k-アイテム集合について，その要素数 $k-1$ 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 C_k からその候補を削除する．

2.3.2 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からなる場合は，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 $\mathcal{U}$ を二つに水平分割する．そして，それぞれのデータベースに対して再帰的に SPMF を計算し，計算された二つの SPMF を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．

2.3.3 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 $\mathrm{cnt}(X)$ と，$\sup(X)$ の期待値を表す $\mathrm{esup}(X)$ である．

このとき，以下の二つの補題が成り立つ [16]．

［補題 2］ $\mathrm{cnt}(X) < \texttt{minsup}$ 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］ $\mu = \mathrm{esup}(X)$, $\sigma = (\texttt{minsup} - \mu - 1) \ / \ \mu$ 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．

3. GPGPU を用いた確率的頻出アイテム集合マイニング

本節では，GPU 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述べる．3.1 節では先行研究の単一 GPU による手法 [10] について簡単に説明する．その後，3.2 節で単一ノード上での複数 GPU による手法を，3.3 節で複数ノード上の手法を述べる．

3.1 単一 GPU による高速化

単一 GPU による手法の基本的な流れは pApriori アルゴリズムと同様で，候補の生成と確率的頻出アイテム集合の抽出の二つの処理からなる．候補の生成は 2.3.1 節で説明したアルゴリズムにより GPU 上で行なう．また，確率的頻出アイテム集合の抽出は，(1) 包含判定，(2) 枝刈り，(3) フィルタリング，(4) SPMF の計算，の四つのステップからなる．

包含判定では，生成された各候補が，不確実トランザクションデータベースの各トランザクションに含まれているかを判定する．一つのトランザクションに対する判定結果は 1 ビットで表現することが可能なため，判定結果はビット列によって保持する．これにより，$k > 1$ の場合は，$k-1$ のときの包含判定結果を利用することで，高速に処理することが可能となる．すなわち，ある k-アイテム集合の包含判定をするには，そのアイテム集合の生成に利用された，二つの確率的頻出 $(k-1)$-アイテム集合の包含判定結果の間のビット単位の AND 演算を行なえばよい．

二つめのステップの枝刈りでは，2.3.3 節で説明した二つの値を計算し枝刈りを行なう．枝刈りされなかった候補に対しては，フィルタリングを行なう．ここでは，候補を含んでいないトランザクションを，SPMF の計算から除くための前処理をする．これは，トランザクションが候補を含まない場合，そのトランザクションの存在確率は候補の SPMF に影響しないためである．この前処理の後，実際に SPMF を計算し，候補が確率的頻出アイテム集合であるかを判定する．

3.2 単一ノード・複数 GPU による高速化

複数 GPU を用いる場合，メモリが分散して存在するため，いかにデータを分割するかが重要となる．分散メモリ環境における頻出アイテム集合マイニングの研究は数多く存在する [4], [18]．これらの研究のデータの分割方式は主に (1) 支持度のカウントの分割，(2) データベースの分割，(3) 候補の分割，の三つに分けられる．本稿では，候補の分割に基づく手法を提案する．これにより，pApriori アルゴリズム中で最も計算量の多い SPMF

Algorithm 1: Prefix partitioning

```
1  k = 1
2  |  包含判定・枝刈り // CPU
3  |  CPU から各 GPU に候補とすべての包含判定結果を転送
4  |  フィルタリング・SPMF の計算 // GPU
5  |  確率的頻出 1-アイテム集合を CPU 側に集約
6  k = 2
7  |  候補 k-アイテム集合の生成 // CPU
8  |  候補を prefix をもとに分割し GPU に転送
9  while 候補集合 C_k ≠ ∅ do
10 |  // 包含判定には k − 1 の包含判定結果を利用
11 |  確率的頻出 k-アイテム集合の抽出 // GPU
12 |  確率的頻出 k-アイテム集合を CPU 側に集約
13 |  各 GPU にすべての確率的頻出 k-アイテム集合を転送
14 |  k ← k + 1
15 |  候補 k-アイテム集合の生成 // GPU
```

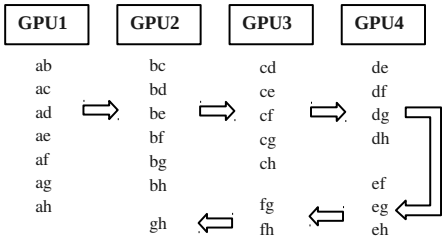


図 2　Prefix partitioning

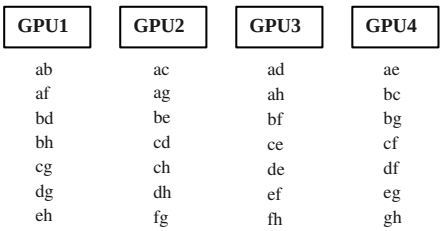


図 3　Round-robin partitioning

の計算を，各 GPU が独立して処理できるようになる．3.2.1 節で，単純な候補の分割方式による手法を，3.2.2 節と 3.2.3 節では負荷分散を考慮した手法を説明する．

3.2.1　Prefix Partitioning

ここでは，二つのアイテム集合の prefix が異なる場合には結合可能とならない，という性質を生かした手法について説明する．本手法における処理の流れを Algorithm 1 に示す．この手法は候補の prefix をもとに分割を行なっているため，*Prefix Partitioning (PP)* と呼ぶ．PP は (1) $k = 1$ の処理, (2) $k = 2$ における候補の分割, (3) その後の処理，の三つのフェイズに分けられる．

まず，不確実トランザクションデータベースから候補 1-アイテム集合を生成し，それらに対し CPU 上で包含判定と枝刈りを行なう（2 行目）．GPU ではなく，CPU 上でこの処理をすることで，データベースを GPU に転送する必要が無くなり，データ転送量を削減できる．また，単一 GPU の手法では，GPU の特性上これらのステップを別々に行なったほうが高速であったが，実際にはこれらのステップはまとめて処理可能であるため，ここでは同時に処理している．この時点で残っている候補を等分して，各 GPU に転送する（3 行目）．このとき同時に，後の包含判定で必要となるため，すべての候補の包含判定結果も転送しておく．その後，各 GPU の担当する候補に対してフィルタリング・SPMF の計算をし（4 行目），CPU 側に確率的頻出 1-アイテム集合を転送する（5 行目）．

$k = 2$ では，5 行目で集めた確率的頻出 1-アイテム集合をもとに，候補 2-アイテム集合を生成する（7 行目）．そして，候補を prefix をもとに分割し GPU に転送する（8 行目）．例えば，8 個の確率的頻出 1-アイテム集合 {a}, {b}, {c}, {d}, {e}, {f}, {g}, {h} が発見され，GPU が四つあると仮定すると，図 2 のように候補が割り当てられる．各 GPU に対して割り当てられる候補の数をできるだけ均等にするために，図 2 のようにジグザグに prefix による割り当てを行なう．すなわち，GPU1 から 4 に prefix が a から d の候補をそれぞれ割り当てたあと，prefix が e の候補を GPU4 に，prefix が f の候補を GPU3 に，という順番で割り当てていく．

次に，3.1 節で述べた手法により，各 GPU 上で確率的頻出 k-アイテム集合の抽出をする（11 行目）．その後，発見された確率的頻出 k-アイテム集合を CPU 側に集約し（12 行目），これらをすべての GPU にブロードキャストする（13 行目）．これは，次に行なう候補の生成の枝刈りフェイズに必要なためである．ただし，候補の生成のもととなる確率的頻出アイテム集合はそれぞれの GPU 上で発見されたもののみを用いる．このように，9–15 行目を候補が無くなるまで繰り返す．

PP は候補を prefix にしたがって分割することで，$k = 2$ 以降は GPU 間のデータの依存関係を少なくできる．しかし，prefix に従って候補を分割すると，SPMF の計算が必要な候補が多いグループとそうでないグループができる可能性があり，負荷分散の観点からは望ましくないものとなる．この点を改善した手法を以下の節で述べる．

3.2.2　Round-Robin Partitioning

本節で説明する手法は，候補を round-robin 方式で分割するため，*Round-Robin Partitioning (RRP)* と呼ぶ．基本的な流れは PP と同様である．主な違いは，Algorithm 1 の 8 行目，10–11 行目，13–15 行目に存在する．具体的に言うと，8 行目を

```
8  ラウンドロビン方式で候補を GPU に割り当て転送
```

に，10–11 行目を

```
10 // 包含判定には k = 1 の包含判定結果を利用
11 確率的頻出 k-アイテム集合の抽出 // GPU
```

に，13–15 行目を

```
13 k ← k + 1
14 候補 k-アイテム集合の生成 // CPU
15 ラウンドロビン方式で候補を GPU に割り当て転送
```

にしたものとなる．

表 2　ノードの構成

	CPU	GPU
Processor	Intel Xeon E5645 (2.4GHz, 6 コア) x2	NVIDIA Tesla M2090 (1.3GHz, 512 コア) x2
メモリ	48 GB	6GB/GPU

表 3　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	密度
Accidents	468	33.8	340,183	7.2%
T25I10D500K	7558	25	499,960	0.33%
Kosarak	41270	8.1	990,002	0.020%

8 行目では，図 3 に示すように，各 GPU にラウンドロビン方式で順々に候補を割り当て，転送をする．

10–11 行目での大きな違いは，GPU 上で包含判定をする際に $k-1$ の包含判定結果ではなく $k=1$ のものを利用する点である．RRP では prefix による分割ではないために，ある k-アイテム集合の生成元が別々の GPU に分散している可能性があり，$k-1$ の包含判定結果を利用するには別の GPU が保持しているデータを参照する必要がある．しかし，この処理には GPU 間のデータ転送が必要なため，避けるべきである．一方，$k=1$ の場合の包含判定結果は最初に GPU に転送してあるので，これを利用することで無駄な通信を削減できる．k-アイテム集合 X の包含判定は，各アイテム $i \in X$ の包含判定結果を用いて，$k-1$ 回のビット単位の AND で処理可能である．

13–15 行目での違いは，候補の生成を GPU 上ではなく CPU 上で行ない，ラウンドロビンで候補の分割を行なう点である．これにより，毎イテレーションにおいて負荷分散を行なうことができる．また，候補の転送のみなのでデータの転送量もそれほど多くならず，通信時間は小さく抑えられる．

3.2.3 Count-Based Partitioning

本節では，候補の cnt の値をもとに候補の分割を行なう手法 *Count-Based Partitioning (CBP)* について説明する．3.1 節で述べたように，アイテム集合 X の SPMF の計算には $\mathrm{cnt}(X)$ 個のトランザクションのみを用いればよい．すなわち，SPMF の計算時間は cnt の値に依存する．また，GPU の処理時間の大部分を SPMF の計算が占めている．そのため，cnt を考慮して候補を分割することで，GPU 間の負荷分散の達成が期待される．

GPU 上では，SPMF の計算の単純化のために，$\mathrm{cnt}(X)$ 個ではなく $2^{\lceil \log_2 \mathrm{cnt}(X) \rceil}$ 個のトランザクションを用いている [10]．そのため，2 のべき乗個のトランザクションに対する SPMF の計算時間がわかれば，候補の処理時間の推定ができる．ここでは，これらの時間をあらかじめ計測しておき，その値を候補の処理時間の推定値として利用する．計測した値は実行環境依存で，データには依存しないため，一度計測すればあとは使いまわすことができる．

CBP の処理の流れは RRP と同様で，候補の分割方式のみが異なる．候補の分割は，各 GPU が担当する候補の処理時間の合計が均等になるように行なう．まず，各 GPU の担当処理時間を 0 に初期化する．ここで，GPU の担当処理時間とは，GPU に割り当てられた候補の推定処理時間の合計を意味する．その後，各候補に対して，担当処理時間が最小の GPU を探しその GPU に候補を割り当てる，という処理を繰り返す．

$k=1$ の場合はアイテム集合 X の $\mathrm{cnt}(X)$ が候補の分割時点で計算されているため，処理時間の推定に $\mathrm{cnt}(X)$ の値をそのまま利用できる．一方，$k>1$ の場合には，候補の分割時点では cnt が計算されていない．そこで，重みを計算する際には，アイテム集合 X の $\mathrm{cnt}(X)$ を $\min_{X' \subset X, |X'|=k-1} \mathrm{cnt}(X')$ で代替する．これは，$\mathrm{cnt}(X)$ の上限の値を表し，実際にはこれよりも低い値となることが予想される．しかし，処理時間の推定に必要なのは $\mathrm{cnt}(X)$ 以上の 2 のべき乗の値であるため，ある程度の誤差ならば結果には影響がないと考えられる．

3.3 複数ノード・複数 GPU による高速化

本節では，各ノードが複数の GPU をもつクラスタ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述べる．ここでは，すべてのノードが入力の不確実トランザクションデータベースを保持していると仮定する．

基本的には 3.2 節で説明した手法と同様の方式で，各ノードに候補を分割して並列処理を行なう．ノード間の候補の分割には RRP または CBP を用いる．具体的な処理の流れは以下のようになる．$k=1$ の場合は，Algorithm 1 の 2–5 行目を各ノードが実行する．ここで，各ノードが処理する候補はノードごとに異なるものを対象とするため，抽出される確率的頻出アイテム集合も異なる．そのため，各ノードで確率的頻出アイテム集合を発見したあとに，次の候補生成のために他のノードにその結果をブロードキャストする必要がある．

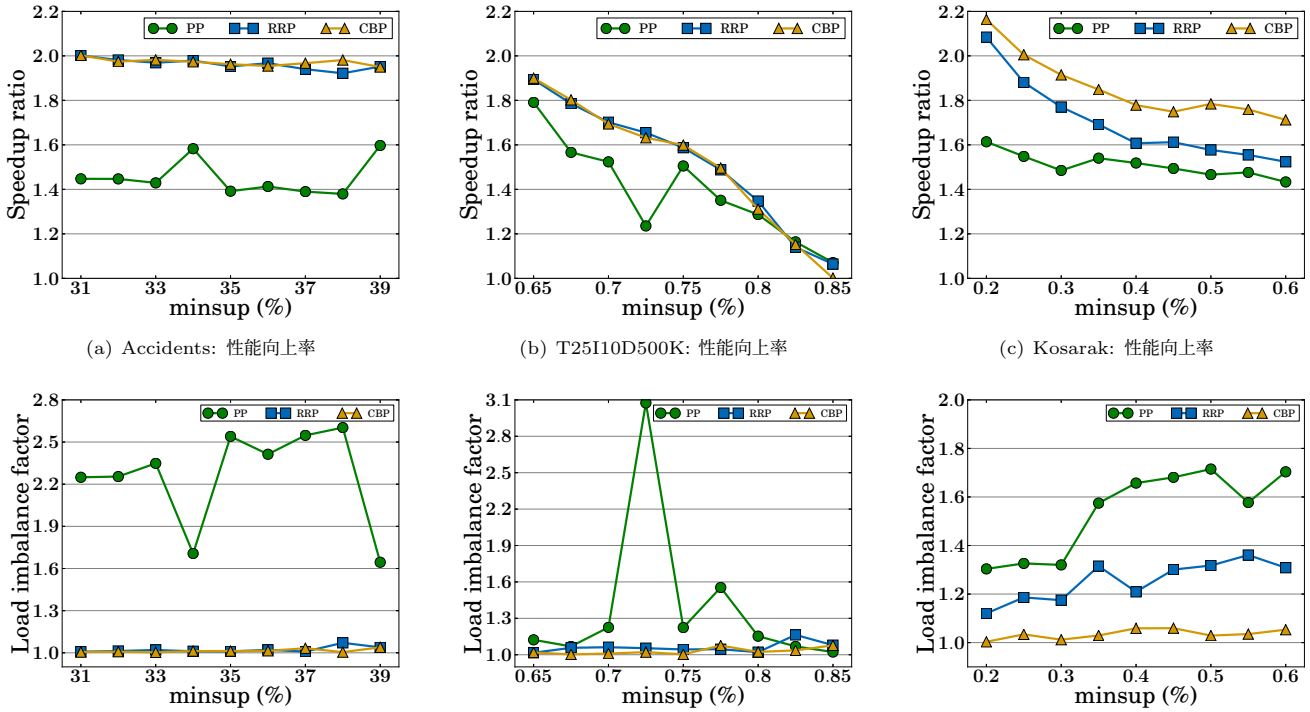
次に $k=2$ に進み，候補の生成を行なう．各ノードが担当する候補は RRP または CBP によって決定する．このとき，ノード内の候補の分割も同様の手法を用いる．その後，各ノードで見つかった確率的頻出 2-アイテム集合をブロードキャストし，k を 1 増やし次の反復に進む．以上のように，ノード間の通信を確率的頻出 k-アイテム集合のみにすることで，通信時間を極力削減している．

4. 実　　験

4.1 実験環境

提案手法を CUDA(注1), OpenMP, MPI を用いて実装した．実験は，表 2 に示したノード 8 個からなるクラスタ上で行なっ

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



(a) Accidents: 性能向上率　(b) T25I10D500K: 性能向上率　(c) Kosarak: 性能向上率

(d) Accidents: 各 GPU における SPMF の計算時間の偏り　(e) T25I10D500K: 各 GPU における SPMF の計算時間の偏り　(f) Kosarak: 各 GPU における SPMF の計算時間の偏り

図 4　単一ノード上で複数 GPU を用いた実験結果

た．ノード間は InfiniBand QDR (Quad Data Rate) で接続されている．

三つのデータセットに対して実験を行なった．各データセットの特徴を表 3 に示す．密度は，トランザクションの平均要素数をアイテム数で割ったものである．Accidents と Kosarak は，FIMI Repository(注2)にある実データである．Accidents は比較的密なデータセットで，Kosarak は三つの中で最も疎なデータセットである．明記していない場合，`minsup` を，Accidents ではトランザクション数の 33%，Kosarak ではトランザクション数の 0.2%とする．T25I10D500K は，IBM data generator(注3)によって生成された合成データである．`minsup` はデータセットのサイズの 0.65%とする．これらのデータセットに対して，トランザクションの存在確率を平均 0.5，分散 0.02 の正規分布により与える．また，デフォルトの `minprob` を 0.5 とする．

4.2　単一ノードによる実験

本節では，単一ノード上の複数 GPU による手法について評価する．図 4 に実験結果を示す．図 4(a)–4(c) は，単一 GPU による手法 [10] に対する，三つの提案手法 (PP, RRP, CBP) の性能向上率を示している．これらのグラフから以下の三つのことがわかる．

（1） `minsup` の値が小さくなるにつれて，複数 GPU の手法の性能向上率があがっていく．

（2） Accidents と T25I10D500K では，RRP と CBP が PP と比較して常に良い性能を示している．

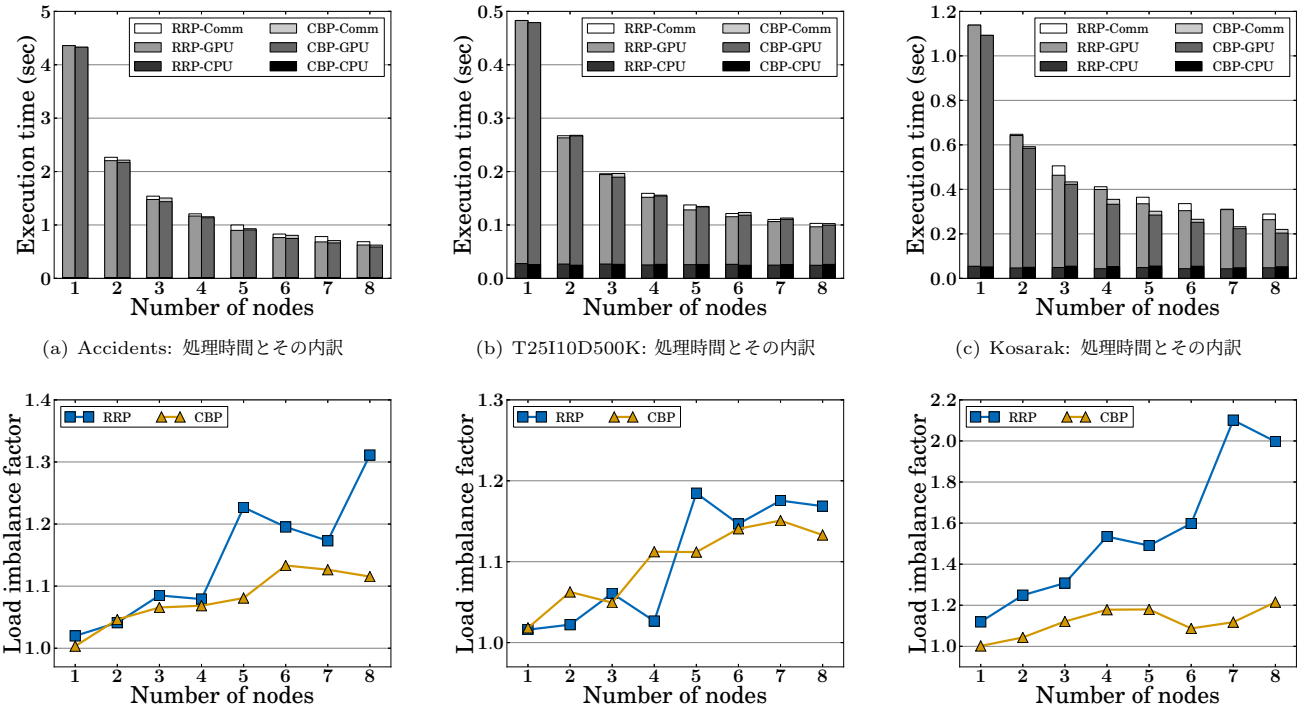
（3） Kosarak では CBP が最も良い性能を示している．

(1) の理由は，`minsup` の値が小さくなると候補の数が増加し，各 GPU が並列に処理できる仕事が増えるためである．ただし，Accidents はアイテム数が少なく候補の数はそれほど多くならないが，良い性能向上率となっている．これは以下のように考えられる．SPMF の計算量は 2.3.3 節で述べた cnt に依存しているため，この値が大きいと SPMF の処理に時間がかかる．そして，Accidents では cnt の値が極めて大きくなるため，SPMF の計算時間が支配的となる．そのため，Accidents では，候補が少ない場合でも常に 2 倍程度の性能を示している．また，図 4(c) で `minsup` が 0.2%の場合に 2 倍以上の性能となっているのは，候補の数が非常に多くなった場合に，GPU 上の候補生成よりも，CPU 上の候補生成が高速になるためである．

(2) の理由は，RRP と CBP が，PP よりも上手く負荷分散を行なえているためである．図 4(d)–4(f) は各提案手法を用いたときの GPU 間の SPMF の計算時間の偏り度合いを示している．縦軸は，各 GPU で行なった SPMF の計算時間をそれぞれ合計し，最大の計算時間を最小の計算時間で割ったものを示している．すなわち，1 の場合が最も負荷分散されていて，大きくなるほど負荷が偏っていることを示している．図 4(d)–4(e) から，PP では負荷が大きく偏っており，RRP と CBP は負荷分散が達成されていることがわかる．Accidents と T25I10D500K ではでは，候補の cnt の値にあまりばらつきがないため，RRP と CBP の性能はほぼ同じになっている．

(3) の理由は，他の二つのデータセットと違い，Kosarak では候補の cnt の値に大きなばらつきがあるためである．この場合，RRP では上手く負荷が分散されないが，CBP では cnt の

(注2)：`http://fimi.cs.helsinki.fi/`
(注3)：`http://miles.cnuce.cnr.it/~palmeri/datam/DCI/datasets.php`



(a) Accidents: 処理時間とその内訳　(b) T25I10D500K: 処理時間とその内訳　(c) Kosarak: 処理時間とその内訳

(d) Accidents: 各 GPU における SPMF の計算時間の偏り　(e) T25I10D500K: 各 GPU における SPMF の計算時間の偏り　(f) Kosarak: 各 GPU における SPMF の計算時間の偏り

図 5　複数ノードを用いた実験結果

値を考慮しているため負荷分散が達成されている．

図 4(b) で PP の性能向上率が大きく下がっている箇所があるのは以下のような理由による．prefix ごとに生成される確率的頻出アイテム集合の数にばらつきがあるため，抽出された確率的頻出 1-アイテム集合によって，仕事が上手く分散される場合とされない場合がある．ここでは，図 4(e) からもわかるように，GPU 間で大きく仕事が偏ってしまい大幅に性能が下がった．

4.3　複数ノードによる実験

本節では，複数ノードを用いた二つの手法 CBP と RRP について評価する．図 5 に実験結果を示す．ここでは，各ノードがデータを読み終えてから，マスターノードにすべての確率的頻出アイテム集合を集約するまでを処理時間としている．

図 5(a)–5(c) は，各データセットに対してデフォルトの `minsup` を用い，ノード数を変化させたときの RRP と CBP の処理時間とその内訳を示している．処理時間は，通信時間 (`Comm`)，GPU での処理時間 (`GPU`), CPU での処理時間 (`CPU`) の三つからなる．ここで，`GPU` は $k > 1$ の包含判定，$k > 1$ の枝刈り，フィルタリング，SPMF の計算を含み，`CPU` は $k = 1$ の包含判定と枝刈り，候補生成を含む．また，図 5(d)–5(f) は，4.2 節と同様に GPU 間の負荷の偏り度合いを示している．

Accidents の場合，CBP が RRP と比較して良い性能を示しており，最大 7 倍の高速化を達成している．4.2 節でも述べたとおり，Accidents では SPMF の計算に非常に時間がかかるため，各 GPU が並列に処理できる仕事量が多い．また，図 5(d) から，CBP の方が負荷分散できているため，良い性能向上率となった．

T25I10D500K では，CBP と RRP はほぼ同じ性能で最大 4.5 倍の高速化となった．ノード数を増やしてもあまり高速化を達成できていないのは，候補の cnt の値が小さく GPU が行なう処理が少ないためである．そのため，すべてのノードが処理する $k = 1$ の包含判定と枝刈りが，全体の処理時間の 10–25%を占めており，この部分がボトルネックとなる．また，CBP と RRP がほぼ同じ性能となったのは，候補間で cnt にあまりばらつきがないため，RRP でも負荷分散ができているためである（図 5(e)）．

一方，Kosarak では cnt の値にばらつきがあり，ノード数が増えるにつれて RRP の場合の負荷の偏りが大きくなっていく（図 5(f)）．CBP では負荷が分散されているため，RRP と比較して 1.3 倍程度の性能となった．

5. 関 連 研 究

本節では，関連研究について，頻出アイテム集合マイニング，確率的頻出アイテム集合マイニング，GPGPU の三つの観点から述べる．

頻出アイテム集合マイニング． 相関ルールマイニングの問題は Agrawal らによって初めて導入された [2]．頻出アイテム集合マイニングは，相関ルールマイニングの部分問題の一つであり，処理時間の大部分を占める．そのため，この問題を効率的に処理するために数多くのアルゴリズムが研究されてきている．最も主要なアルゴリズムとして，Apriori [3] と FP-growth [8] がある．

また，これらのアルゴリズムの並列化の研究も数多くなされている．Agrawal ら [4] は Apriori をもとに，シェアード・ナッシング・アーキテクチャにおける三つの手法を提案した．この

他にも，1990 年代において，多くの研究が存在した [18]．これらの研究では主に分散メモリ環境を対象としている．

近年では，CPU のコア数の増加に伴い，マルチコアを活用する手法が見られる．Parthasarathy ら [14] は，共有メモリのマルチプロセッサにおいて，負荷分散やデータの局所性を考慮した Apriori の並列化を行なった．Liu ら [11] は，現代のマルチコアプロセッサの活用を目指して，キャッシュを意識した FP-growth の並列アルゴリズムについて研究した．

確率的頻出アイテム集合マイニング．上記の並列アルゴリズムは従来のトランザクションデータベースが対象の場合は効率的に処理可能であるが，これらを不確実データベースにそのまま適用するのは実用的ではない．不確実データベースは近年非常に注目を集めており，様々な分野で多数の研究が行なわれている [1]．

不確実性を考慮した頻出アイテム集合マイニングの研究も盛んに行なわれており，多数の手法が存在する．Chui ら [6] は Apriori を不確実トランザクションデータベースへと適用した，U-Apriori を提案した．彼らは頻出アイテム集合の判定に支持度の期待値を用いていた．しかし，後に Bernecker ら [5] は，支持度の期待値では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．代わりに，支持度の分布を考慮した手法を提案した．上記の研究は各アイテムに存在確率が付与されている，attribute-uncertainty model を対象としているが，Sun ら [16] は各トランザクションに存在確率が付与されている tuple-uncertainty model を対象とした研究を行なった．また，Tong ら [17] は既存の代表的な手法を実装し，それらを公平な基準をもとに性能を評価した．

GPGPU．GPGPU は，高性能計算分野だけでなくデータベース分野などにおいても，近年注目されている技術の一つである．例えば，He ら [9] は CPU-GPU のハイブリッドな手法による，関係演算の高速化の研究を行なった．また，GPGPU に関するサーベイ論文も存在する [13]．

GPU を用いた頻出アイテム集合マイニングの手法はいくつかの研究がある．Fang ら [7] は GPU ベースの手法と CPU-GPU のハイブリッドな手法の二つを提案した．また，Silvestri ら [15] は DCI アルゴリズム [12] の GPU による並列化を行なった．

6. ま と め

本稿では，先行研究で行なった単一 GPU による確率的頻出アイテム集合マイニングの手法を発展させ，候補の分割にもとづく複数 GPU を用いた手法を提案した．提案手法 PP では，候補の prefix をもとに分割し，単一ノード上で複数 GPU による並列処理を行なった．さらに，負荷分散を考慮した二つの手法 RRP と CBP を提案した．また，RRP と CBP を発展させ，複数 GPU を含むノードからなるクラスタを対象とした手法を開発した．評価実験により，CBP が最も良い性能となることを示した．さらに，クラスタを対象とした手法は，GPU が行なう仕事が十分にある場合に有効であることを示した．

謝辞　本研究の一部は科学研究費補助金基盤 A (24240015A)，「エクサスケール計算技術開拓による先端学際計算科学教育研究拠点の充実」事業による．

文　献

[1] C. C. Aggarwal and P. S. Yu, “A Survey of Uncertain Data Algorithms and Applications,” *IEEE Trans. Knowl. Data Eng.*, vol. 21, pp. 609–623, May 2009.

[2] R. Agrawal, T. Imieliński, and A. Swami, “Mining Association Rules between Sets of Items in Large Databases,” *Proc. ACM SIGMOD Int’l Conf. Management of Data (SIGMOD)*, pp. 207–216, 1993.

[3] R. Agrawal and R. Srikant, “Fast Algorithms for Mining Association Rules,” *Proc. 20th Int’l Conf. Very Large Data Bases (VLDB)*, pp. 487–499, 1994.

[4] R. Agrawal and J. C. Shafer, “Parallel Mining of Association Rules,” *IEEE Trans. Knowl. Data Eng.*, vol. 8, pp. 962–969, Dec. 1996.

[5] T. Bernecker, H.-P. Kriegel, M. Renz, F. Verhein, and A. Zuefle, “Probabilistic Frequent Itemset Mining in Uncertain Databases,” *Proc. 15th ACM SIGKDD Int’l Conf. Knowledge Discovery and Data Mining (KDD)*, pp. 119–128, 2009.

[6] C.-K. Chui, B. Kao, and E. Hung, “Mining Frequent Itemsets from Uncertain Data,” *Proc. 11th Pacific-Asia Conf. Advances in Knowledge Discovery and Data Mining (PAKDD)*, pp. 47–58, 2007.

[7] W. Fang, M. Lu, X. Xiao, B. He, and Q. Luo, “Frequent Itemset Mining on Graphics Processors,” *Proc. Fifth Int’l Workshop on Data Management on New Hardware (DaMoN)*, pp. 34–42, 2009.

[8] J. Han, J. Pei, and Y. Yin, “Mining Frequent Patterns without Candidate Generation,” *Proc. ACM SIGMOD Int’l Conf. Management of Data (SIGMOD)*, pp. 1–12, 2000.

[9] B. He, M. Lu, K. Yang, R. Fang, N. K. Govindaraju, Q. Luo, and P. V. Sander, “Relational Query Coprocessing on Graphics Processors,” *ACM Trans. Database Syst.*, vol. 34, pp. 21:1–21:39, Dec. 2009.

[10] Y. Kozawa, T. Amagasa, and H. Kitagawa, “GPU Acceleration of Probabilistic Frequent Itemset Mining from Uncertain Databases,” *Proc. 21st Int’l Conf. Information and Knowledge Management (CIKM)*, pp. 892–901, 2012.

[11] L. Liu, E. Li, Y. Zhang, and Z. Tang, “Optimization of Frequent Itemset Mining on Multiple-Core Processor,” *Proc. 33rd Int’l Conf. Very Large Data Bases (VLDB)*, pp. 1275–1285, 2007.

[12] S. Orlando, P. Palmerini, R. Perego, and F. Silvestri, “Adaptive and Resource-Aware Mining of Frequent Sets,” *Proc. IEEE Int’l Conf. Data Mining (ICDM)*, pp. 338–345, 2002.

[13] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, “GPU Computing”, *Proc. IEEE*, vol. 95, pp. 879–899, May 2008.

[14] S. Parthasarathy, M. J. Zaki, M. Ogihara, and W. Li, “Parallel Data Mining for Association Rules on Shared-Memory Systems,” *Knowl. Inf. Syst.*, vol. 3, pp. 1–29, Feb. 2001.

[15] C. Silvestri and S. Orlando, “gpuDCI: Exploiting GPUs in Frequent Itemset Mining,” *Proc. 20th Euromicro Int’l Conf. Parallel, Distributed and Network-based Processing (PDP)*, pp. 416–425, 2012.

[16] L. Sun, R. Cheng, D. W. Cheung, and J. Cheng, “Mining Uncertain Data with Probabilistic Guarantees,” *Proc. 16th ACM SIGKDD Int’l Conf. Knowledge Discovery and Data Mining (KDD)*, pp. 273–282, 2010.

[17] Y. Tong, L. Chen, Y. Cheng, and P. S. Yu, “Mining Frequent Itemsets over Uncertain Databases,” *Proc. VLDB Endow.*, vol. 5, no. 11, pp. 1650–1661, Jul. 2012.

[18] M. J. Zaki, “Parallel and Distributed Association Mining: A Survey”, *IEEE Concurrency*, vol. 7, no. 4, pp. 14–25, Oct. 1999.