プライバシー保護クロス集計における行列演算の GPU による最適化

永井　彰 †　　五十嵐　大 †　　松林　達史 ‡　　山本　剛 †

† 日本電信電話株式会社 NTT 情報流通プラットフォーム研究所
180-8585 東京都武蔵野市緑町 3-9-11
‡ 日本電信電話株式会社 NTT コミュニケーション科学基礎研究所
619-0237　京都府相楽郡精華町光台 2-4

†{ nagai.akira , ikarashi.dai, yamamoto.go }@lab.ntt.co.jp
‡tatsushi@cslab.kecl.ntt.co.jp

あらまし　近年，個人のプライバシー情報や企業の機密情報を保護しながら利活用するニーズが高まってきており，一つにデータマイニングやアンケート集計においてプライバシーを保護しながらも結果を得るプライバシー保護クロス集計の研究が盛んになってきている．このプライバシー保護クロス集計を実現する技術の 1 つとして，再構築法が知られている．しかし，この計算法では属性数が増加した場合に，メモリの容量不足が問題となる．この問題の解決手段として，いくつかの実装方法が考えられるが逆に計算時間が課題となってしまう．本稿では，並列計算デバイス GPU(Graphic Processing Unit) を用いることで計算力を補い，上記課題を解決する手法を提案する．

Optimization of Matrix Operation in Privacy Preserving Cross Tabulation by GPU

Akira NAGAI†　　Dai IKARASHI†　　Tatsushi MATSUBAYASHI‡
Go YAMAMOTO†

†NTT Information Sharing Platform Laboratories, NTT Corporation
3-9-11, Midoricho, Musashino-shi, Tokyo 180-8585, Japan
‡NTT Communication Science Laboratories, NTT Corporation
2-4 Hikaridai Seika-cho Soraku-gun, Kyoto, 619-0237, Japan

†{ nagai.akira, ikarashi.dai, yamamoto.go }@lab.ntt.co.jp
‡tatsushi@cslab.kecl.ntt.co.jp

Abstract　In recent years, the researches of privacy-preserving cross tabulation, which can aggregate data with preserving privacy is becoming active. As a technique to realize this tabulation, Reconstruction method is known. However, this technique has a memory efficiency problem when a DB has many attributes. As solutions to that problem, some methods are conceivable, but in contrast, time efficiency becomes problem with these methods. In this paper, we propose a method to solve these problems using GPU, which is a parallel processing device.

1　はじめに

元来 GPU は 3D グラフィックスの演算・表示を行うための専用ハードウェアであった．ところが，GPU の演算能力が向上するに従ってその演算能力を非グラフィックス用途に用いる研究が行われるようになった [14]．これは，GPU が CPU と同程度の価格ながら数十倍の並列処理性能をもつデバイスだからである．そこで，GPU 供給元最大手の一つである nVIDIA 社は，GPU の並列処理性能を 3D グラフィックスだけでなく汎用演算にも適用する GPGPU(General Purpose GPU) のための開発プラットフォームとして「CUDA」(Compute Unified Device Architecture) を 2007 年から提供し始めた．CUDA は C 言語と同じ書式でアルゴリズムの記述が可能であるため，アルゴリズム実装実験における試行錯誤が容易である．特に，CUDA を用いた実装実験は GPU 上の高速化手法の研究に好適であり，多くの研究がなされている [8]〜[12]．例えば [8]，[11] では，共通鍵暗号 AES(Advanced Encryption Standard) の GPU 上での高速化手

法を提案し，CUDA による実装を報告している．

一方で，近年個人情報保護法や情報漏洩の問題の顕在化により，プライバシーに関する情報の保護が重要視されてきている．これに伴いデータマイニングやアンケート集計においてプライバシーを保護しながらも結果を得るような，**プライバシー保護データマイニング**の研究が注目されている ([1], [2] 等)．プライバシー保護データマイニングでは，エンドユーザから提供される情報は，データベース管理者に対しても保護されるため，情報提供者はもちろんのこと，よりスムーズに情報提供を受けられる情報取得側から見ても利益が大きい．

プライバシー保護データマイニングで利用される一般的なデータマイニング手法としては，決定木分類器，相関ルール抽出などが挙げられるが，本稿では**クロス集計**のみを扱う．なぜならクロス集計は直接的にアンケート集計で使われるほか，決定木分類器 [6, pp.21-29]，相関ルール抽出 [6, pp.41-48] 等多くのデータマイニング手法の中で使用される基本的かつ重要な演算だからである．

クロス集計を行う際に，プライバシー保護を実現する方法として，暗号や秘密分散に基づいたセキュア関数計算 ([2] 等) とデータを**撹乱**し**再構築**を行う再構築法 ([1]，[7]) が存在する．本稿では，セキュア関数計算よりも計算効率が良いと考えられてる再構築法を採用する．

ただし，再構築法は，その計算にメモリを大量に消費することが知られている [4]，[7]．例えば，何らかのクロス集計を行うとき，集計する対象物が 2 個と 20 個のときでは，メモリ消費量は 100 倍も変わる．結果，これまでの再構築法手法ではメモリ不足による問題を抱えていた．そこで本稿では，メモリの使用量を抑える手法を提案する．このとき，従来手法よりも計算量が増加するが GPU を用いることでその計算力を補う．

以降，第二章で GPU アーキテクチャの概要を述べ，３章でクロス集計における再構築法のアルゴリズムと問題点を説明し，４章で GPU を適用した再構築法の実装手法を提案し，５章で実験結果，６章でまとめる．

2　nVIDIA GPU アーキテクチャ

本章では，nVIDIA の GPU アーキテクチャの概略を説明し CPU のそれとは大きく異なることを見る．以後，GPU と言えば nVIDIA の GPU を指すこととする．

2.1　GPU 上のタスク処理

GPU は複数個のマルチプロセッサから構成される．さらに，1 つのマルチプロセッサにつき 8 個のコア[1] で構成される．例えば，TeslaC1060 は 30 個のマルチプロセッサを持ち合計 240 コアとなる．そして，処理単位をスレッドと呼び，スレッド数が並列数に相当する．プログラマはスレッド数を任意に決めることができ，例えば，スレッド数を 128 に設定した場合 128 個のコアが同じ処理を同時に行うことになる．TeslaC1060 の場合，コア数が 240 個のため 1 度に同時実行できる処理数は 240 になる．しかし実際のアーキテクチャは，図 1 より 8 コアは常に同じ処理を行う．

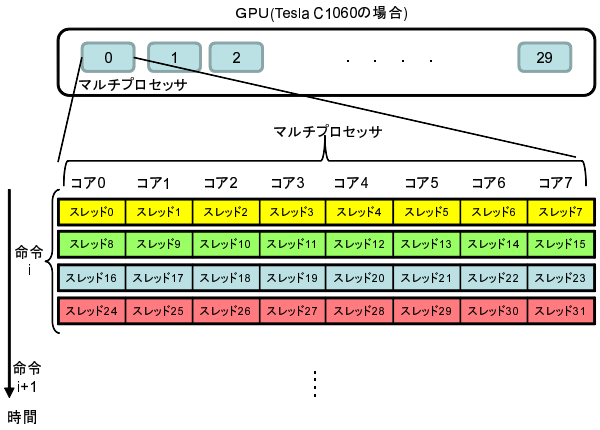

図 1: スレッドの実行状況

また図 1 より同じ命令を 4 サイクルに渉り実行する構造になっている．これは，処理を行うコアの周波数に比べ命令を供給するプロセッサの周波数が 4 倍近く低いためである．従って，各マルチプロセッサごとに 32 スレッド以上のスレッド数を設定しないと並列化による高速化効果が少なくなるという特徴を持つ．

2.2　GPU のメモリ

GPU のメモリは 5 種類 (グローバル，共有，コンスタント，テクスチャ，ローカル) から構成されており，それぞれ異なった特徴を持っている．テクスチャメモリは表示用であり，ローカルメモリはプログラマからは利用できないため，本研究では，グローバルメモリ，共有メモリ，コンスタントメモリの利用を検討した．本稿では，グローバルメモリと共有メモリの利用による高速化手法を次章以降で説明するためここでは，グローバルメモリと共有メモリについて簡単に説明する．

- グローバルメモリ
  CPU から転送されるデータを格納するメモリであり，CPU に演算結果のデータを渡すときもこのメモリに格納する必要が

[1] ここでコアとは，命令を実行可能なユニットを指す．

ある．容量は多い (Tesla C1060 では 4GB) が，アクセス速度が非常に遅く 400〜600 サイクルを要するため，なるべくアクセスを避けることが重要になる．

- 共有メモリ
  マルチプロセッサ内のコアで共有されるメモリであり，他のコアと同じデータを扱う場合などに利用される．GPU は，コア同士でレジスタを共有できないためにこのメモリが用意されている．容量は少ない (TeslaC1060 ではマルチプロセッサごとに 16KB) が，アクセス速度が非常に速くレジスタとほぼ同じ速度でアクセスが可能である．

2.3 CPU と GPU の高速実装の違い

2.1,2.2 から GPU は 1 つの処理を複数のコアで処理させる実装には適しておらず，多数のデータに対して同一の処理を行う実装に適している．言いかえるとコアごとに別々の処理を行わせていけない．なぜなら，1 つの処理に複数のコアを使用すると，共有メモリの利用によるコア同士のデータやりとり時や，あるコアのみが実行する命令がある場合などにアイドル状態のコアが発生してしまい，パフォーマンスを低下させてしまうからである．

3 再構築法

本章では再構築法についてそのアルゴリズムと問題点を見る．

3.1 準備

前提となる用語を定義する．

3.1.1 テーブル T

本稿で扱うテーブル T とは次のようなものである．

テーブルは情報提供者からのデータの集まりである．各情報提供者からのデータは**レコード**と呼ばれ, 各レコードはいくつかの予め定められた項目に対する値から成り立つ．この項目のことを**属性**と呼ぶ．

表 1 に簡単なテーブルの例を示した．表 1(a) における各行, “10 代, 男性” 等が各レコードに対応している．また “年代”, “性別” はそれぞれ属性で “10 代”, “男性”.

表 1: (a) テーブルの例，(b) クロス集計の例．

ID	年代	性別
001	10 代	男性
002	30 代	女性
003	20 代	女性
004	40 代	男性
⋮	⋮	⋮

(a)

年代	人数	
	男性	女性
10 代	4	3
20 代	4	1
30 代	5	7
40 代	8	2
⋮	⋮	⋮

(b)

3.1.2 クロス集計

クロス集計とはテーブルの複数の属性に着目し, 着目したすべての属性に関して値が等しいようなレコードを集計する集計法である．

表 1 にクロス集計の簡単な例を示した．表中の “人数” の列の値がここでは集計値であり, 例えば “年代” と “性別” に着目したクロス集計値のひとつ, “10 代” で “男性” であるレコードの数は “4” である．

この “10 代” で “男性” であることのような, 複数の属性に対する値をひとつの値とみなしたものをクロス値または集計値と呼ぶこととする．

従って，クロス集計は要素の組み合わせと一致するレコードの総和をとったものを成分値とする大きさが各属性の要素数を掛け合わせた数からなる横ベクトルとして考えることができる．

テーブル T に対するクロス集計を $\boldsymbol{x}$ とする．上記の場合，$\boldsymbol{x} = (4, 3, 4, 1, 5, 7, 8, 2, \ldots)$ となる．

3.2 再構築法概要

再構築法は，次の 2 つの処理から成る．情報提供者がレコードを提供する際に提供情報の値を変更する処理（撹乱と呼ぶ）と，撹乱されたデータをから統計データを取り出す処理（再構築と呼ぶ）の 2 つである．

撹乱には各情報提供者は自分のデータを確率的にランダムに変化させるランダム置換 [3] と意図的にノイズを付加する手法 [1] が存在する．特に本稿では，属性値に対して適用範囲が広い前者のランダム置換を採用する．また，これらの操作には非可逆性が成り立つため，一度撹乱するとその後本来のデータを復元することは困難である．これによって，情報提供者のプライバシー情報を保護することが可能になる．

一方，再構築には逆行列の計算する手法やベイズ推定等が [3] で提案されている．一般的に撹乱と再構築にどの手法を用いても撹乱が非可逆操作であるため，再構築で得られた推定値と真の統計値との間には乖離が生じる．文献 [3] によって，属性値の数が増えるとこの乖離値が大

きくなることが示されているため本稿では，ベイズ手法のみを採用する．

一般的に，撹乱の際に多くのデータを変更すればするほどプライバシー保護度合を上げることができる一方で，マイニング精度は低下してしまう関係が存在する．

3.3 撹乱処理

撹乱処理ではテーブル T における各属性の属性値を属性内の他の属性値に**維持確率**の下で変換する．維持確率を行列表現したものを**遷移確率行列**と呼ぶ．

3.3.1 遷移確率行列による撹乱

属性 a の遷移確率行列 A^a の (k,l) 成分を次のように定義する．

$$A^a_{kl} = \begin{cases} \rho_a + \frac{(1-\rho_a)}{M_a} & k = l \text{ のとき} \\ \frac{(1-\rho_a)}{M_a} & k \neq l \text{ のとき} \end{cases}$$

ただし，M_a は属性の要素数，ρ_a は属性 a における**維持確率**と呼ばれるパラメータであり，撹乱前の属性値を確率 ρ_a で維持する．維持しないときは，属性値から一様ランダムに選ばれる．**例** $M_a = 2, \rho_a = 0.6$，のときの遷移確率行列は次の通り．

$$A^a = \begin{pmatrix} 0.8 & 0.2 \\ 0.2 & 0.8 \end{pmatrix}$$

これは，今テーブル T が属性として ”性別 ”を持っているとし，確率 0.8 で属性値 ”男性 ”という値を男性のままに維持し，確率 0.2 で男性が女性に変わることを意味する．属性値が女性の場合も同様に定義される．このように各属性値から各属性値への変換確率を表したものが遷移確率行列であり，遷移確率行列に基づいてテーブル T から T' に変換する．

また属性が増えた場合も同様に遷移確率行列を定義できる [7]．今 2 つの属性を A^{a_0}, A^{a_1} とすると，遷移確率行列は行列のサイズが $M_{a_0}M_{a_1} \times M_{a_0}M_{a_1}$ の正方行列である．

$$\begin{pmatrix} A^{a_0}_{11}A^{a_1} & \cdots & A^{a_0}_{1M_{a_0}}A^{a_1} \\ \vdots & \ddots & \vdots \\ A^{a_0}_{M_{a_0}1}A^{a_1} & \cdots & A^{a_0}_{M_{a_0}M_{a_0}}A^{a_1} \end{pmatrix} \quad (1)$$

$(M_{a_1}k + i, M_{a_1}l + j)$ 成分である $A^{a_0}_{kl}A^{a_1}_{ij}$ は属性 A^{a_0} の要素 k から要素 l に遷移し，かつ A^{a_1} 要素 i から要素 j に遷移するときの確率を表している．つまり，上記の行列は遷移しうる全ての確率を表現している．また，この行列は行列 A^{a_0} と A^{a_1} のクロネッカー積そのものである．よって，遷移確率行列 A は次のようにクロネッカー積を用いて表現できる．

$$A = A^{a_{n-1}} \otimes A^{a_{n-2}} \otimes \cdots \otimes A^{a_0}$$

3.4 再構築処理

撹乱後のテーブル T' に対するクロス集計を $\boldsymbol{y}$ とする．本節では，前節で与えた撹乱されたクロス集計値 $\boldsymbol{y}$ から真のクロス集計 $\boldsymbol{x}$ を再構築し推定するアルゴリズムを与える．

3.4.1 反復ベイズ手法

反復ベイズ手法はベイズの定理に基づく手法で，$\boldsymbol{x}$ を推定する手続きは Algorithm 1 のようになる．

Algorithm 1 Reconstruction

$\boldsymbol{x}^0 \leftarrow \boldsymbol{y}$
$i \leftarrow 0$
repeat
  $\boldsymbol{x}^{i+1} \leftarrow \boldsymbol{x}^i \cdot (A(\boldsymbol{y}/(\boldsymbol{x}^i A))^t)^t$
  $i \leftarrow i + 1$.
until $| \boldsymbol{x}^{i+1} - \boldsymbol{x}^i |_{L1} \leq \epsilon$
return $\boldsymbol{x}^i$

ただし t は転置，$\cdot$ と / は成分ごとの乗算と除算を表す．また，$| \boldsymbol{x}^{i+1} - \boldsymbol{x}^i|_{L1}$ は $\boldsymbol{x}^{i+1}$ と $\boldsymbol{x}^i$ の間の $L1$ 距離と呼ばれ，各成分の差の絶対値の総和として表される．ϵ は事前に定める定数であり終了条件である．ϵ が小さいほど繰り返し回数が増え計算量は増すが精度は向上する．

3.5 再構築法の問題点

3.3.1 より属性の要素数が 2 倍，3 倍になるとその遷移確率行列のサイズは 4 倍，9 倍と膨れあがる．例えば，3 つの属性に対し属性の要素数が 100 個,100 個,10 個の場合，遷移確率行列のサイズは $100,000 \times 100,000$ となり保持に 40GB のメモリを要する．文献 [7] では，3 つの属性の要素数がそれぞれ 18,18,18 の場合に [4] の提案手法ではメモリを確保できなかったことを報告している．従って，属性や属性の要素数が増え，遷移確率行列が巨大化するとメモリ不足の問題が発生する．これに対し遷移確率行列を SSD(Solid State Drive) や HDD(Hard Disk Drive) に保存し，呼び出す手法は例えば，SSD のデータ転送速度が 1GB/s 程度としても遷移確率行列の属性の数が 18 個,18 個,18 個の場合反復回数が 10 回以上であったことと繰り返し 1

回につき遷移確率行列を 2 回呼び出すことから 20sec 以上かかることが推測できる．文献 [7] では上記 18 個,18 個,18 個の場合反復ベイズ手法で約 14sec が報告されており，外部記憶装置に遷移確率を保持する手法は計算時間の点から現実的な解決策とは言えない．

4 提案手法

本章では前章の問題に対し，クロネッカー積を計算する前の遷移確率行列を保持し，それらから撹乱処理時に用いる遷移確率行列を計算毎に GPU を用いて生成する手法を提案する．つまり，Algorithm 1 の $\boldsymbol{x}^i A$ における A の各成分を $A^{a_0}, A^{a_1}, \cdots$ から計算し $\boldsymbol{x}$ の成分と積和演算を行っていく手法である．従って，これまで以上に計算量は増加するが，提案手法によりメモリにおける制約を取り除くことが可能となり，結果，より多くの属性や属性の数の場合に保護クロス集計を実施できる．このとき，CPU 実装の場合計算量の増加によって計算時間が問題となるが，GPU が CPU よりも行列計算に有効であることが [16] などからも明らかであり，計算時間を飛躍的に短縮できることが期待できる．

特に本稿では，遷移確率行列の定義に応じて 2 つの GPU 実装手法を提案する．提案手法 1 はクロネッカー積を含む計算に対して汎用的に利用でき，提案手法 2 は各属性の遷移確率行列が 2 値からなる場合に特化した手法である．

今簡単のため，2 属性 A^{a_0}, A^{a_1} に限定して説明する．ただし，属性が 3 つ以上になっても同様の実装が可能である．

4.1 提案手法 1

$A^{a_0} \otimes A^{a_1}$ 行列の第 1 列を見ると (1) より，$\vec{\alpha}_1 = \begin{pmatrix} A^{a_1}_{11} \\ \vdots \\ A^{a_1}_{M_{a_1}1} \end{pmatrix}$ とすると $\begin{pmatrix} A^{a_0}_{11}\vec{\alpha}_1 \\ \vdots \\ A^{a_0}_{M_{a_0}1}\vec{\alpha}_1 \end{pmatrix}$ となる．従って，設定したスレッド数に合わせ $A^{a_0}_{i1}$ と $\vec{\alpha}_1$ を共有メモリに格納する．例えば，$M_{a_0} = 100, M_{a_1}$=100, スレッド数=500 に設定した場合，$A^{a_0}_{11}, \cdots, A^{a_0}_{51}$ と $\vec{\alpha}_1$ を共有メモリに格納し，500 スレッドで $\begin{pmatrix} A^{a_0}_{11}\vec{\alpha}_1 \\ \vdots \\ A^{a_0}_{51}\vec{\alpha}_1 \end{pmatrix}$ を計算する．さらに $\boldsymbol{x}^i$ の 500 成分と乗算をとる．つまり $\cdot$ を成分ごとの乗算とすると $\begin{pmatrix} \boldsymbol{x}^i_1 \\ \vdots \\ \boldsymbol{x}^i_{500} \end{pmatrix} \cdot \begin{pmatrix} A^{a_0}_{11}\vec{\alpha}_1 \\ \vdots \\ A^{a_0}_{51}\vec{\alpha}_1 \end{pmatrix}$ を 500 スレッドによって計算し，この成分の和を並列リダクション [15] を用いて高速に求める．このとき共有メモリの使用量は結果用を格納する分を含め，5Kbyte 内に納まっている．これを適切なベクトルの値に変更し 20 回繰り返すことで，$\boldsymbol{x}^i A$ の 1 つの成分を計算できる．各マルチプロセッサが列ごとに同処理を行うことで，並列に計算を行う．これによって，GPU のメモリに転送し保持する行列は各属性の遷移確率行列でよく，例えば属性の要素数が 100 個,100 個のときでも，約 80Kbyte の消費に抑えられる．一般的に，提案手法では遷移確立行列に対するグローバルメモリ消費量は従来法の $4 \cdot M^2_{a_0} \cdot M^2_{a_1} \cdot M^2_{a_2} \cdot \cdots$ に対して $4 \cdot (M^2_{a_0} + M^2_{a_1} + M^2_{a_2} + \cdots$ となる．

4.2 提案手法 2

もう一つの提案手法は，遷移確率行列の定義から各属性の遷移確率行列が $\rho_a + \frac{(1-\rho_a)}{M_a}, \frac{(1-\rho_a)}{M_a}$ の 2 値から構成されていることに着目する．すると各遷移確率行列の全成分を保持する必要はなく，この 2 値のみを保持すればいいことがわかる．

提案手法 1 と同様に $A^{a_0} \otimes A^{a_1}$ 行列の第 1 列目を見ると，$\begin{pmatrix} A^{a_0}_{11}\vec{\alpha}_1 \\ A^{a_0}_{21}\vec{\alpha}_1 \\ A^{a_0}_{31}\vec{\alpha}_1 \\ \vdots \\ A^{a_0}_{M_{a_0}1}\vec{\alpha}_1 \end{pmatrix} = \begin{pmatrix} A^{a_0}_{11}\vec{\alpha}_1 \\ A^{a_0}_{21}\vec{\alpha}_1 \\ A^{a_0}_{21}\vec{\alpha}_1 \\ \vdots \\ A^{a_0}_{21}\vec{\alpha}_1 \end{pmatrix}$

$\vec{\alpha}_1$ もまた $\begin{pmatrix} A^{a_1}_{11} \\ A^{a_1}_{21} \\ A^{a_1}_{21} \\ \vdots \\ A^{a_1}_{21} \end{pmatrix}$ となる．

従って， $A^{a_0}_{21}\vec{\alpha}_1$ と $A^{a_0}_{21}\vec{\alpha}_1$ で $\boldsymbol{x}^i A$ の計算が可能となる．これによって，上記 1 の提案手法に比べ，共有メモリやレジスタの使用量をより軽減することが可能となり，さらに計算量も提案手法 1 より小さくなる．また，実行ブロックの増加による並列度の向上が期待できる．例えば，属性が 2 つの場合保持すべきは取り得る確率の 4 値を保持すればよく，グローバルメモリ消費量はわずか 20Byte となる．また共有メモリの使用も提案手法の半分以下に抑えることが可能である．提案手法 2 の一般的なグローバルメモリ使用量は，4*2*(属性の数) となる．

5 実験結果

本章では前章で示した提案手法 1,2 を GPU 実装し，遷移確率行列を維持する従来手法と提案手法 1 の CPU 実装と比較する．コンパイラは CUDA2.3，マシン環境は GPU:GeForce GTX 285，CPU:Core 2 Duo 3.16GHz，OS:CentOS5.2 である．維持確率は 0.6, クロス値は乱数を用いたため反復回数をこれまでの経験からの平均で

表 2: 速度評価 (s)

	100 個,100 個	100 個,1000 個
従来手法 (CPU)	16.322	不可
提案手法 1(CPU)	26.277	600 以上
提案手法 1(GPU)	0.461	42.561
提案手法 2(GPU)	0.231	21.109

ある 12 回，属性数は 2，要素数は 100 個,100 個のときと 100 個,1000 個のときでシミュレーションを行った．そのとき 5 回の計測の平均値の結果を表 2 に示す．これまで 100000×100000 の遷移確率行列を保持することができないため，従来手法では計算できなかったが GPU 実装による提案手法を適用することで計算が可能となった．

6 おわりに

6.1 結論

本稿では, 再構築法によるプライバシー保護クロス集計のこれまで問題となっていたメモリの使用量に関する制約を取り除く手法を提案した．このとき，問題となる計算量増加に対して GPU 実装を用いることで計算力を補い本提案手法が CPU 実装よりも有効であることを示した．

6.2 今後の課題

他の改善手法としてコンスタントメモリの利用による高速化も考えられ，同様の実験をしたところ 100 個,100 個の場合で 0.451(s) とコンスタントメモリも使用したほうが速いが，数% の高速化であるためコンスタントメモリの利用に当たりさらなる検討が必要である．他にもクロネッカー積をある程度計算したり，いくつに分解したときが最適なのかなどの検討は今後の課題である．再構築法の再構築処理は GPU アーキテクチャに適しており，今後さらなる高速化が十分期待できる．

参考文献

[1] R. Agrawal and R. Srikant. Privacy-preserving data mining. *Proc. of the 2000 ACM SIGMOD Intl. Conf. on Management of Data,* 2000.

[2] Y. Lindell and B. Pinkas. Privacy Preserving Data Mining. *CRYPTO, Vol. 1880 of Lecture Notes in Computer Science*, Springer, pp. 36-54, 2000.

[3] R. Agrawal, R. Srikant and D. Thomas. Privacy Preserving OLAP. *SIGMOD Conference ACM*, pp. 251-262, 2005.

[4] 高見澤秀久, 有次正義. プライバシーを保護するカウント演算の多値属性分類への適用. *DEWS2007*, 2007.

[5] C. Blake and C. Merz. UCI repository of machine learning databases, 1998. `http://archive.ics.uci.edu/ml/`

[6] 元田浩, 津本周作, 山口高平, 沼尾正行. データマイニングの基礎. オーム社, 2006.

[7] 五十嵐大，千田浩司，高橋克己．多値属性に適用可能な効率的プライバシー保護クロス集計．CSS2008，2008.

[8] Svetlin A. Manavski,“ CUDA COMPATIBLE GPU AS AN EFFICIENT HARDWARE ACCELERATOR FOR AES CRYPTOGRAPHY ”, 2007 IEEE International Conference on Signal Processing and Communications (ICSPC 2007), 2007.

[9] S. Fleissner“ GPU-Accelerated Montogomery Exponentiation ”, ICCS 2007,LNCS 4487, pp. 213-220, Springer-Verlag, 2007.

[10] R. Szerwinski and T. Guneysu “ Exploiting the Power of GPUs for Asymmetric Cryptography ”, CHES 2008, LNCS 5154, pp. 79-99, 2008.

[11] N. Pilkington, B. Irwin“ A CANONICAL IMPLEMENTATION OF THE ADVANCED ENCRYPTION STANDARD ON THE GRAPHICS PROCESSING UNIT ”, ISSA 2008.

[12] Daniel J. Bernstein, Tien-Ren Chen, Chen-Mou Cheng, Tanja Lange, and Bo-Yin Yang, “ ECM on Graphics Card ”, Proc of Eurocrypt 2009, 2009.

[13] Matt Pharr 「GPU Gems 2」 ボーンデジタル 2005.

[14] Hubert Nguyen 「GPU Gems 3」 ボーンデジタル 2008.

[15] CUDA テクニカルトレーニング Vol 2:CUDA ケーススタディ

[16] CUDA での高速計算，`http://www.easize.jp/download/Workshop20071003.pdf` (2009/9/1 版)