Vol.2011-HPC-130 No.15
2011/7/27

メモリアクセラレータで強化した GPU の CG 法による評価

小郷絢子[†]　田邊 昇[††]
高田雅美[†]　城 和貴[†]

本報告では，係数行列や解ベクトルが GPU 上に載りきらないほど大きな連立一次方程式を共役勾配法(CG 法)で解く際に，メモリアクセラレータの利用を提案する．提案アクセラレータは GDDR5 ポートなどに接続され，デバイスメモリの厳しい容量制約を緩和するとともに，Gather 機能によりキャッシュや GDDR 系メモリが苦手とする間接アクセスを連続アクセス化する．本報告では，フロリダ大学の疎行列コレクションを用いて提案方式の性能評価を行った．その結果，テクスチャキャッシュが効くような小さな行列でも，単体性能はテクスチャキャッシュを用いる既存手法の 1.05～2.01 倍に向上した．従来手法は行列サイズを大きくした時，GPU 内キャッシュのヒット率が低下し，性能低下する．解ベクトルがデバイスメモリ容量を超えると PCI express を通過する通信により，さらなる性能低下が予想される．それに対し，本手法はそれらの心配が無い．

Performance Evaluation of a GPU Enhanced with a Memory Accelerator by Using CG-method

Junko Kogou[†]　Noboru Tanabe[††]
Masami Takata[†]　and　Kazuki Joe[†]

In this paper, we propose the use of memory accelerator to solve systems of liner equation, which coefficient matrices and solution vector unable to be had on board by CG method. In the case of existing method, when the size of matrix is increase, performance decrease because of hit probability on GPU cache decrease. We predict that performance decrease because of access of exceed access, the case solution vector is over device memory capacity. In contrast, there is no risk, thanks to this method. Proposed accelerator is connected to such as GDDR5 port, it ease severe capacity limit, and make indirect access which is unsuitable for cache and GDDR memory into direct access using gather function. In this paper, we evaluate the performance of proposed strategy with University of Florida Sparse Matrix Collection. The result showed from the 1.05 to the 2.01 times acceleration over the case of existing performance record with the texture cache, even if small matrix which has effect of texture cache.

1. はじめに

ベクトル型スーパーコンピュータの演算能力は COTS の CPU や GPU で代替可能なケースが多い．GPU の演算能力は既に 1 TFLOPS を超えており，それを生かした GPGPU 研究の成功例[1]は数多く報告されている．一方，キャッシュや GPU の統合メモリアクセスでは救済できない大容量メモリに対するランダムアクセスを主体にするアプリケーションでは，必ずしも COTS がベクトル型スーパーコンピュータを代替できない．GPU 基板上のデバイスメモリの容量は現状では最大でも 6GB であり，それを超える大規模データを処理する場合、バースト転送しか効率的に実行できない通信経路(PCI express)がボトルネックになっていた．

上記の問題を解決するため，筆者らは Scatter/Gather 機能を有する拡張大容量機能メモリと GPU を GDDR5 ポートや PCI express によって接続するヘテロジニアスシステムを提案している．

以下､本論文では第 2 章で CG 法の概要と GPU 上での CG 法の高速化について述べ，第 3 章で提案システムのアーキテクチャを述べる．第 4 章では提案システムにおける CG 法の動作を示す．第 5 章では性能評価を示し，最後に第 6 章でまとめる．

2. CG 法

本研究ではアプリケーションとして共役勾配法(CG 法)を検討対象とする．本章では CG 法の概要と、GPU 上での CG 法の高速化について述べる．

CG 法は大規模な疎行列を計数行列とした連立一次方程式や固有値求解において最もよく使われる計算方法であるクリロフ部分空間法の代表的な解法となっている．ナイーブな CG 法は正定値対称行列向けの反復法の一種で，数学的には方程式の未知数の個数を上限とする有限の反復回数で収束する．ところが，実際の数値計算では浮動小数の精度不足によって，性質の悪い行列では収束しない．その収束性を改善するた

[†]奈良女子大学
Nara women's university
[††]株式会社　東芝
Toshiba corporation

めや，非対称行列を扱えるようにするために，通常，前処理が併用される．前処理としては不完全コレスキー分解(IC)や不完全 LU 分解(ILU)といった直接法から後天的な非零要素に伴う計算を省略した不完全な直接法が代表的である．最近では単精度浮動小数による CG 法を倍精度浮動小数による CG 法の前処理的に用いる混合精度型のアルゴリズムがメモリバンド幅削減を主目的に注目されている．本研究は混合精度型のアルゴリズムを用いることを想定し，その大半の計算時間を占めることになる単精度浮動小数による CG 法の実行時間について検討を行う．

CG 法の実行時間における最も重い処理は疎行列ベクトル積である．疎行列はメモリ容量や計算時間の節約の観点から CRS 形式や JDS 形式などの非零要素のみを集めた格納形式を用いて実行される．いずれの格納方式でも最内側ループには一次元配列(ベクトル)の間接参照（添字が配列になっている配列の参照）がある．ランダムに近い非零要素配置を有する行列を扱う場合，上記の間接参照がキャッシュのライン単位のアクセスや，GPU の Coalesced アクセスと相性が悪いランダムなアクセスを発生しがちである．その結果，疎行列ベクトル積はメモリバンド幅がボトルネックとなる．このため疎行列ベクトル積はベクトル型スーパーコンピュータ以外での効率的な処理が困難だった．

近年，GPU のメモリバンド幅が CPU のそれより大きいことに着目し，GPU で疎行列ベクトル積を高速化する研究が数多く試みられている．既に NVIDIA 社などからいくつかの疎行列ライブラリが公開されている．配列の格納形式を工夫する研究が多いが，列ベクトルアクセスの高速化手法としてはテクスチャキャッシュを利用する方法が一般的である．GPU のテクスチャメモリは GPU 側からは読み出し専用であるものの，再利用性のある配列である列ベクトルをホスト側から格納し，テクスチャメモリをアクセスするための専用の関数(tex1Dfetch や tex2D など)を用いて列ベクトルをアクセスできる．テクスチャメモリはテクスチャキャッシュ上にキャッシングされるため，疎行列の非零要素の配置によってはデバイスメモリへのアクセスが抑制される．本報告の後半ではテクスチャキャッシュの効果についても評価する．

近年の GPU(nVidia 社の場合は Fermi アーキテクチャ採用の GPU)ではテクスチャメモリのみならず，デバイスメモリ上の他のカテゴリーのメモリ領域(グローバルメモリ，ローカルメモリ)もキャッシング可能な汎用の L1 キャッシュ，L2 キャッシュが搭載されるようになってきた．これにより特別な関数を用いなくても列ベクトルはキャッシュを用いてデバイスメモリアクセスを抑制しながらアクセス可能になる．本報告の後半では GPU の汎用キャッシュの効果についても評価する．

疎行列計算のベンチマークにしばしば用いられるフロリダ大学の疎行列コレクションに登録されている程度の小さめな行列では，CPU や GPU においてもある程度のキャッシュヒット率を確保できる．しかし，GPU に搭載できるキャッシュの容量には限りがあり，Fermi アーキテクチャの C2060 では L2 キャッシュでも 768KB にすぎない．デバイスメモリサイズに近い行列や，それに対応した大きな列ベクトルに対しては十分なヒット率を確保できないと考えられる．頻繁にミスヒットし，ミスヒット時には 128 バイトのキャッシュライン中に 4 バイトまたは 8 バイトしか使用するデータが無いような非効率的なメモリアクセスを繰り返すようになると考えられる．この状態では全アクセス数の約 1/3 を占める列ベクトルアクセスの大半に対して単精度の場合に 32 倍，倍精度の場合に 16 倍に増加したメモリバンド幅を必要とする．この問題にどう対処するかの解決策が望まれる．

さらに大きな行列を扱うために複数ノードで並列処理する場合は，図 1 に示すように疎行列ベクトル積の処理は疎行列を構成する行ベクトル群と列ベクトルの積に分解できる．行間にはデータ依存関係が無いため，メモリ容量の制約に合わせ疎行列を行単位で GPU に分割することは基本的には容易である．

ただし，GPU における従来の実装においては列ベクトルの大きさとデバイスメモリの間の大小関係における制約が存在する．つまり，入力された列ベクトルは全て GPU がローカルにコピーを保持できないと，行ベクトルの非零要素の位置に対応する列ベクトルの要素を読み出す際に，一般的にはランダムでバースト長が短い GPU 間通信が発生してしまう．非零要素の配置パターンが一般には特定できないため，アプリケーションへの汎用性を保ったまま効率的にタイリングを行うことは困難である．

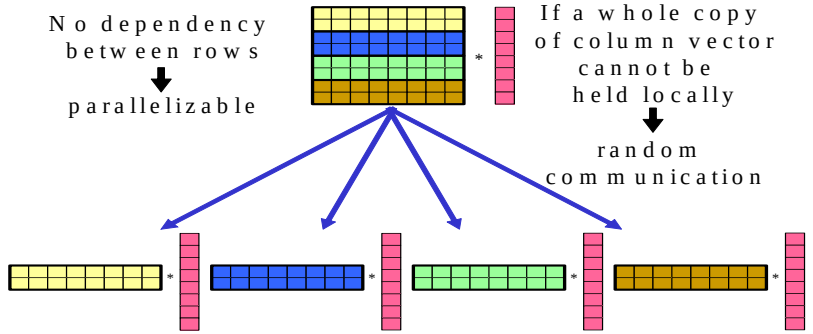

図 1　疎行列ベクトル積における並列性

本研究では疎行列そのもののみならず，疎行列に乗じられる密な列ベクトルすら 1 個の GPU のデバイスメモリに入りきらない大きな問題も対象とする．例えば 4GB のデバイスメモリがある GPU において列ベクトルと行ベクトルを半分ずつ使って格納した場合，倍精度浮動小数の要素数が 256M 個を超えるベクトルを扱うとランダムア

クセスが GPU 外部に溢れる．つまり 1000^3 以上の格子点を扱う大規模な行列ベクトル積を単純な GPU クラスタは現実的には並列処理することが困難である．この問題にどう対処するかの解決策が望まれる．

3. 提案システムアーキテクチャ

3.1 基本概念

図 2 に提案システムアーキテクチャの基本概念を示す．機能メモリと GPU 等のアクセラレータを組み合わせることにより、従来のシステムでは効率が悪かったメモリアクセスを効率化し、その結果として高い実効性能を得る方式を提案する。図 2 に機能メモリのインタフェースに PCI express 等の高バンド幅な標準 I/O を用いる場合の提案アーキテクチャの基本概念を示す． PCI express スイッチ等の共有アドレス空間上にデバイスをマップする機能を有する結合網を介してこれらを多数結合する．このような方式により，メモリ容量とメモリバンド幅と結合網バンド幅と演算能力のバランスを維持したスケーラビリティ向上，低消費電力化と低コスト化を実現することができる．

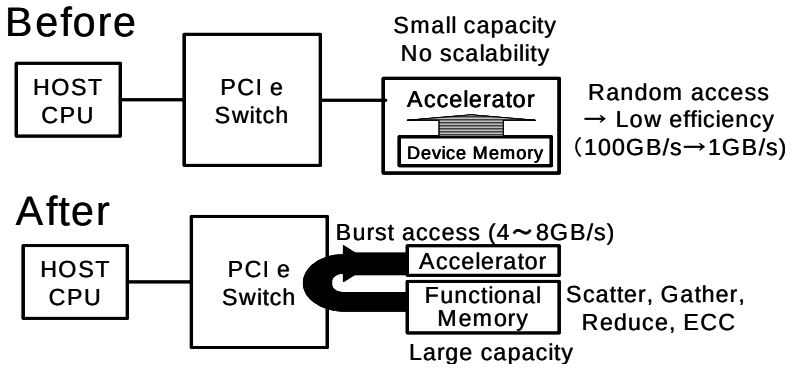

図 2　提案アーキテクチャの基本概念

機能メモリはアクセラレータの外付けデバイスとして，メモリ容量の厳しい制限を解消し，エラー訂正機能が付いた拡張メモリとして用いられる．さらに機能メモリはホストの主記憶と異なり，PCI express 等の標準 I/O を通過するデータ量を削減する機能や転送効率を向上させるための機能を有する．機能メモリの具体的な機能として代表的なものは，DIMMnet-3[12][13]に実装されている Scatter/Gather(分散/収集)機能である．

従来の GPU などのシステムではランダムアクセスバンド幅が 1GB/s 程度しか得られなかった．提案システムでは Scatter/Gather 機能によってランダムアクセスがバーストアクセスに変換される．PCI express を経由してデバイスメモリに転送する場合は 4～8GB/s のピークバンド幅に近いバンド幅が得られるようになる．

3.2 機能メモリのインタフェース

機能メモリのインタフェースとしては，上記の説明では一例として PCI express を用いるものについて説明した．他にも GPU ベンダー側での対応が必要になるものの GPU 向けには SLI (Scalable Line Interconnect)や GDDR5 デバイスメモリインタフェースなどの高バンド幅なインタフェースの利用が考えられる．高バンド幅なインタフェースは一般的にバースト長が長くないと本来の性能を発揮できないことが多い．しかし，機能メモリの分散/収集機能によりバースト長が飛躍的に伸び，転送効率の向上が期待できる．特に GDDR5 DRAM は 1 チップあたり現段階で 7Gbps ×32 ビット幅で 28GB/s が得られる．NVIDIA® Tesla™ C2050/2070 は GDDR5 ポートを 384 ビット分(32 ビット×12)に設置している．この GDDR5 ポートを将来の GPU 上で機能メモリ用に 1 チップ分増設または切換え可能に改良することで PCI express より高いバンド幅を機能メモリ側に提供することが可能であると考えられる．理論上，GDDR5 はさらに 4 倍の 28Gbps まで向上できるとされており，本用途への応用は有望と思われる．

3.3 処理の流れ

図3に基本概念に基づく機能メモリアクセス処理の流れを示す．

(1) 機能メモリ(例えば PCI express 子基板を実装した DIMMnet-3)へのコマンドキューは PCI 空間上にマップされる．よって，ホスト（またはアクセラレータ）はアクセスしたい機能メモリに割り当てられた上記のコマンドキューに対応するアドレスに所定フォーマットでコマンドを書き込む．
(2) 上記に応じて，上記書き込みトランザクションは実行され，アクセスする機能メモリのコマンドキューにコマンドが書き込まれる．
(3) 機能メモリはコマンドキューからコマンドを取り出して，記載された内容の機能(例えば遠隔リストベクトルロード：RVLL)を実行し，指定があれば応答データや完了フラグをコマンドに記述されたアドレスに書き込む．
(4) 上記に応じて，書き込みトランザクションは実行され，コマンドは終了する．
(5) ホスト（またはアクセラレータ）は十分に余裕のあるタイミングでコマンドを起動できない場合は必要に応じて上記完了フラグをポーリングする．もしコマンドが完了していれば，デバイスメモリ上に連続化かつアライメント調整されて格納済みのベクトルデータに対する後続の処理 (SIMD 演算ループ (Cell/B.E.,SpursEngine)やWarp (GPU))を実行する．

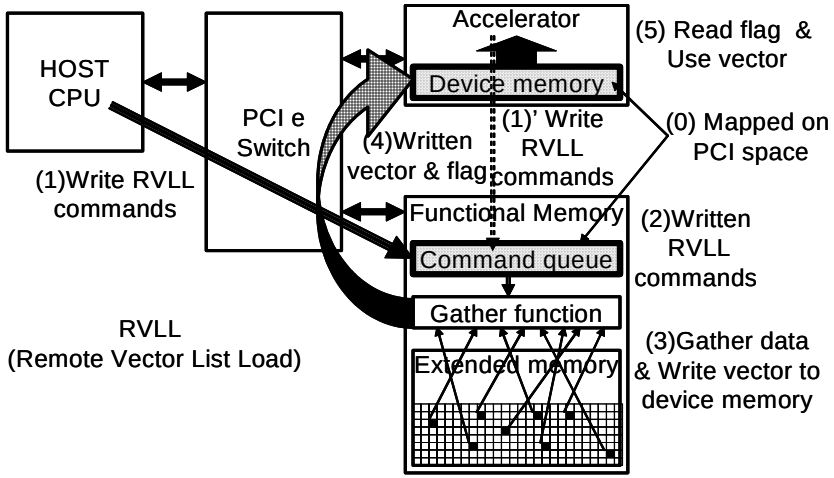

図 3. 機能メモリアクセス処理の流れ.

4. 提案システムにおける CG 法の動作

本章では，提案システムにおける CG 法の動作の流れについて論じる．本報告では混合精度型のアルゴリズムを用いることを想定し，その大半の計算時間を占めることになる単精度浮動小数による CG 法の動作について述べる．

4.1 基本方針

CG 法ではαやβを密ベクトルの内積演算が主体の計算によって生成し，その値を用いて列ベクトルを更新させながら,疎行列ベクトル積を繰り返すという流れになる．密ベクトルの内積部分の演算回数は連立方程式の未知数の個数(係数行列の行数)に比例し，疎行列ベクトル積の演算回数は係数行列の非零要素数に比例する．行列の非零要素の配置にもよるが，一行あたりの非零要素数が多い行列ほど疎行列ベクトル積の計算時間が支配的となり，一行あたりの非零要素数が少ない行列ほど密ベクトルの内積演算の比率が上がる．特に後者の場合には，密ベクトルの内積演算を GPU ではなくホスト側に行わせると，転送時間が増加することと併せて，性能上の問題がホスト側に移行する危険性があるため，極力 GPU 内部で全ての演算処理が完結するようにプログラミングすることが望ましい．

一方，提案システムを用いて疎行列ベクトル積を高速化する場合，結果の列ベクトルを機能メモリが書き戻す必要がある．複数の機能メモリを用いている並列システムの場合は，結果をマルチキャストする必要がある．ただし，この通信は機能メモリ間の結合網の作り方を工夫することで，機能メモリ数にほとんど依存しない時間でマルチキャストすることが可能であると考えられる．さらに，ストリーミングを適用することにより疎行列ベクトル積の計算処理と，結果ベクトルのマルチキャストをオーバーラップさせることも可能であると考えられる．

4.2 疎行列ベクトル積

提案システム向けの疎行列ベクトル積アルゴリズム[16][17]は我々の研究グループによって既に提案済みである．詳細は文献[16][17]を参照されたい．以下，その概要を示す．

4.2.1 前処理

前処理における行列の整形と転置の流れを図 4 に示す．なお，この前処理は提案システムを使わない場合もある程度の有効性が期待できる．本報告の後半では提案システムを使わない場合の本前処理の効果についても評価する．

(1) 配列を行方向に圧縮
(2) 適切な折り目を決め，折り目以上の非零要素がある行を複数行に分割の上，折り目まで 0 パディング
(3) 元の行数＋折畳み回数を下回らない 32 の倍数まで空行をパディング
(4) 転置し，CRS 形式と同様なインデックス配列を生成

4.2.2 提案システムによるプリロード処理

図 5 に機能メモリによるベクトルのプリロードの流れを示す.

機能メモリには前節(4)において転置した状態のインデックス配列を指定した間接ベクトルロード(Gather)コマンドを実行することで，必要なデータを機能メモリの buffer 上に Gather した上で，GPU のデバイスメモリ(グローバルメモリ)にバースト転送する. GPU 上では隣接スレッドがグローバルメモリ上の連続アドレスに並んだ適切なベクトルのデータをアクセスする形に処理が変換される.

なお，ストリーミングによって，機能メモリによるベクトルのプリロードと，カーネル処理を並行して実行させることにより，前者の転送時間の大半はカーネル実行時間に隠蔽されるものと考えられる．その実装と評価は今後の課題とする.

4.2.3 メインカーネル部

提案システムを用いて前節のようにプリロードを行う場合は GPU で実行されるカーネル部は，上記の前処理によって同じ長さの短い密ベクトルと密ベクトルの内

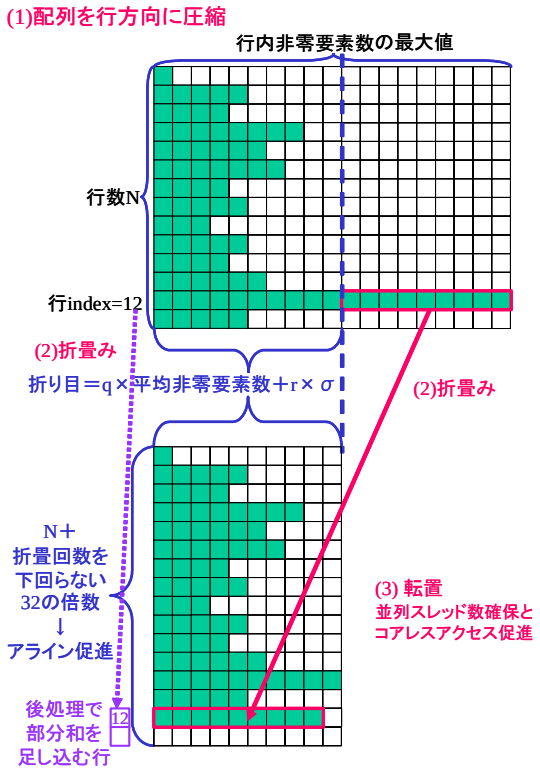

図 4.　前処理における行列の整形と転置の流れ.

積処理を多数のスレッドが実行する状態に置き換えられる．行列およびベクトルへのアクセスはアラインメントされた位置からのスレッド番号順に連続するグローバルメモリへの直接参照となり，全アクセスがコアレストアクセスとなる．

提案システムを用いない場合は，テクスチャメモリまたはキャッシングされるグローバルメモリに列ベクトルを格納し，前処理のインデックス配列を用いた間接参照により列ベクトルをアクセスしつつ, 密ベクトルとの内積計算を行う. この場合，キャッシュが有効なサイズや非零要素配置を持つ行列では，キャッシュの効果により列ベクトルのアクセスが高速化され，処理が高速化する．

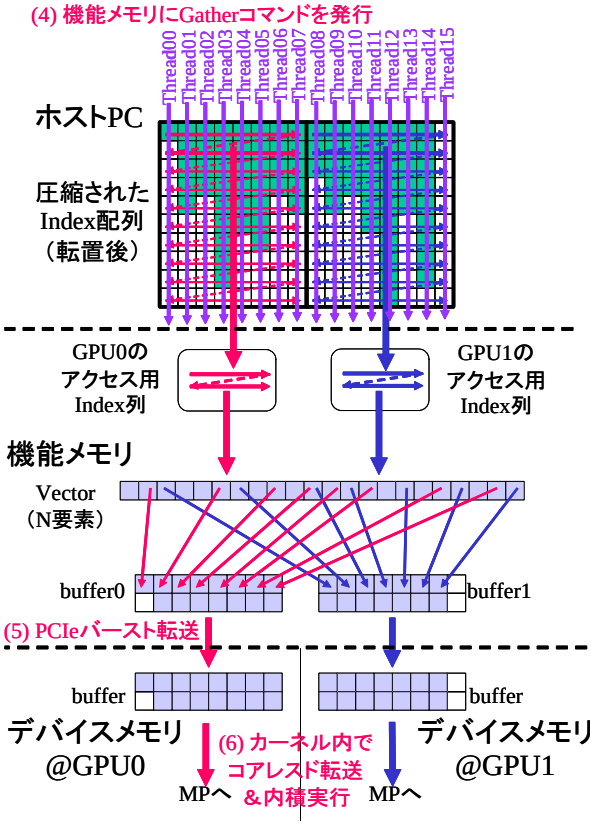

図 5.　機能メモリによるベクトルのプリロードの流れ.

4.2.4 後処理カーネル部

提案システムを用いる場合は各スレッドが累積したスカラ値からなる部分ベクトルを，出来上がったところから機能メモリに放送する．折り畳みが不要な行の結果を転送しつつ，折り畳んだ行については部分和を足しこんで，最終的な結果ベクトルの値を計算した上で，その結果も機能メモリに放送する．

提案システムを用いない場合は，上記の処理において機能メモリへの放送は不要である．テクスチャキャッシュを用いる場合は結果ベクトルをホスト側に転送し，ホスト側から再度新しい結果で更新する必要がある．

5. 評価

5.1 実験環境とテスト行列

今回の実験に用いた計算機環境を表 1(C1060 環境)および表 2(C2050 環境)に示す．また，実験に用いた行列を表 3 に示す．

表 1　測定環境(C1060 環境)

ホスト	Intel® Core(TM) i7 CPU 920 @ 2.67GHz　?GB
GPU	Nvidia Tesla C1060(MP 数 30)
デバイスメモリ	メモリバンド幅 103GB/s、 4GB
ホスト I/F	PCI express x16 Gen.2(最大バンド幅 8GB/s)
OS	Fedora10
CUDA	Cuda3.0

表 2　測定環境(C2050 環境)

CPU	Intel® Xeon(R) CPU X5670 @ 2.93GHz
GPU	Nvidia Tesla C2050(コア数 448)
デバイスメモリ	メモリバンド幅 144GB/s、3GB
ホスト I/F	PCI express x16 Gen.2(最大バンド幅 8GB/s)
OS	Red Hat Enterprise Linux Client release 5.5 (Tikanga)
CUDA	Cuda3.2

行列は University of Florida Sparse Matrix Collection[21]から抜粋した．これらは本研究が想定する「乗ずるベクトルが GPU のデバイスメモリに入りきらないほど大きい問題」ではないが，本評価ではそのような大きな問題を提案システム上の複数 GPU に分割して実行する場合に，各 GPU に分配されるデータが上記の行列集と同等の性質を保持していると仮定する．筆者らの研究グループでは先行研究[16][17]で疎行列ベクトル積の評価を行ったため，その結果と併せた考察を行うことを想定し，評価行列は文献[16][17]と同じものを用いた．

ここで用いられている行列のサイズでは最大でも乗ずるベクトルは 6.3MB というデバイスメモリ容量に比べると微々たるものである．よって，本来想定する状況よりもかなりキャッシュが効きやすい状況（他グループの先行研究に有利な状況設定）での評価であり，キャッシュを用いない提案方式には不利な状況設定での評価になる．

表 3　評価に用いた行列

行列名	行数	非零要素数			
		合計	行平均	行最大	標準偏差
Na5	5,832	155,731	26	185	35.71
msc10848	10,848	620,313	57	300	49.40
exdata_1	6,001	1,137,751	189	1501	390.27
G3_circuiit	1,585,478	4,623,152	2	4	2.18
thermal2	147,900	3,489,300	23	27	6.86
hood	220,542	5,494,489	24	51	13.31
F1	343,791	13,590,452	39	306	19.97
ldoor	952,203	23,737,339	24	49	12.90

5.2 評価プログラム

本章の評価に用いた CG 法のプログラムはカーネル部の列ベクトルアクセス手法が異なる以下の 3 種類である．いずれも 4 章で紹介した文献[16][17]の前処理を行ったものに対して疎行列ベクトル積を行うようなプログラムになっている．これらによってカーネル部の列ベクトルアクセス手法の違いのみがどのように処理速度に反映されるのかを知ることができる．

(1) テクスチャメモリ版

本プログラムは GPU のテクスチャメモリに列ベクトルを格納し，Tex2D 関数によってアクセスすることでテクスチャキャッシュの効果を利用するものである．性能の基準として用いるとともに，収束するまでの反復回数の採取も行い，後述する(3)の反復回数としてその値を用いる．

(2) 共有メモリ版

本プログラムは共有メモリを介してデバイスメモリ上の列ベクトルをアクセスするものである．Fermi(C2050 環境では上記のアクセスが汎用キャッシュ（L1 および L2 キャッシュ）によって加速される．

(3) 提案システム版

本プログラムは提案システムによってデバイスメモリ上に使用する前に整列された列ベクトルをアクセスして計算に用いるものである．ソースコード上は間接参照のかわりにアラインされた位置への直接参照によるバーストアクセスとなり，Coalesced アクセスとなる．文献[18][19]の技術で十分なメモリアクセスするープ

ットが得られるものと仮定する．

5.3 テクスチャキャッシュにおけるヒット率

C1060 環境における(1)のプログラムのテクスチャキャッシュのヒット率を測定した．測定にはプロファイラを用いた．CUDA3.0 においては tex_cache_hit および

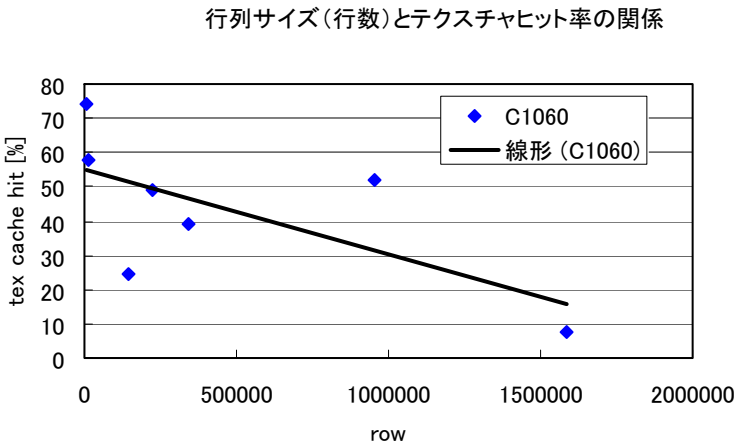

図 6. 行列サイズ(行数)とテクスチャキャッシュヒット率の関係.

tex_cache_miss という性能カウンタの値を計測することができる．

図 6 に行列サイズ(行数)とテクスチャキャッシュヒット率の関係を示す．行数が多くなると小さなテクスチャキャッシュから列ベクトルアクセスがはみ出すため，ヒット率が悪くなっていく傾向がわかる．線形近似を行った場合，急峻な傾斜で右下がりであり，行数が大きくなった時にこの勢いでヒット率が下がると今回の測定範囲以上の大きさの行列ではキャッシュの効果はほとんど期待できない．測定に用いた行列の中で最も行数が多い G3_circuit(行数 1,585,478)ではヒット率は 7.74%に過ぎず，この程度の大きさの行列でもテクスチャキャッシュでは扱いきれず溢れている状態といわざるを得ない．G3_circuit は 1 行あたりの非零要素が平均 2 と少ないため，ライン内に再利用されるデータがほとんど載っていないこともヒット率を低くする原因と考えられる．

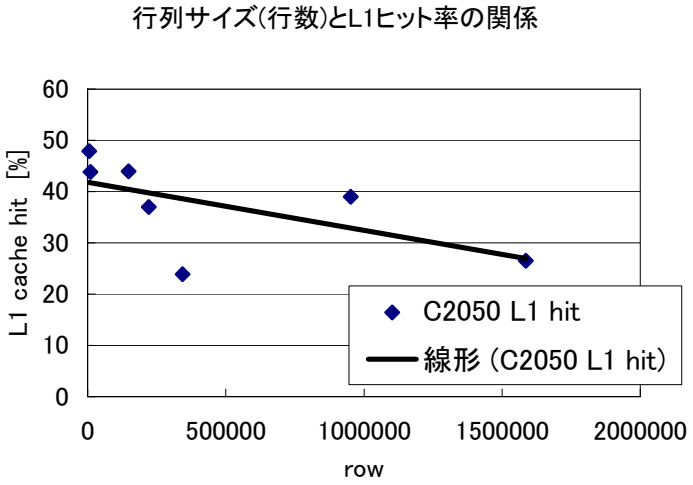

図 7. 行列サイズ(行数)と L1 キャッシュヒット率の関係.

5.3 汎用キャッシュにおけるヒット率

C2050 環境における(1)のプログラムの汎用キャッシュ（L1 キャッシュ）のヒット率を測定した．測定にはプロファイラを用いた．CUDA3.0 においては l1_global_load_hit および l1_global_load_miss という性能カウンタの値を計測することができる．

図 7 に行列サイズ(行数)と L1 キャッシュヒット率の関係を示す．キャッシュの Preference は L1 キャッシュが大き目(L1 が 48KB，共有メモリが 16KB)となる設定における結果である．行数が多くなるとテクスチャキャッシュの場合と同様に，ヒット率が悪くなっていく傾向がわかる．G3_circuit が 26.5%であり，テクスチャキャッシュを C1060 上で用いる 7.74%よりは改善している．注意深く図 6 と図 7 を観察すると F1 はテクスチャキャッシュを用いる場合はヒット率がさほど低くないが，汎用キャッシュを用いる場合は 23.9%とヒット率が下がる．この現象は汎用キャッシュの場合，再利用性が無い行列データまでキャッシュを経由してしまっており，非零要素総数が多い F1 ではヒット率が減少する結果になったと考えられる．この効果を予防

表 4　列ベクトルアクセス法の違いによるカーネル実行時間［ms］

	C1060			C2050	
	Texture	Shared	Proposed	Shared	Proposed
Na5	0.0822	0.1102	0.0367	0.0482	0.0396
msc1848	0.2052	0.3923	0.1192	0.1277	0.1131
G3_circuit	1.2805	1.5580	0.7497	0.7574	0.5825
thermal2	0.9421	1.4510	0.5178	0.4949	0.4402
hood	1.1757	2.3379	0.8118	0.8269	0.7395
F1	4.7696	7.4038	3.5963	2.8900	1.9807
ldoor	4.9873	10.3343	3.5675	3.5770	3.1879

するには，再利用性の無いデータのロードはキャッシュを無視して直接ロードするタイプの命令[22]が使用されるようにプログラムすることが望ましい．

5.4 列ベクトルアクセス法の違いによるカーネル実行時間

前述の 3 種類の評価プログラムのカーネル実行時間を表 4 に示す．実行時間の積算値を反復回数で割り算した平均値を示している．時間測定には CUDA Event を用いる方法で行った．表中の数値の全て単位はミリ秒である．現状のカーネルは疎行列ベクトル積の大半の計算を担っているが，行折り畳みの部分和の足し込みを行う後処理についてはカーネル外（ホストでの実行）となっている．

C1060 のテクスチャキャッシュを用いたもの(Texture)を 1.0 とした際の速度比を図 8 に示す．結果としては，全ての行列において提案方式が高速である．ただし，追加ハードウェアの効果はこれらの小さめの行列に対しては限定的であることがわかる．この結果を言い換えると，提案ハードウェアは小さい行列におけるテクスチャキャッシュや汎用キャッシュがある程度効いている状態の GPU のスループットと同等以上のスループットをキャッシュの効果に頼らずに置き換え可能であることがわかる．前節の実験より行列の巨大化はキャッシュの効果を減らすことと合わせると，今回の実験より大きな行列を用いると提案ハードウェアの有無による差は広がってくると考えられる．

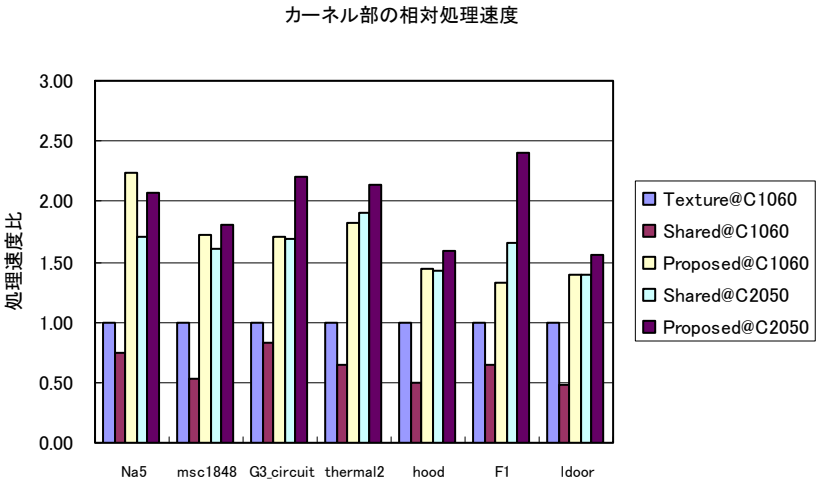

図 8.　カーネル部(疎行列ベクトル積)の相対処理速度

C2050 上では C1060 上であまり高速化しなかった Shared のプログラムがデバイスメモリバンド幅の向上と汎用キャッシュの効果によって C1060 上の提案方式なみに高速化した．この範囲の行列サイズでは L1 はミスヒットだが，まだ L2 ではヒットしていると考えられる．しかし L1 全体に比べて L2 の容量は極端に大きいわけではないので，L1 と同様な限界性は行列をさらに大きくしていくと顕在化すると考えられる．

なお，文献[]には F1 と ldoor の 2 つの行列には倍精度の場合の C2050 上での疎行列ベクトル積の JDS 形式等を用いた場合の処理時間が記載されている．提案システムを用いず前処理のみ用いている表 4 中の時間(C2050 の Shared)は，F1 の場合は文献[]の JDS の 1/4.1，ldoor の場合は ELL の 1/2.74 の時間である．倍精度と単精度の違いでバンド幅が半分で済むことにより 2 倍の性能差がでることを考慮しても，前処理アルゴリズム(Fold 法)だけでも従来方式より高速化が得られていると考えられる．

5.5 列ベクトルアクセス法の違いによる CG 法実行時間

全て対称行列だが正定値でない行列もあり，単精度浮動小数で実行しているため，

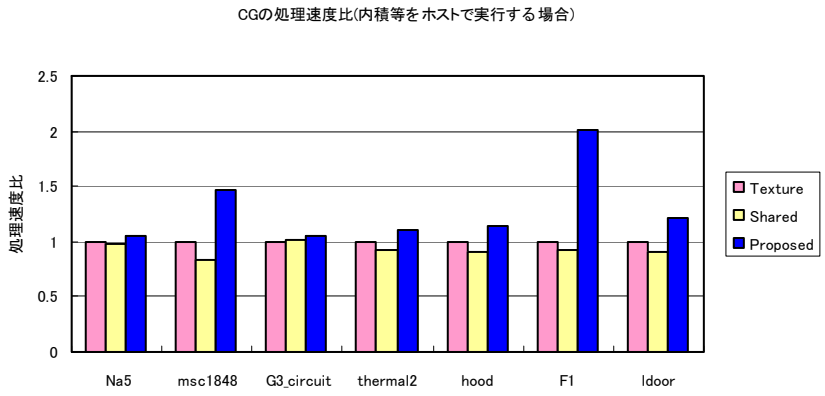

図 9. CG 全体の相対処理速度(内積等をホストで実行する場合)

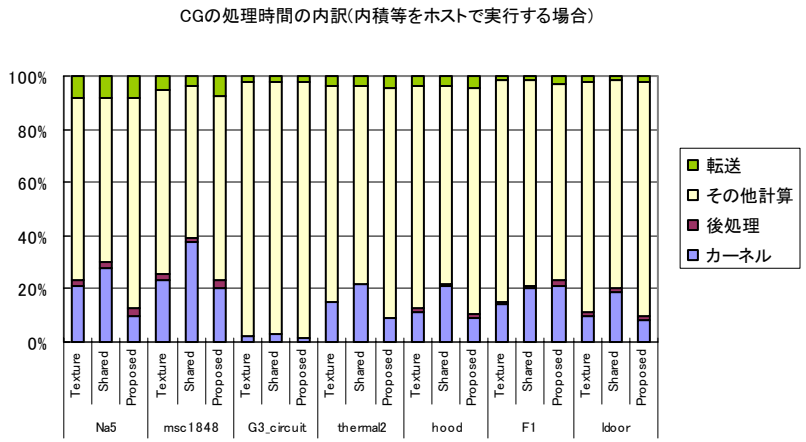

図 10. CG 全体の処理時間の内訳(内積等をホストで実行する場合)

反復回数は行列ごとに大きく異なり，きちんと収束していない場合もある．よって，収束した場合は収束するまでの反復回数と時間を測り，それ以外では誤った解への収束(誤差が規定以下にならない状態での振動)が始まるまでや，非数割り込みが発生するまでの反復回数と時間を記録した．提案手法の反復回数はそれらを用い，一反復あたりの実行時間を比較することができる．前述の 3 種類の評価プログラムの C1060 における CG 法全体の相対的処理速度を図 9 に示す．

結果としては，テクスチャキャッシュが効くような小さな行列でも，単体性能はテクスチャキャッシュを用いる既存手法の 1.05～2.01 倍に向上した．ただし，図 8 のカーネル部の速度比よりさらに穏やかな差となって観測された．これはカーネル部以外の処理をホスト上で実行していることが原因と予想される．現時点では最適化の途中であり，疎行列ベクトル積以外の部分はホスト上で実行させており，その弊害がここで現れていると考えた．そのことを確認するために，カーネル部以外の処理時間も分類して測定した．CG の処理時間の内訳を図 10 に示す．

図 10 においてホスト～GPU 間転送の時間や，折り畳んだ少数の行の部分和の足し込みを行う後処理はあまり大きな割合を占めておらず，その他の計算(疎行列ベクトル積以外の内積等)が大半を占めるようになっていることがわかる．

内積等の演算はベクトルサイズが方程式の未知数であり，G3_circuit のように 1 行あたりの平均非零要素数が 2 しかないような計算では疎行列ベクトル積と内積の演算量はあまり差が無い．このため，G3_circuit では疎行列ベクトル積が GPU 上で高速化された結果，ほとんどの時間がその他の処理になってしまったと考えられる．他の行列も程度の差はあるものの GPU 上での疎行列ベクトル積の高速化によって，ホスト CPU 上でのその他の処理が大きな割合を占めている．

ここで，密行列の内積は Segmented scan 法などの並列処理が可能な計算であり，Nvidia 社が GPU 用に提供する数値演算ライブラリである CUBLAS 等によって高速化が可能であると考えられる．本報告の〆切の段階ではその実装と評価を終えることができなかったが，口頭発表日までに CUBLAS 化を行った評価を完了させる予定である．

6. おわりに

本報告では，係数行列や解ベクトルが GPU 上に載りきらないほど大きな連立一次方程式を共役勾配法(CG 法)で解く際に，メモリアクセラレータの利用を提案した．提案アクセラレータは GDDR5 ポートなどに接続され，デバイスメモリの厳しい容量制約を緩和するとともに，Gather 機能によりキャッシュや GDDR 系メモリが苦手とする間接アクセスを連続アクセス化する．フロリダ大の疎行列コレクションを用いて提

案方式の性能評価を行った．その結果，テクスチャキャッシュが効くような小さな行列でも，単体性能はテクスチャキャッシュを用いる既存手法の 1.05〜2.01 倍に向上した．この加速率は内積計算のホストでの処理などの未実装部分による性能劣化により鈍った加速率であり，未実装部分を GPU 化することで更なる性能向上が期待できる．従来手法は行列サイズを大きくした時，GPU 内キャッシュのヒット率が低下し，性能低下することも確認した．解ベクトルがデバイスメモリ容量を超えると PCI express を通過する通信により，さらなる性能低下が予想される．それに対し，本手法はそれらの心配が無い．

今後の課題は密ベクトル内積の CUBLAS 化の実装と評価，機能メモリへの書き戻し時間の隠蔽を考慮したストリーミングの実装と評価，より大きな多くの行列を用いた L2 キャッシュの限界点の評価，多数の GPU と機能メモリを用いた並列システム上でのスケーラビリティの評価，機能メモリの設計と評価などがある．

謝辞 本研究の一部(DIMMnet-3 の開発)は総務省戦略的情報通信研究開発推進制度(SCOPE)の一環として行われたものである．

参考文献

1) Nvidia : "CUDA Zone", http://www.nvidia.co.jp/object/cuda_home_jp.html
2) N. Bell, M. Garland : "Eficient Sparse Matrix-Vector Multiplication on CUDA", NVIDIA Technical Report NVR-2008-004, Dec. 2008
3) M. M. Baskaran, R. Bordawekar : "Optimizing Sparse Matrix-Vector Multiplication on GPUs", IBM Research Report, RC24704, Apr. 2009
4) A. Cevahir, A. Nukada, S. Matsuoka : "An Efficient Conjugate Gradient Solver on Double Precision Multi-GPUSystems", Symposium on Advanced Computing Systems and Infrastructures (SACSIS2009), pp.353-360, May 2009
5) A. Monakov, A. Lokhmotov and A. Avetisyan :" Automatically Tuning Sparse Matrix-Vector Multiplication for GPU Architectures", 5th International Conference on High Performance Embedded Architectures and Compilers (HiPEAC 2010), Lecture Notes in Computer Science 5952, pp.111-125, 2010.
6) F. Vazquez, G. Ortega, J. J. Fernandez and E. M. Garzon : "Improving the Performance of the Sparse Matrix Vector Product with GPUs", 10th IEEE International Conference on Computer and Information Technology (CIT 2010), pp.1146-1151, 2010.
7) M. M. Dehnavi, D. M. Fernandez and D. Giannacopoulos : "Finite-Element Sparse Matrix Vector Multiplication on Graphic Processing Units", IEEE Trans. Magnetics 46(8):2982-2985, 2010.
8) 大島，櫻井，片桐，中島，黒田，猪貝，伊藤: " Segmented Scan 法の CUDA 向け最適化実装"，情報処理学会 HPC 研究会 Vol.2010-HPC-126 No.1, pp.1-7, Aug. 2010.
9) 久保田，高橋 : "GPU における格納形式自動選択による疎行列ベクトル積の高速化"，情報処理学会 HPC 研究会 Vol.2010-HPC-128 No.19, Dec. 2010.
10) NVIDIA : "CUSPARSE User Guide", http://developer.download.nvidia.com/compute/cuda/3_2/toolkit/docs/CUSPARSE_Library.pdf
11) : "cusp-library : Generic Parallel Algorithms for Sparse Matrix and Graph Computations ", http://code.google.com/p/cusp-library/
12) N. Tanabe, H. Nakajo : " An Enhancer of Memory and Network for Cluster and Its Applications", IEEE PDCAT'08, pp.99-106, Dec. 2008
13) N. Tanabe, H. Hakozaki, Y. Dohi, Z. Luo, H. Nakajo : " An enhancer of memory and network for applications with large-capacity data and non-continuous data accessing", The Journal of Supercomputing, Vol. 51, No. 3, pp. 279-309, Dec. 2009
14) N. Tanabe, M. Sasaki, H. Nakajo, M. Takata, K. Joe : "The Architecture of Visualization System using Memory with Memory-side Gathering and CPUs with DMA-type Memory Accessing", International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'09) , pp. 427-433, Jul. 2009
15) N. Tanabe, T. Tsukamoto, A. Ohta, H. Nakajo : " Efficiency Improvement for Discontinuous Accesses of Cell/B.E. Using Hardwired Scatter/Gather on Memory-side ", IEEE ICCEE'10, Nov. 2010
16) N. Tanabe, Y. Ogawa, M. Takata, K. Joe : " Scaleable Sparse Matrix-Vector Multiplication with Functional Memory and GPUs", Euromicro PDP'2011, pp. 101 - 108, Feb.2011
17) 小川，田邊，高田，城 :“機能メモリと GPU の PCI express 接続によるヘテロ環境における超大規模疎行列ベクトル積の性能予測”，情報処理学会 HPC 研究会 Vol.2010-HPC-126 No.20, Aug. 2010.
18) 田邊, Nuttapon，中條 :“Gather 機能付き拡張メモリのアクセス性能の評価”，情報処理学会 HPC 研究会, Vol.2010-HPC-128, Dec. 2010.
19) 田邊, Nuttapon，中條，小川，高田，城: “GPU と拡張メモリによる疎行列ベクトル積性能の行列形状依存性軽減”，情報処理学会 HPC 研究会, Vol.2010-HPC-129, Mar. 2011.
20) G. E. Blelloch, M. A. Heroux, M. Zagha : "Segmented Operations for Sparse Matrix Computation on Vector Multiprocessors", Technical Report. UMI Order Number: CS-93-173., Carnegie Mellon University, 1993
21) Tim Davis : " The University of Florida Sparse Matrix Collection", http://www.cise.ufl.edu/research/sparse/matrices/
22) NVIDIA : "PTX : Parallel Thread Execution ISA Version 2.1", http://developer.download.nvidia.com/compute/cuda/3_1/toolkit/docs/ptx_isa_2.1.pdf