Vol.2010-HPC-126 No.20
2010/8/4

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

小川裕佳[†]　田邊 昇[††]
高田雅美[†]　城 和貴[†]

本報告では，疎行列ベクトル積のベクトルがデバイスメモリに入りきらないほど大きな問題向けに，GPU が PCI express 越しに Gather 機能を有する大容量機能メモリをアクセスするシステムを用いた並列処理方式を提案する．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，間接アクセスの直接アクセス化により，単体性能は既存研究の最大 4.1 倍に向上した．GPU 内キャッシュが溢れる心配も無い．GPU 間通信を完全に排除可能にした構成によりスケーラビリティは保証されており，PCI express のバースト転送バンド幅で制約される単体性能にノード数を乗じたものが並列実効性能となる．

Performance Estimation of a Huge Sparse Matrix-Vector Multiplication on a Hetero Environment Constructed by PCI express with GPU and Functional Memory

Yuka Ogawa[†]　Noboru Tanabe[††]
Masami Takata[†]　and　Kazuki Joe[†]

In this report, we propose a parallel processing strategy for huge scale sparse matrix-vector product whose vector cannot be held on a device memory. The strategy uses a system connected by PCI express with GPUs and functional memories with gather function. We evaluate the performance of proposed strategy with University of Florida Sparse Matrix Collection. The result shows the 4.1 times acceleration over the existing performance record with a GPU in the maximum case. There is no risk of performance degradation by overflowing cache capacity on GPU. Because of the architecture without inter-GPU communications, scalability is guaranteed. Therefore, parallel effective performance is the product of number of nodes and single GPU performance limited by burst transfer bandwidth of PCI express.

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 4GB 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 PCI express によって接続するヘテロジニアスシステムと，その上での疎行列ベクトル積のスケーラブルな高速化を提案する．

以下、本論文では第 2 章で解決すべき課題を示し，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向けの疎行列ベクトル積アルゴリズムを示す. 第 5 章では従来の GPU クラスタや提案システムで想定される性能ボトルネックについて述べる．第 6 章では性能評価を示し，最後に第 7 章でまとめる．

2. 解決すべき課題

本研究ではアプリケーションとして疎行列ベクトル積を検討対象とする．疎行列ベクトル積は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 CG 法を代表とするクリロフ部分空間法の中核的処理である．よって非常に広範囲の科学技術計算アプリケーション上で実行時間の大半を占める．このため，数多くの研究がこの高速化に向けて行われてきた．しかしながら，とりわけランダムに近い非零要素配置を有する行列を扱う場合，キャッシュが効きにくく，メモリバンド幅がボトルネックとなる．このため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は容易ではなかった．一方，近年では広大なメモリバンド幅を背景に GPGPU でも複数の実装成功例[1][2][3][4]が報告されるようになってきた．

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．

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

ⓒ 2010 Information Processing Society of Japan

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．

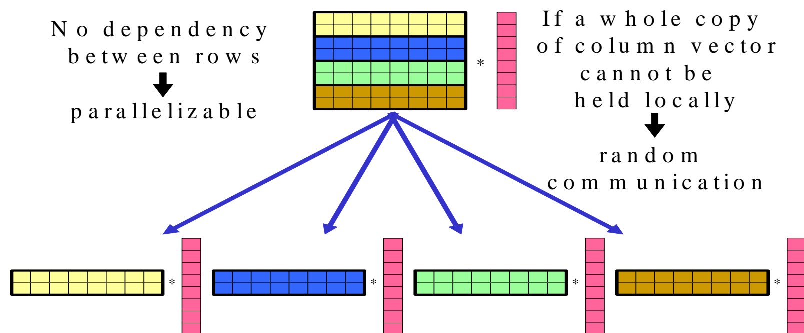

図 1　疎行列ベクトル積における並列性

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を対象とする．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．

3. 提案システムアーキテクチャ

図 2 に提案アーキテクチャの基本概念を示す．PCI express 等の高バンド幅な標準 I/O を介してアクセラレータと機能メモリを結合する．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．

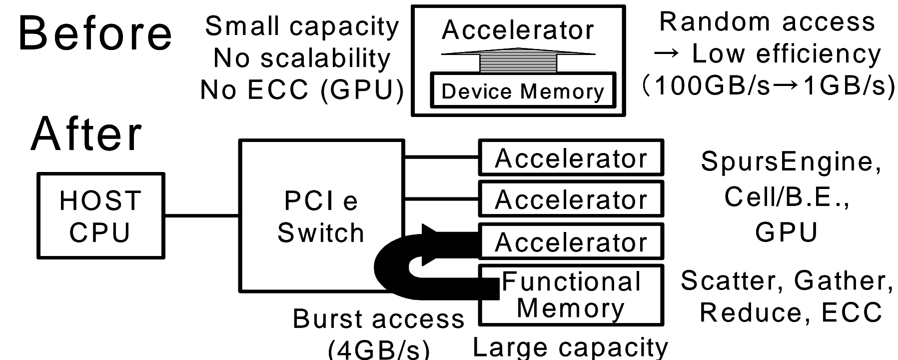

図 2　提案アーキテクチャの基本概念

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[5][6][7]に実装されている Scatter/Gather(分散/収集)機能である．

PCI express スイッチを搭載し PCI express を木構造状に分岐増設する製品が複数市場に出てきている．一部のスイッチ製品ではリーフノード間のルーティングをサポートしている．これらを階層的に接続すれば，木のルート付近のバンド幅の制約にかかることなく，低コストで PCI 空間上にマップされた多数のデバイス間での読み書きを多数並列に実行できる．つまり，機能メモリと GPU の個数の比率を調整することで，GPU あたりの実効バンド幅やメモリ容量をスケーラブルに調整できる．

4. 提案システム向け疎行列ベクトル積

本章では，GPU のデバイスメモリに入りきらないほど大きなベクトルに対する疎行列ベクトル積の提案システム向けの手順について論じる．

1.1 基本方針

まず，行列ベクトル積においては，行列データは 1 回の積和演算にしか用いられないため再利用性が無い．つまり行列データを共有メモリやキャッシュによって再利用する意義はない．行列に乗ずるベクトルには多少の再利用性が存在するが，帯幅が大きくない帯行列的な非零要素の配置でない限り，GPU 上の小容量なキャッシュではデバイスメモリ（キャッシングされる Texture メモリ）に入りきらないほどの大きなベ

ⓒ 2010 Information Processing Society of Japan

クトルを効率的に再利用することは困難であると考えられる．よって，行列データや行列に乗ずるベクトルデータの格納法やアクセス方法は，再利用よりもグローバルメモリからの転送を効率化することを優先して考える．

提案システムを用いる場合の基本的な考え方としては，GPU 上で実行する前の前処理として，GPU が扱いやすい状態にデータ構造を整える．これに加えて，GPU 向けの最適化を促進するためには，実効バンド幅が高いコアレストアクセスになるようにする点、最内側ループ内に IF 文が来ないようにする点、スレッドを多数起動して負荷が均衡するようにする点などを考慮する必要がある．

1.2 前処理

以上を踏まえて，以下の前処理を行う．前処理における行列の整形と転置の流れを図 3 に示す．

(1) 行列の整形

アプリケーションによって一行あたりの非零要素数には差があるとともに，その値のばらつき方も異なる．単純な行分割による負荷分散では非零要素数最大の行のみによって実効時間が決まってしまう．これを回避するために，行列の形を整形する必要がある．

その方法にはいくつか考えられるが，例えばホスト上で適宜 0 パディング（CRS 形式などで省略されていた零要素の位置に記憶領域を割り当て，そこを 0 で初期化すること）や折り畳み（非零要素が多い行を分割し，複数スレッドに割り当てること）を行うことで行列の形を整形する．この例では，大半の行で折り畳みが生じず，かつ，一行あたりの平均非零要素数にできるだけ近い列数を持つ縦長の二次元配列に整形する．この列数が全スレッドの最内ループの回数となるため，これを最適化することが実行時間短縮につながる．

例えば，折り目の位置を行内非零要素数の平均に係数 q を乗じたものとし，最適な q の値を経験的に探す．本論文の後半の評価においては q=1.5 の場合について評価を行った．

他の例としては，折り目を行あたり平均非零要素数＋行あたり非零要素数の標準偏差σ×r とする方式もありうる．ここで r=2 とすれば，分布を正規分布と仮定した場合には 95.4%の行で折り畳みが生じないようにできるので概ね 10%以下の行数増加に留めつつ，実行時間に直結するカーネルの最内ループ回数を行内最大非零要素数から行内平均非零要素数＋2σに短縮できる．σの導入は分布の違いをある程度反映した折り目を与えると考えられるが，平均値以下の位置で積極的に折り畳むと良い場合をこのままでは反映できない．

より汎用な最適化指標を与えるべく，上記の二つを併合した q×行内非零要素数の平均＋r×σを折り目として，最適値を与える係数(q,r)の探索は今後の課題とする．

なお，GPU での最適化に関して，アラインメントを考慮する必要があるが，次のステップ（転置）に伴い，上記の列数はアラインメントには影響しない．

一方，整形後の配列の行数はスレッド数に対応するとともに，これが半端な値であると次のステップ（転置）によって行の先頭位置のアラインメントがずれてしまう．よって，折り畳み分を加算した行数より大きく，かつ 32（複数 GPU で実行する場合は GPU 数×32）で割り切れる行数に，0 パディングによって整形する．

なお，整形方法として，0 パディングによる無駄な演算を抑制したアルゴリズムである Segmented Scan 法[8]などを適用する変形例もありうるが，GPU 上で実行した場合は IF 文やアラインメントずれの影響も予想され，その優劣は明白ではない．その実装や比較検討は今後の課題とする．

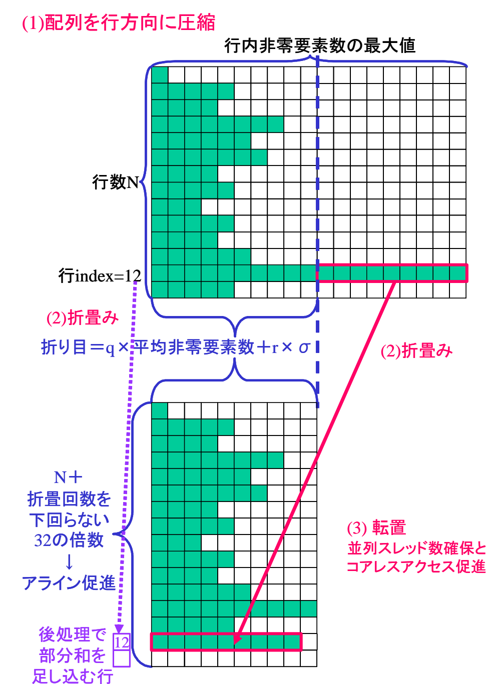

図 3　前処理における行列の整形と転置の流れ

(2) 行列およびインデックスの転置と転送

通常の CRS 形式による行列の格納方式によれば，行列の非零要素は同じ行の非零

ⓒ 2010 Information Processing Society of Japan

要素がアドレス連続方向に並ぶ．一方，GPU は隣接スレッドが同時にアクセスするデータが隣接アドレスに並ぶ時に最も効率的なメモリアクセスになる．各スレッドが行または部分行を担当するようにカーネル処理を割り当てた場合，上記の条件を満たすために(1)において整形した配列を転置する．その結果，横長の二次元配列となる．

複数 GPU で実行する場合は上記の横長配列を縦方向に等分した配列を各 GPU に分配する．一行あたりの平均非零要素数が多い行列の場合，転置処理自体がキャッシュベースのホスト CPU には苦手な処理になる．その場合は，(1)でできた配列を機能メモリに転送し，機能メモリ上で等間隔アクセスによる並べ替えを行った上で，GPU のグローバルメモリにバースト転送することで問題を回避できる．

(3) 機能メモリによるベクトルのプリロード

CRS 形式や JDS 形式などによって非零要素のみを用いた行列ベクトル積を行う場合，カーネルの最内側ループには通常だと間接参照が必要になる．つまり，乗ずるベクトルを格納する配列のインデックスが配列になっているループである．この配列が GPU のグローバルメモリに入りきらない場合には，ランダムな GPU 間通信が発生してしまい，GPU 台数を大きくしていった場合のスケーラビリティに重大な問題が発生するケースが多くなると考えられる．

そこで，容量の制約が GPU より緩い拡張機能メモリを適切な台数の GPU ごとに設置し，そこに乗ずるベクトルを格納することで，この小規模クラスタ内部に全ての通信を閉じ込める．

図 4 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(2)において転置した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 PCI express バスを介してバースト転送する．このようにすることで，GPU の PCI express バスは効率的に動作するようになるとともに，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．

1.3 カーネル部

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．

1.4 後処理

カーネル処理を終えたところで，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ホストの主記憶に転送する．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する．この計算を複数の GPU で行うと別 GPU にある部分和との加算が発生してスケーラビリティが低下する可能性がある．さらに GPU は IF 文の実行がホスト CPU に比べて得意ではない．このため，部分和を全てホストに転送し，ホスト CPU 上で折り畳んだ行の値を足しこむのが望ましいと考えられる．

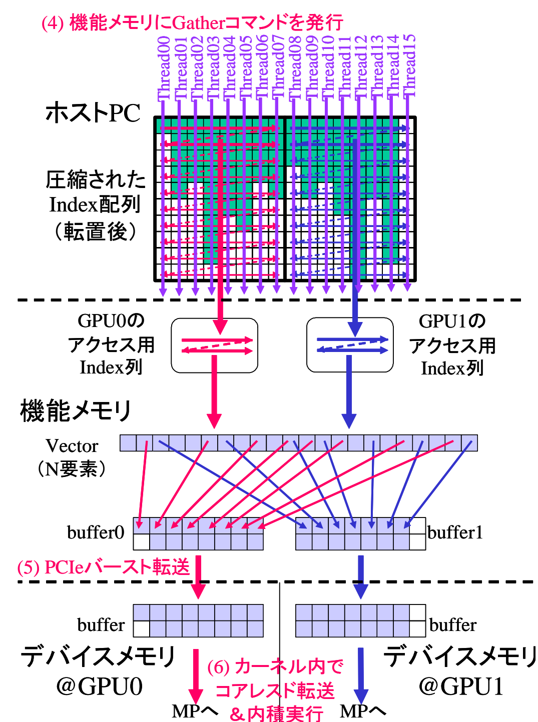

図 4　機能メモリによるベクトルのプリロードの流れ

ⓒ 2010 Information Processing Society of Japan

2. 想定されるボトルネック

本章では，提案方式やそれを用いない GPU クラスタにおいて想定されるボトルネックについて考察する．

2.1 デバイスメモリバンド幅

単精度浮動小数の密ベクトルと密ベクトルの内積に要するメモリバンド幅と演算の比率は 4 バイト/FLOP である．提案方式ではデバイスメモリ上に両方のベクトルが存在することになるので，デバイスメモリバンド幅がボトルネックになる場合の FLOPS 値は，評価の章で用いた GeForce9800GT の場合で 57.6GB/s/4B/FLOP＝14.4GFLOPS となる．Tesla C1060 の場合は 103GB/s/4B/FLOP＝25.75 GFLOPS となる．

一方，提案システムを用いない単純な GPU クラスタの場合，特に非零要素の位置にあるベクトルの要素を別の GPU から集めてこなければならず，その実効バンド幅は上記のデバイスメモリバンド幅とはかけ離れたものになる．ローカルのデバイスメモリアクセスで済む場合と済まない場合の比率によって，ベクトルに対する実効デバイスメモリバンド幅は大きく変動する．

2.2 PCI express バンド幅

PCI express Gen.2 x16 のピークバンド幅はどちらの GPU でも片方向あたり 8GB/s である．提案方式ではホストまたは機能メモリからの行列の転送や乗ずるベクトルの転送はこの経路で行われるので,デバイスメモリへのアクセスが連続化された場合は，PCI express のバンド幅がボトルネックとなる．ただし，ここで発生する転送はバースト転送であるため転送効率は高い．

一方，提案システムを用いない単純な GPU クラスタの場合，他の GPU との通信がこの経路を用いることになる．その際のバースト長は長く取ることが困難なので，実効バンド幅は提案システムを用いるよりも大幅に低くなると予想される．さらにその通信は Infiniband などのノード間結合網を介するので，通常そのバンド幅は PCI express のバンド幅よりも低い．

2.3 機能メモリ実効バンド幅

機能メモリにおける不連続アクセス時の実効バンド幅が上記のバンド幅を維持できない場合はこれがボトルネックとなる．これは機能メモリを並列に用いることで補うことも可能である．

通常の PC の主記憶はキャッシュライン単位のアクセスに対して最適化されたメモリシステムである．つまり連続アクセスに対するバンド幅は効率的であるが，不連続アクセスに対するバンド幅は低い．一方，本用途に用いられる機能メモリは不連続アクセスのスループットを高める構成を取る．具体的にはベクトル型スーパーコンピュータのメモリシステムのように，多数のバンクから構成されるインターリーブドメモリに近い構成にすれば，不連続アクセススループットが高まる．他にも，Cell/B.E.の主記憶として有名な XDR-DRAM や,その後継である XDR2-DRAM は DRAM チップの内部に多くのバンクが存在するため不連続アクセススループットが高い機能メモリの構成に適していると考えられる．具体的な機能メモリの構成や，そこで得られる不連続アクセスの実効バンド幅の評価は今後の課題とする．

3. 評価

3.1 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(Tesla 環境)および表 2(9600GT 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(Tesla 環境)

CPU	Intel® Core(TM)2 Duo CPU E8400 @ 3.00GHz
主記憶	2.7GB
GPU	Nvidia Tesla C1060(MP 数 30，メモリバンド幅 103GB/s)
ホスト I/F	PCI express x16 Gen.2(最大バンド幅 8GB/s)
OS	fedora9
CUDA	Cuda2.2

表 2 測定環境(9800GT 環境)

CPU	Intel® Core(TM)2 Duo CPU E8400@ 3.00GHz
主記憶	3.25GB (DDR2 dual channel)
GPU	Nvidia Geforce9800GT (MP 数 16，メモリバンド幅 57.6GB/s)
装着した ホスト I/F	PCI express x16 Gen.1 (最大バンド幅 4GB/s)
OS	Microsoft® Windows®XP Profesional Version 2002
CUDA	Cuda2.2

ⓒ 2010 Information Processing Society of Japan

行列は University of Florida Sparse Matrix Collection[9]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．先行研究である Cevahir らの研究[4]でも同様の行列を用いて疎行列ベクトル積の実測値を公開しているが，今回は特に Cevahir らのプログラムであまり高速化しなかったものを中心に抜粋した．

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(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

3.2 1GPU 内に収まる疎行列ベクトル積性能

上記の疎行列に対する疎行列ベクトル積の処理性能を測定した．提案手法の結果として示されている値は，提案システムによって GPU が使用するタイミングより前に GPU のデバイスメモリ上に機能メモリによるプリロードが完了していると仮定した場合の単精度浮動小数を用いた場合の処理速度である．提案手法については折り畳みをしない場合と，行平均の 1.5 倍(q=1.5)の場合の二種類について測定した．ここでは折り畳んだことにより発生する累積加算時間は隠蔽されるか，または全体の計算時間に比べ十分に小さいものと近似している。その結果を表 4 に示す．

比較対象として Cevahir らの研究における実測値を文献[4]のグラフから読み取り，併記している．上記は JDS 形式の行列格納法を基にしており，我々の最適化方針に近い方向性を有しているものの，JDS 形式では GPU に送るべき配列が 4 種類になっており，我々の 2 種類より多い上，Texture メモリに対するキャッシングによってバンド幅を改善しているもののベクトルへのアクセスは間接参照であるため，差が生じているものと思われる．表 4 において太字で示してある値が文献[4]の性能より高速化している．最も高速化したものは thermal2 で，キャッシュの効果を全く使っていないにもかかわらず，文献[4]の 4.1 倍の性能が得られた．

また，JDS 形式では結果の書き込みにおいても間接参照になっており，この部分がコアレスド転送にならない．元来，JDS 形式は間接参照にも強いベクトルプロセッサ向けに開発された方式であり，この点で必ずしも GPU 向けになっていない．これに対して提案方式では結果の書き込みも全てコアレスド転送になっており，この点も差が生じている要因の一つと考えられる．

表 4　疎行列ベクトル積の単体性能［GFLOPS］

	JDS[4]	提案手法(最大行合せ)		提案手法(折畳み)	
GPU	C1070	C1060	9800GT	C1060	9800GT
Na5	3	1.29	1.46	**5.31**	**3.86**
msc10848	3.5	2.78	1.85	**8.38**	**5.35**
exdata_1	3.4	2.10	1.53	**8.01**	**4.92**
G3_circuiit	9	11.23	7.87	**15.08**	**9.13**
thermal2	3.3	**13.54**	**10.22**	折畳みなし	折畳みなし
hood	11.5	9.17	5.76	**13.18**	7.66
F1	7.1	N.A.	N.A.	**11.25**	**7.25**
ldoor	9.8	10.68	6.25	**13.30**	7.83

最大行合せを行う提案方式の測定プログラムは 4 章で述べた提案アルゴリズムのうち折り畳みが実装されていない．行列整形を非零要素数が最大の行に合わせる．このため非零要素数が最大の行の実行時間に全実行時間が決定されている状態であり，最大と平均が 2 しか違わない thermal2 以外の行列では負荷分散がかなり酷い状態の測定値である．F1 の行列については 0 パディングによるデータサイズの肥大化に伴い，現状のプログラムは本測定環境では cudaMemcpy()実行中に実行不能になってしまい

ⓒ 2010 Information Processing Society of Japan

Vol.2010-HPC-126 No.20
2010/8/4

測定値が得られていない．

一方、平均の 1.5 倍の位置での折り畳みを適用した提案アルゴリズムによって表 5 のように列数(最内側ループ数、実行時間に対応)は最低で 1/5.3，平均 1/3.0 となるのに対し，行数(スレッド数)は平均 1.2 倍にしか増加しない．非零要素数が多い上位 5 種類に着目すると平均 1.08 倍にしか行は増えない．行の増加率は列の減少率による高速化を鈍らせる方向に働くが，大きな行列では行増加率が低水準にある．よって，折り畳みは行列サイズの増加に対して好ましい傾向を示していることがわかる．

なお，元から負荷分散がうまく行っていた thermal2 のみについては 1.5 倍の位置の折り目が最大値を超え，折畳みは発生しなかった．よって，この場合の動作は最大値合わせと同じ(加速率 1)になる．

表 5　折畳み有無と整形後の形状の変化（転置前）

	行数（スレッド数）			列数(ループ数)		
	折畳み前	折畳み後	比率	行最大	平均×1.5	比率
Na5	5888	7680	1.30	185	39	4.74
msc10848	10880	14080	1.29	300	85	3.53
exdata_1	6016	9344	1.55	1501	283	5.30
G3_circuit	1585536	1738880	1.10	4	3	1.33
thermal2	147968	147968	1.00	27	34	1.00
hood	220544	239104	1.08	51	36	1.42
F1	343808	394880	1.15	306	58	5.28
ldoor	952320	1038464	1.09	49	36	1.36
		平均比率	1.20		平均比率	3.28

次に，折り目の最適化について考察する．本報告では折り目を平均の 1.5 倍に固定して測定を行った．しかし，この 1.5 という係数 q には何らか根拠があるわけではなく，傾向をつかむために最初に測定を試みた条件に過ぎない．つまり，最適化の余地を残している．

最適化の第一の指標は，GPU に入力するために折り畳みによる整形がなされた行列の行数と列数の積（以下，計算面積とする）である．第二の指標は折り畳みの結果として後処理に回された累積加算数である．これらから構成される計算時間が最低になるような前処理整形を行うことで最適化に近づくことができると考えられる．第一の指標と第二の指標は計算量と計算時間が共通の傾きを持っていないので，計算量と計算時間の関係を明らかにしないと完全な最適化にはならない．

まず，最適化の手始めとして，第一の計算量の目安である計算面積（q=1.5 の場合の面積に対する比率）が係数 q によってどのように変化する傾向があるのかについて示したのが図 5 である．

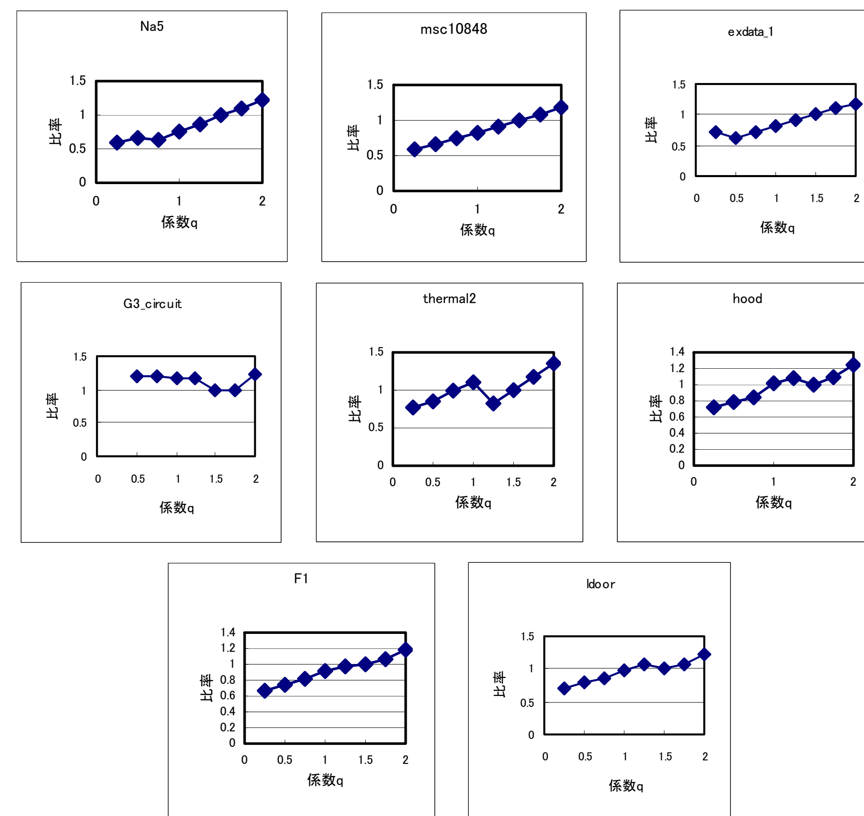

図 5　係数 q と計算面積の関係

図 5 に示されるように，グラフの形状は細かく分けると 3 種類，大別すると 2 種類に分類できる．

ⓒ 2010 Information Processing Society of Japan

(1) 最小値を持つもの（exdata_1, G3_circuit）
(2) 極小値を持つが，折り目は小さいほうが良いもの（thermal2）
(3) 単調増加のもの（その他の行列）

ここで，(1)のケースについては，GPU 上での内積計算時間と，GPU またはホスト CPU 上での累算計算時間の間に極端な差がない限り，図 5 で示された最小値が最適点を与えると考えられる．この最適点における演算性能は exdata_1 の場合が 9800GT 環境においては **7.56GFLOPS(表 4 の 1.54 倍高速化)**，Tesla 環境においては **12.29GFLOPS(表 4 の 1.53 倍高速化)**となり，表 4 の値より改善された．G3_circuit の場合は元々平均が 2 と小さいこともあり，折り目の位置が q=1.5(表 4)の場合と q=1.75 の場合で差が出なかった．

一方，計算面積最小の観点から最適な折り目の位置を考える場合，(2)は(3)と同類のものとして分類される．前述のとおり第二の指標の観点からは面積最小とすることが必ずしも最適ではないので，(2)(3)のケースについては計算量だけではなく計算時間の評価軸において第二の指標との間のトレードオフを考慮する必要がある．計算面積が単調増加するグループのこの点を考慮した最適化については今後の課題とする．

次に，現在の GPU 製品において性能を規定すると考えられるボトルネックについて考察する．提案方式ではコアレストアクセス条件の厳しい旧式の GPU である Geforce9800GT においても比較的良好な FLOPS 値が得られている．これは提案システムと提案アルゴリズムが GPU 本位のデータ構造変換を行っている効果である．

また，上記の折り畳み後の性能は概ね 4GFLOPS を超えている．提案システム上で PCI express Gen.2 x16 または PCI express Gen.3 x16 によって GPU と機能メモリが接続される場合は，2 回の浮動小数演算に対して 1 個の浮動小数を PCI express から供給する必要があることから，それぞれ 4GFLOPS，8GFLOPS が PCI express バンド幅が律速する動作モードでの性能限界値になる．上記の FLOPS 値の大小関係から，GPU の演算能力やデバイスメモリバンド幅がボトルネックになるのではなく，やや PCI express のバンド幅が不足している状況であることがわかった．

上記の結果は，提案システムでは PCI express の性能が同じである演算能力やデバイスメモリバンド幅のより低い，安価で消費電力の低いミッドレンジやローエンドの GPU に交換しても，ハイエンドの GPU を用いる場合とさほど性能差が出ないということを意味する．これは全体性能を稼ぐために台数を大きくする際に，電力効率や価格性能比の向上に貢献する．

なお，過去の我々の予備評価[10]においては PCI express はボトルネックではないと結論づけた．しかし，それはデバイスメモリをランダムアクセスする際に得られるバンド幅に基づく数百 MFLOPS 程度の実効性能を維持するにあたってボトルネックにならないという結論である．一方，提案方式によって連続化とアラインメント補正をすることで，GPU がより本来の能力を取り戻す方向に改善された後では PCI express がボトルネックになるというのが今回得られた知見であり，両者は異なる内容に関する．

3.3 1GPU 内に収まらない疎行列ベクトル積性能

提案システムにおいては 1 台の機能メモリとそれに接続している GPU をノードとして，それらが行分割によってノード間の通信を全く行わず，完全に並列に動作する．よって，機能メモリに乗ずるべきベクトルが入りきる十分に大きな問題の場合には，GPU 間通信というスケーラビリティ制約要因を完全に排除しているため，GPU 内に収まる疎行列ベクトル積の性能に，ノード数を乗じたものがシステム全体の性能となる．つまり，ノードの単体実効性能が PCI express で制約される 4GFLOPS であれば，それを 1000 ノード有する GPU クラスタでは約 4TFLOPS の実効性能が得られるものと考えられる．さらに，GPU 間通信が皆無であることから提案システムのスケーラビリティと行列の形状は無関係である．

一方，Cevahir らの研究[4]では，上記の評価で用いた行列については PCI express スイッチで接続された TeslaC1070 内部の 4 台の GPU を用いても 1 台の GPU の場合の 0.8 倍から 1.1 倍程度のスケーラビリティしかない．さらに，乗ずるべきベクトルがデバイスメモリ上に乗り切らない場合は，GPU ごとに分割してそのベクトルを保持することになるため，他の GPU が保持している部分ベクトル上のデータをネットワーク経由で取りに行く必要がある．分割台数が大きくなればなるほど，ローカルのデバイスメモリにある確率は減るためスケーラビリティ問題が深刻化すると考えられる．よって，デバイスメモリ上に全てが載っている場合の測定値である上記の FLOPS 値からさらに絶対性能や，スケーラビリティが劣化するのは確実であると考えられる．

さらに，Cevahir らの最近の別の研究[12]では，前処理として hypergraph-partitioning[13][14]を上記に追加して通信を抑制することで，スケーラビリティを改善し，32 ノードの PC クラスタ上で 64 台の Tesla を用いて 94GFLOPS を達成している．これを GPU1 台あたりにすると 1.47GFLOPS であり，1GPU での実行よりかなり落ち込んでいる．より大きなクラスタに対してはパーティションの減少に伴い通信の増加が必然なため，更なる効率の低下が避けられないものと考えられる．さらに，パーティショニングは例えば棒状のものを離散化したときのように本質

ⓒ 2010 Information Processing Society of Japan

的にうまく行く場合と，うまく行かない場合があり，スケーラビリティと行列の形状は敏感であると考えられる．これに対して，提案方式にはそのような欠点がない．

4. おわりに

提案アーキテクチャは，メモリ容量とランダムアクセススループットを強化した機能メモリが PCI express バスのバーストアクセスにより GPU のデバイスメモリ上に整列したデータを書き込む．適切な PCI express スイッチを用いることにより，その実効バンド幅を GPU の総数とは無関係に保つことができるため，疎行列ベクトル積の行列サイズをスケーラブルにできる．

本報告では，提案アーキテクチャ向けの疎行列ベクトル積のアルゴリズムを提案し，Florida University Sparse Matrix Collection を用いた性能評価を行った. その結果，単体性能においては，負荷分散を行うための行折り畳みを実装しないバージョンでも先行研究に迫る FLOPS 値を観測した．特に負荷分散が最初から取れている行列においては先行研究の最大 4.1 倍の性能向上を観測した．行折り畳みを実装することで他の行列でも負荷分散が良くなり加速が得られた．先行研究での測定値はキャッシュが概ね効いている状態と考えられるが，本手法は先行研究とは異なり，キャッシュの効果を一切使っていないので，さらに大きな行列を扱う時のヒット率低下による性能低下の心配も無い．ただし，実際には PCI express のバンド幅がボトルネックとなることが明らかになった．つまり，現状の GPU における PCI express と演算性能とのバランスから，提案システムで利用すべき GPU はハイエンドではなくミッドレンジやローエンドである．

一方，提案方式では細粒度でランダムな GPU 間通信がローカルな大容量機能メモリへのバーストアクセスに変換されているため，完全なスケーラビリティが確保されている．よって，提案方式は比較的高い水準の単体 FLOPS 値を安価で低電力なミッドレンジやローエンドの GPU から引き出した上で，それを多数並べることによって高い絶対性能と，良好な対電力性能，対価格性能を両立できる見通しを得た．ベクトル型スーパーコンピュータが経済的な問題から今後市販されなくなったとしても，その代替システムとして本提案システムは有望である．

今後の課題は行折り畳みの最適化を実装した評価，Segmented scan 法[8]を実装した評価，ストリーミングの実装と評価，機能メモリの設計と評価，間接アクセスの直接アクセス化によってボトルネックになることが明らかになった PCI express の代わりに GPU のデバイスメモリポートの一部を機能メモリの接続インタフェースとする構成の評価などがある．

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) N. Tanabe, H. Nakajo : " An Enhancer of Memory and Network for Cluster and Its Applications", IEEE PDCAT'08, pp.99-106, Dec. 2008
6) 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
7) 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
8) 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
9) Tim Davis : " The University of Florida Sparse Matrix Collection", http://www.cise.ufl.edu/research/sparse/matrices/
10) 小川，田邊，高田，城 :“GPU と機能メモリを用いたヘテロシステムによるスケーラブルな疎行列ベクトル積高速化の提案”, SACSIS2010, pp.109-110, May 2010
11) A. Cevahir, A. Nukada, S. Matsuoka : " Fast Conjugate Gradients with Multiple GPUs", The International Conference on Computational Science 2009 (ICCS 2009), pp. 893-903, May, 2009.
12) A. Cevahir, A. Nukada, S. Matsuoka : " High performance conjugate gradient solver on multi-GPU clusters using hypergraph partitioning", Computer Science - Research and Development, Vol.25, No.1-2, pp.83-91, May 2010.
13) U. V. Catalyurek and C. Aykanat, “Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication,” IEEE Transactions Parallel and Distributed Systems, vol. 10, no. 7, pp. 673. 693, 1999
14) Bora Ucar, U. V. Catalyurek :“On scalability of hypergraph models for sparse matrix partitioning”, 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing(PDP 2010), Feb. 2010.

ⓒ 2010 Information Processing Society of Japan