Vol.2010-MPS-80 No.6
2010/9/28

CUDA に適用したループ再構築手法の評価

小 郷 絢 子[†1]　高 田 雅 美[†1]　城　和 貴[†1]

本稿では，既存のプログラムを高速化するために，GPGPU(General Purpose Computing on Graphic Processing Unit) のためのループ再構築手法を適用し，評価する．GPGPU で用いる GPU(Graphic Processing Unit) として，tesla C1060 を採用する．tesla C1060 では GPGPU 用開発環境である CUDA が使用可能である．

本稿において，ループ再構築手法例としてストリップマイニングとループ交換を取り上げる．これらのループ再構築手法に対してサンプル・プログラムを適用した際の性能評価を行う．その結果，ストリップマイニングでは分割数の変化によってループアンローリングが適用され，実行時間に影響がでることが分かった．一方ループ交換では，データへのアクセス順序を考慮することによって高速化が可能であることが判明した．これらの結果を踏まえ，実プログラムにおける性能を評価するために，k-means 法にストリップマイニングを，メディアンフィルタにループ交換を適用する．

Evaluation of Loop Transformation methods with CUDA

Junko Kogou,[†1] Masami Takata[†1] and Kazuki Joe[†1]

In this paper, we improve and evaluate loop transformation methods in GPGPU (General Purpose Computing on Graphic Processing Unit) for speed-up of existing programs. To adopt GPGPU, we use tesla C1060, which is known as a GPU (Graphic Processing Unit). In tesla C1060, CUDA, which is one of development environments for GPGPU, can be used.

In this paper, we target strip mining and loop interchange, which are loop transformation methods. To evaluate these loop transformation methods, we validate the performance of these loop transformation methods using simple sample programs. As the result in strip mining, execution time is effected by loop unrolling, which is changed at division number. On the other hand, in loop interchange, through access order to data is attentioned, speed-up can be done. To evaluate the performance in existing programs, we adopt strip mining to k-means and loop interchange to median filter in the light of the results.

1. はじめに

近年，遺伝子発現解析，天体シミュレーション，画像処理など，各種研究分野で膨大なデータを扱うことが増えてきている．そのような膨大なデータの処理に対して，一般的な計算機で処理をする場合長い時間がかかる．そこで，結果を短時間で出す為に高速演算処理を行うことが必要である．そのための解決策として，数台の高性能計算機を並列に用いるクラスタコンピューティングや，Cell などのマルチコアプロセッサを用いた高性能なハードウェアが用いられている．しかしこれらは高性能であるが故に非常に高価であるため，一般に普及することはなく一部の研究者のみしか使用できない．

本研究では，汎用性のある高速化手法を提案する．高速化手法には主に次の 3 点が挙げられる．1 つ目は，アルゴリズムの開発である．最適化するにあたって最も効果的であるが，新しいアルゴリズムの開発は短時間でできるものではなく，汎用性も低い．また，各アルゴリズムごとに行う必要があるので，本研究の対象外とする．2 つ目は，ハードウェアの改善である．このための方法として画像処理に用いる GPU（Graphic Processing Unit）を演算に使用する GPGPU（General Purpose Computation on GPU）の利用が盛んである．GPU はゲーム業界の技術速度の向上にともない，高性能化と低価格化がすすんでいる．本研究では GPU として tesla C1060 を使用する．Tesla C1060 とは NVIDIA 社が販売している GPGPU 用のグラフィックボードである．GPGPU の魅力として演算性能が高い，価格が安い，一般のパソコンにも装着可能という 3 点が挙げられる．3 つ目として，プログラムの最適化がある．従来のものはコンパイル時に自動的にループ再構築理論により，ループ部分の書き換えによる高速化手法を用いてある程度の最適化が成されている．ただし，この最適化はすべての計算において最適とは言えず，場合によっては手動でより効率的な最適化を探し，行うことが求められる．よって今回はその中でもストリップマイニングとループ交換に着目する．提案する 2 つのループ再構築手法を既存のアルゴリズムである k-means 法とメディアンフィルタに適用し，性能を評価する．

以下，2 章ではループ再構築理論について述べる．3 章ではループ再構築手法を tesla に適用する際の留意点を述べる．4 章では実際に tesla でのループ再構築理論の性能についての実験結果とその考察について．5 章では 4 章での結果を踏まえて k-means とメディアン

†1 奈良女子大学
Nara women's University

Vol.2010-MPS-80 No.6
2010/9/28

```
do i = 1 ,n
   a[i] = a[i] + c
end do
        ↓
lastsi = (n/m)*m
   do si = 1 , lastsi ,m
        do l = si , si+m-1
        a[i] = a[i] + c
        end do
   end do
   do i = lastsi+ 1 ,n
        a[i] = a[i] + c
   end do
```

プログラム例1

```
do i = 1 ,n
        a[i+k] = a[i] + c
end do
          ↓
do si = 1, n, k
        do l = si, si+k-1
        a[i+k] = a[i] + c
        end do
end do
```

プログラム例2

図 1 ストリップマイニング

フィルタに適応した実例を述べる．

2. ループ再構築手法

最適化には，実行時間を短くする最適化と所要記憶容量を小さくする最適化とがある．ループ再構築手法とは，プログラムのループ部分を書き換えて実行時間を短くする最適化の 1 種である．プログラムにおいてループを指定する部分がある場合は，そのループの内側が最も多く実行される．また，既存プログラムの多くがループを使用しており，ループ部分の実行がそのプログラム全体の実行時間を占めることが多い．そのため，ループ部分を最適化することが高速化に大きく影響を与えると考えられる．そのための方法をまとめたものがループ再構築手法と呼ばれている．代表的な例としては，ストリップマイニング，ループ交換等がある．これらの適応条件は変換によって依存関係が壊れないことである [1].

ストリップマイニングとは，1 重ループを 2 重ループに変換するものである．図 1 はストリップマイニングの例である．図 1 のプログラム例 1 は，1 重ループを長さ m のループに細分して，それを (n/m) 回繰り返すように書き換えたものである．この書き換えによる効果として細分化したループでの依存がなくなり並列処理可能となることが挙げられる．図 1 のプログラム例 2 のように長さ k のループに細分することによって，内側のループには依存がなくなるため，並列実行可能となる．このような変換はサイクル縮小と呼ばれる．適応条件は，演算の順序が変更されないので特にない．

ループ交換とは，多重の完全入れ子ループで入れ子の順序を交換するものである．効果としてはアクセスの連続性による，並列化可能性と内側ループの局所性が増す可能性が挙げられる．図 2 は例として 2 重ループの外側ループと内側ループを入れ替えたものである．この交換によって，図 2 のような b[i] = b[i] + a[i, j] を並列化できる．また，図 2 の a[i, j] へのアクセスに関してループ交換を行うと，図 3 のような配列格納順序の場合，配列 a[i, j] へのアクセスが連続的になる．適応条件に関しては，全ての距離ベクトルの要素が正かゼロならば，ループの順序をどのように入れ替えてもよい [1].

```
do i = 1, n
        do j = 1, m
        b[i] = b[i] + a[i][j]
        end do
end do
          ↓

do j = 1, m
        do i = 1, n
        b[i] = b[i] + a[i][j]
        end do
end do
```

図 2 ループ交換

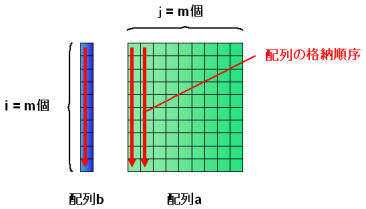


図 3 ループ交換のアクセス連続性

3. 高速化の際の注意点

GPGPU 用のハードウェアでループ再構築理論を行うにあたり，プログラムを効率よく並列計算するものに書き換えることが必要となる．その際考慮すべきはデータの振り分け方・データ処理の方法・メモリの使用法等である．

まず，データの振り分け方で重要なのがスレッド数，ブロック数とワープの関係である．

Tesla C1060 には 30 個の SM（=Streaming Multi Processor）と 1 つの SM あたり 8 つの SP（=Streaming Processor） が搭載されている．そこで，これらを余りなく使用するためにブロック数は 30（=SM 数）の倍数，スレッド数は 8（=SP 数）の倍数にするのが妥当と思われる．しかし CUDA ではスレッドが 32 ずつ処理される為，スレッド数に関しては 32 の倍数にすることが高速化に有効と考えられる．

次にデータ処理の方法について説明する．CUDA では，ワープという特殊なスレッド処理単位が存在するため，これを考慮してプログラムを組む必要がある．ワープとは 32 スレッドずつをまとめたものである．ワープ内の全てのスレッドが同一分岐先，つまりパスに分岐する場合は全スレッドが分岐先のみを実行するが，異なるパスに分岐する場合は全スレッドが両方のパスを実行してしまう．よって実行時間は両方の分岐を実行した時と同様になってしまうため，これは性能低下の原因となりうる．ループ再構築理論ではループ条件による分岐が不可欠であるため，ワープ単位で同じ命令を実行させるようにすることが重要であると考えられる．

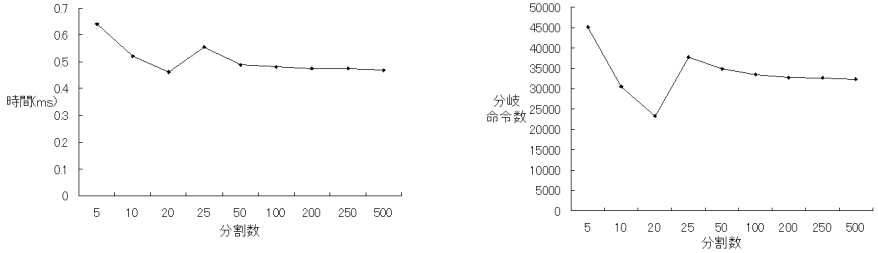
メモリの使用法については，アクセスの低速なグローバルメモリから高速なシェアードメモリにデータをコピーし，演算時にはシェアードメモリからデータを読み込むことによって高速化を図る．

4. tesla C1060 におけるループ再構築手法の評価

本実験では，ストリップマイニングとループ交換について実験を行い，ループ再構築手法による実行時間の影響を分析する．実験には，CPU は Intel core i7 を，GPU は Tesla C1060 を使用する．4.1 節ではストリップマイニングの実験結果と考察について述べる．4.2 節ではループ交換の実験結果と考察について述べる．

4.1 ストリップマイニング

GPGPU においては並列演算可能な部分はできる限り一度に並列演算を行う方が高速に演算できる．しかし一度に並列演算を行うことのできない場合が考えられる．それはシェアードメモリを使用する場合である．シェアードメモリは標準で使用するグローバルメモリと比べて高速アクセスができるので，高速化の為の使用は必須である．しかし，容量が 16KB と少ない為大きなデータを扱う場合，全てのデータをシェアードメモリに載せて演算することはできない．よって分割してデータ転送と演算をする必要がある．この時の分割方法にストリップマイニングの考え方を応用し，分割数の変化によって実行時間にどのような変化が現れるかを実験する．CUDA におけるプログラムで，GPU 実行部分であるカーネル部分の演算には，並列演算が行われる部分と並列演算を行わない通常演算の部分がある．その為，今回の実験ではそれぞれに対してストリップマイニングを行い，分割数の関係を調べる．ここで，並列演算部とは演算部を CUDA により並列演算することをいう．一方，通常演算部とは CUDA での並列演算を用いずに演算する部分を言う．今回のストリップマイニングの実験では，要素数 1000 の 1 次元配列に対して分割数を変化させて実験を行う．例えば分割数 5 でストリップマイニングを行う時，シェアードメモリへの 5 要素の移動と演算を 200 回繰り返すことのなる．つまり，この場合外側ループのイタレーション数は 200，内側ループのイタレーション数が 5 の演算となる．ここで，通常演算部の実験では内側ループをループのまま実行するが，並列演算部での実験では内側ループ部分を並列演算で処理することとする．



実行時間と分割数　　分岐命令数と分割数

図 4 ストリップマイニング　通常演算

まず，分割して転送したデータに対して通常演算を用いる場合の実験を行う．図 4 の左図は，分割数を変えてシェアードメモリに転送し，単純な演算を行う際の実行時間と分割数の関係を表したものである．分割数が 20 までは徐々に高速になっているが，25 で遅くなりその後はまた徐々に高速になっている様子が見られる．図 4 の右図はこのときの分岐命令数と分割数を表したものである．左図と同じ傾向が現れている．分岐命令数とはアセンブラにおける分岐命令 bra の総数である．ループ文や if 文の条件判定時等に使用される命令で，CUDA において bra は高速化を妨げる大きな要因の 1 つである．右図に見られるような分岐命令数の変動はコンパイラによる，ループアンローリングが原因であると考えられる．ループアンローリングとはループ再構築手法の 1 種で，ループを展開し条件判定部分を無くすことによって分岐命令を減らす手法である．CUDA のアセンブラである ptx ファ

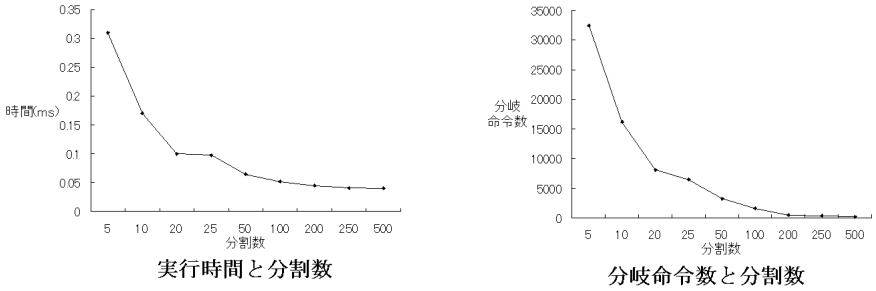


図 5 ストリップマイニング　並列演算

イルにおいて，分割数 20 までに対してはループアンローリングが行われていることが確認できる．このため，分割数 20 で分岐命令数が少なく，実行時間が減少しているのである．よって，通常演算部分のストリップマイニングに関しては，分割数がループアンローリングされる値の中でも，最大の値を取る時，分岐命令数が減り，最も高速に演算が成されることがわかる．

次に，分割して転送したデータに対して並列演算を用いる場合の実験を行う．図 5 の左図は分割数と実行時間の関係を表したものである．分割数が増えるほど実行時間が減少している様子がわかる．CPU におけるストリップマイニングでは外側ループの実行回数が減るだけ内側ループ実行回数が増えるので，総ループ実行回数は同じになる．一方 CUDA を用いた並列処理の実験では，内側ループの分割数分のデータが並列演算処理される．これにより，外側ループの実行回数が増えても内側ループ実行回数は増えない．その結果総ループ回数が減る．これが高速化つながるものと考えられる．

図 5 の右図は，分岐命令数と分割数の関係を表した図である．左図と同じ傾向が読み取れる．ループ回数が減ると，ループ文の条件判定部の実行回数が減るため，図のような傾向になると考えられる．つまり，ループ回数を減らすことによって分岐命令数が減少するため，並列演算可能な範囲内で分割数をなるべく多くすることが高速化につながる．

以上より，GPGPU においてストリップマイニングを適用する場合，分割後の計算が通常計算となる場合は CUDA の最適化オプションでループアンローリングされる最大数を分割数とすべきである．また，分割後の計算が並列化可能な場合は，分割数を可能な限り大きくすべきである．

表 1 ループ交換 実験 1

	× LI	○ LI
実行時間 (ms)	1.067136	0.166304
分岐命令数	10916	618
総命令数	43984	2272

表 2 ループ交換 実験 2

	× LI	○ LI
実行時間 (ms)	1.067232	0.164992
分岐命令数	10916	618
総命令数	43986	2272

4.2 ループ交換

ループ交換の実験として，並列演算部分の実験を行う．まず，本実験では 2 次元配列を対象とする．最初に，図 2 と図 3 で示される，サンプルプログラムを用いて実験を行う．表 1 は，この時の実験結果である．× LI とはループ交換が行われない時を，○ LI とはループ交換を行われる時を意味する．この結果より，ループ交換を行った方が高速であることがわかる．これはループ交換によって内側ループが並列演算可能となるためである．並列化されているため，これにより内側ループの比較や演算など繰り返し行われている演算の分岐命令数と命令数が減少している．

図 2 のループ交換を行った後のサンプルプログラムに対して並列化する場合，i に関して分割される．そのため，各スレッドは 2 次元配列に連続アクセスできない．一般的に配列へのアクセス順序が格納順序と一致する場合の方が速い．そこで，図 2 の配列 a[i][j] を a[j][i] に書き換えて，配列データへの連続アクセスの影響を調べる．表 2 はこの時の実験結果である．ループ交換後に分岐命令数や総命令数が減少し，実行時間が短くなっている点は表 1 と同じである．表 1 との若干の違いはループ交換前の総命令数のみである．これは表 2 の実験ではループ交換前の 2 次元配列へのアクセスが不連続であるため，余分な演算が増えているためと思われる．ゆえに，スレッドによるアクセスの連続性は，考慮しなくてもよいと考えられる．

5. 実　　例

本章では既存プログラムにループ再構築手法を適用する．5.1 節ではストリップマイニングを k-means に適応し，その結果と考察について述べる．5.2 節ではループ交換をメディア

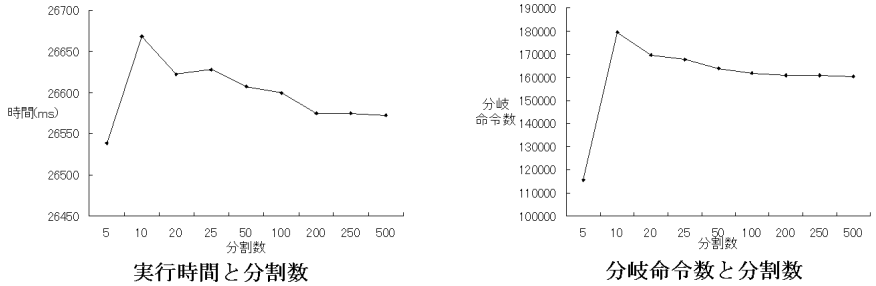


図 6 ストリップマイニングの k-means への適用

ンフィルタに適応し，結果と考察について述べる．

5.1 ストリップマイニングの k-means への適用

k-means とは多量のデータをいくつかのグループに分ける手法の 1 つである．例えば 2 次元上の点をクラスタリングする場合，まず初めに分けたいグループの数だけクラスタと呼ばれる点を適当に作成する．クラスタリングの対象となるデータはサンプルと呼ばれる．各サンプルは，最も距離が近いクラスタのグループに所属させる．サンプルが全て，いずれかのクラスタグループに所属すると，クラスタグループ内で中心となる点を新たなクラスタとする．この手順を繰り返すことにより，クラスタリングが行われる．

本適用では，サンプルデータを並列に演算し，クラスタとの距離は通常演算させることで効率よく行わせる．全てのサンプルに対して，最短距離となるクラスタの割り出しを行う部分を GPU で行う．各サンプルと最も近いクラスタの番号と距離を保存するための配列なども同一のシェアードメモリに保存する．サンプルは，1 ブロックあたり並列演算可能な 512 個ずつシェアードメモリに割り当てる．ここで必要なブロック数はサンプル数を 512 で割った値となる．1 つのサンプルは全てのクラスタとの演算を行うため，クラスタは各ブロック内のシェアードメモリに載せる必要がある．しかしながら，シェアードメモリは 16KB しかないため，クラスタ数の多いレコメンデーションシステムなどに関しては，全てのクラスタを保持することはできない．よってストリップマイニングを用いて分割するごとにシェアードメモリへの転送を繰り返す．そのため，最外側のループではクラスタを分割数で割った回数だけ実行される．

k-means に対するストリップマイニングの影響を確認するために，サンプル数 29696，クラスタ数 1000 のデータを用いて実験を行う．なお配列を確保する際，int 型整数が 30000 程度しか表現されないため，本実験で使用するサンプルの 1 次元配列の要素数はその範囲内で，最大スレッド数 512 の倍数になる最大値を用いている．図 6 は，この実験の結果である．図 6 の横軸の分割数とは，クラスタのうちシェアードメモリに転送する数を意味する．左図はその実行時間と分割数の関係，右図は分岐命令数と分割数の関係を示す．図 6 の左図より，分割数 5 の時に高速で分割数 10 で低速になり，その後は分割数の増加とともに実行時間が減少していることがわかる．クラスタ数が 1000 個以外の場合でも同様の傾向である．この原因として，分岐命令数が挙げられる．図 6 の右図は，分割数と分岐命令数の関係を表したグラフである．左図と同じ傾向が現れていることがわかる．

k-means のプログラムにおいて ptx ファイルを確認すると分岐数は通常以下に示す (1)(2) の式で表すことができると考えられる．

$$(1 + 3 * P + 4000 * P + P + 1) * 2 * 16 \quad (1)$$

$$(1 + 4 + P + 4000 + P + 1) * 2 * 16 \quad (2)$$

ここにおいて P は (1000/分割数) である．cuda profiler による分岐命令のカウントは，ワープ単位かつ 0 番目の SM における bra 数をカウントする．本今回のプログラムでは 0 番目 SM には 2 ブロックが割り当てられており，1 ブロックには 16 ワープが存在すると考えられる．式 (1)(2) はそれぞれ本実験のプログラムの ptx ファイルから読み取れる 1 ワープあたりの分岐命令数に，2 ブロックと 16 ワープの 2*16 をかけ、理論上想定できる分岐命令数の式を求めたものである．(1) は実験に用いたプログラムで求められる基本的な分岐命令数を求める式である．プログラム中では，1 サンプルと全てのクラスタとの距離を求め，最も距離の短いクラスタを割り出すために距離の比較をするために分岐命令が用いられる．1 ワープ内では、1+1 回の分岐命令と、3+4000+1 回の P 回ループ内の分岐命令が見られる．これにより，P の値が少ない程，つまり通常は分割数が多いほど分岐数が減少することとなる．しかし分割数 5 の場合は例外である．なぜならば，分割数 5 の場合はコンパイラによって，アセンブラでループアンローリングが行われているためである．そのため，式 (2) で表されるように，他の分割数とは異なる分岐数となる．

以上より k-means のプログラムでは分割数の変化により分岐命令数が変化し，実行時間に影響を及ぼすことがわかる．並列演算を用いない通常演算部分では，ループアンローリングが行われる分割数 5 で最も分岐命令数が減少し高速となり，ループアンローリングされない分割数では分割数が増える程分岐命令数が減少し高速となる．このとき k-means の実行時間は，既存の CPU 用プログラムでは 181029ms，ストリップマイニングを適用した

Vol.2010-MPS-80 No.6
2010/9/28

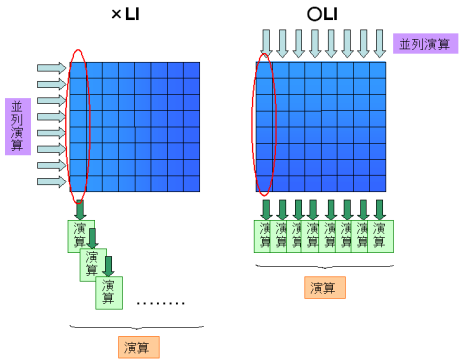


図 7 メディアンフィルタにおけるループ交換の概念

表 3 メディアンフィルタとループ交換

	× LI	○ LI
実行時間 (ms)	99122.46	37172.28
分岐命令数	3116984808	182580384
総命令数	4294967295	1585071183

GPU 用プログラムでは 26538.47ms となる．

5.2 ループ交換のメディアンフィルタへの適応

メディアンフィルタは画像処理手法の 1 つである．画像データから，マスクと呼ばれる決まった大きさの 2 次元データを取り出し，そのデータの中央値をマスク内の全てのデータに置き換えるという作業を行う．今回使用した既存プログラムでは，マスク内中央値を求めるのに，2 次元配列を 1 列ずつ 1 次元に書き込み最大値を探す．まず最大値に 0 を代入することを配列要素数の半分回数繰り返す．その後配列要素値の中で最大値を求めると，元の配列の中央値となる．その際，2 次元配列データを行または列ごとに 1 次元配列に移し変えて最大値の探索を行い，次にソートで求めた行または列のそれぞれの中央値を格納した 1 次元配列の最大値の探索を行う．本実験では，この 2 次元配列から 1 次元配列を作成し，ソートをする部分に対してループ交換の影響を調べる．

図 7 は，データアクセス方向と計算順序の概念図である．× LI，つまりループ交換前は 2 次元配列のデータを 1 行ずつ 1 次元配列に移動しているため，全ての要素の移動後にしか最大値の探索を行えない．一方○ LI，つまりループ交換後は 1 列ずつデータを読み込むため，各列ごとに並列実行可能である．既存のプログラムの中には× LI で開発されているものもあるため，ループ交換が必要となる．表 3 は，図 7 のそれぞれに対する実行結果である．この際，2 次元配列のサイズは 49*49 で 2401 である．この表から，ループ交換を適用することによって並列演算できる部分が増え，総命令数や分岐命令数が減少していることがわかる．これにより，実行時間が減少している．このときメディアンフィルタの実行時間は，既存の CPU 用プログラムでは 181029.20ms，ループ交換を適用した GPU プログラムでは 37172.28ms でとなる．

6. ま と め

本研究では，GPGPU 用ハードウェア Tesla C1060 における，ループ再構築理論の適用を行った．本稿では，2 種類のループ再構築手法を試みた．1 つ目のストリップマイニングは，高速アクセス可能なシェアードメモリに一度に全てのデータが保持できない時に行う必要がある．ここで分割数によって分岐命令数が変化し，分岐命令数が少ない時実行時間は早くなる．通常演算部分ではコンパイラによりループアンローリングが行われる．そのため，ループアンローリングの数が最大値を取るとき，実行時間は早くなると考えられる．ただし，その値はプログラムに順ずると思われる．並列演算部分では分割数を並列演算可能な範囲でなるべく大きくすることで分岐命令数が減少する．この結果を踏まえ，既存プログラムである k-means にストリップマイニングを適用すると，ループアンローリングの最大数 5 で分岐命令数が減り，高速に演算できる．2 つ目のループ交換については，並列演算が行えるようになり，高速化できる．また，一般的には配列データへのアクセスが連続可能である場合高速となるが，GPU の場合，連続アクセスの有無と実行時間の関係を見出すことはできなかった．ループ交換を既存プログラムであるメディアンフィルタに適用すると，サンプルプログラムと同様に，並列演算可能になり，高速化が行えた．

GPGPU では並列演算が可能であり，コンパイラによる最適化が行われる条件や方法などが独特のものになると考えられる．よって，今後他のループ再構築手法についても tesla C1060 特有の高速化の効果を評価していくべきである．

参 考 文 献

1) 中田育男：コンパイラの構成と最適化，朝倉書店 (1999).