Vol.2013-ARC-204 No.4
2013/3/26

Scatter/Gather をどこで行なうべきか？
― 低電力で高スループットなランダムアクセスのために ―

田邊 昇[†1]　冨森苑子[†2]　高田雅美[†2]　城 和貴[†2]

HPC における反復解法のみならず、近年ではビッグデータ解析のニーズの高まりにより、大量データへのランダムアクセスの低電力化・高スループット化のニーズが高まっている。キャッシュベースの CPU や GPU において、キャッシュから溢れる配列に対して Scatter/Gather を行なうと、ライン内の空間的局所性の欠乏により消費電力とスループットの両面で深刻な問題が発生する。特に電力やメモリスループットが不足する将来の大規模計算基盤においては、この問題が年々深刻さを増す。本報告では、上記の問題の解決策として、Scatter/Gather 機能を Hybrid Memory Cube 内で行なうことを提案する。提案方式の電力やスループットに関するメリットについて、モデルを構築しつつ、Graph500 ベンチマーク課題行列に対する疎行列ベクトル積を実例に考察する。

1. はじめに

近年，2018 年頃のエクサスケールマシンの実現に向けた検討[1]が日本でも行われるようになった．エクサスケールマシンではメモリバンド幅を十分に確保できなくなる．大規模疎行列を係数とする連立一次方程式(疎行列ベクトル積)に帰着される応用ではメモリバンド幅への要求が高い．国内の重点アプリケーションの多くがこのクラスに属するため，高速化へのニーズが高い．特に非構造メッシュに由来する疎行列の場合，アクセスアドレスがランダムに近くなり，階層構造やキャッシュアーキテクチャはこのようなアクセスパターンが非常に苦手である．

さらに，エクサスケールマシンの実現や利用のためには，メモリバンド幅のスループット以上に消費電力の抑制が重要である．IPDPS2011 のキーノートスピーチ[2]において当時の Nvidia 社の技術責任者で Stanford 大学教授でもある W. J. Dally は，2018 年における Exa FLOPS マシンにおいて最も重要な制約は電力であり，それを左右するのは演算器ではなく，データを移動する際の電力，中でもオフチップメモリアクセスにおける電力であることを力説している．京コンピュータへの最大電力供給は 20MW とされており，施設への電力供給がさらに 5 割向上したとしても 30MW という制約がエクサスケールマシンの大半の設計項目の出発点になる．例えば，施設への電力供給が 30MW までしかできない場合，電力を浪費すればスループットが高くなるような戦略があったとしても，電力が 100MW かかるような動作をさせることができない．電力遮断を回避するためには，計画停電のように一部のノードを完全にスリープさせたり，プロセッサやメモリやネットワークの周波数を落として使わなければならなくなる. Dally の分析が正しいのであれば，浮動小数演算やアドレス計算などに比べ，データを長距離移動する電力の方がはるかに大きな電力を消費する．よって，無駄なデータを長距離移動しないアーキテクチャが望まれる．

また，近年ではグラフ処理に代表されるビッグデータ処理が注目を集めている．これらにおいても，巨大で複雑な非零要素配置を有する疎行列で表現される処理が必要になる．Web サイトの重要性を与える PageRank などの大規模非定型情報の検索[3]や嗜好分析・リコメンデーションにおいて，疎行列処理の大規模化と低電力化と高速化が求められている．

Graph500 ベンチマーク[4]はビッグデータ処理（グラフ解析）のベンチマークであるとともに，疎行列処理のベンチマークとして Top500[5]を補うハイエンド HPC システムのランキングに用いられているため，エクサスケールマシンにとっても重要な意味を持つ．筆者らの研究[6][7][8]によって，Graph500 ベンチマークを代表とするグラフ解析処理における空間的局所性は極めて低い(キャッシュライン内に平均 1 個程度しか有効データがないこと)が明らかになっている．このため，解析対象のグラフ（疎行列）の時間的局所性がキャッシュ容量でカバーできる範囲に無い場合は，キャッシュは性能低下や電力浪費の元凶になる．

筆者らは上記のような課題に対して，Scatter/Gather 機能を有する Hybrid Memory Cube(HMC)[9]の採用を提案してきた．Scatter/Gather は演算に必要なデータのみをレジスタ上に集め(Gather)たり，演算が終わったデータをばらばらなアドレスに分散（Scatter）して書き込む操作のことである．

元来，Scatter/Gather はベクトル型スーパーコンピュータの命令に採用されてきた．しかし，従来のベクトル型スーパーコンピュータのメモリシステムは大量のメモリ間長距離配線があるため，消費電力が大きくなってしまう．提案メモリはベクトルアーキテクチャを継承しつつも，メモリチップとメモリコントローラ間の配線長を三次元スタッキングによって極度に短縮し，データ移動に伴う消費電力を不連続アクセスに対しても削減するものである．

†1(株)東芝
Toshiba corporation
†2 奈良女子大学
Nara Women's University

*Intel , Xeon Phi は，米国およびその他の国における Intel Corporation の商標です.

一方，Larrabee[10]，Xeon® Phi™[11] [12]といった Intel®* 製メニーコアプロセッサには，キャッシュとの組合せを想定した Scatter/Gather を行なう SIMD 命令が搭載されている．一見，提案手法と似ているので，これで上記の課題が解決すると考える読者がいるかもしれないが，処理性能や電力消費において，提案メモリシステムとの間には大きな差が存在する．その差の根源は「Scatter/Gather 機能がどこにあるのか」という点にある．

本報告では，Scatter/Gather 命令の有無に関わらずキャッシュアーキテクチャ（ライン単位の外部メモリアクセス）で動作する場合と，提案メモリシステムで動作させる場合の疎行列ベクトル積における処理速度と消費電力のモデルを整理する．さらに，処理性能モデルの妥当性を Graph500 ベンチマークに用いられる疎行列を元に検証する．

本報告の構成は以下の通りである．2 章では従来の Scatter/Gather に関して述べる．3 章では提案メモリシステム(Scatter/Gather 機能を有する Hybrid Memory Cube)について紹介する．4 章では処理速度と消費電力のモデルに関して述べる．5 章では評価実験について述べる．6 章ではまとめと今後の課題について論ずる．

2. 従来の Scatter/Gather

本章では従来の Scatter/Gather としてベクトル型スーパーコンピュータとメニーコアに搭載されている Scatter/Gather に関して整理する．

2.1 ベクトル型スーパーコンピュータの Scatter/Gather

ベクトル型スーパーコンピュータ[13]には昔から Scatter/Gather を行なうベクトルロード/ストア命令を搭載している．等間隔ベクトルロード/ストア命令，リスト(間接参照) ベクトルロード/ストア命令，マスクベクトルロード/ストア命令がそれに相当する．

主記憶が多段結合網を介した多数のメモリバンクを並列稼動させるインターリーブ構成となっており，不連続なアクセスに対してもスループットが低下しにくいようなメモリシステムとなっている．

2.2 Intel 製メニーコアの Scatter/Gather 命令

Intel 社が自社のマイクロプロセッサ上に Scatter/Gather 命令を搭載したのは Larrabee が最初である．Larrabee は当初グラフィックと HPC の両方をカバーするコプロセッサとして開発された．その後，HPC 向けに特化して進化したものが MIC あるいは昨年市販が始まった Xeon® Phi™ である．Xeon Phi の Instruction Set Manual は原稿執筆時点では公開されていないが，Intel 社が各所で行なったプレゼンのスライドの中で Scatter/Gather 命令に関する言及があり，現行の Xeon® Phi™ にも Larrabee 同様に図 1 に示すような動作をする Scatter/Gather 命令を搭載されている．

これらの Scatter/Gather 命令は 512bit のベクトルレジスタ上の 16 個のインデックスとベースアドレスによるベクトル長 16 固定の間接ベクトルロードあるいはストアを実現する．マスクビットによる Scatter/Gather 命令も搭載されている．Larrabee の場合，Scatter/Gather 命令の実行時間は間接参照されるデータが属するキャッシュライン数に比例して延びる．全データ(16 個)が L1 キャッシュにヒットした場合でも，16 ラインにまたがるようなインデックスだった場合は 16 サイクルを消費する．

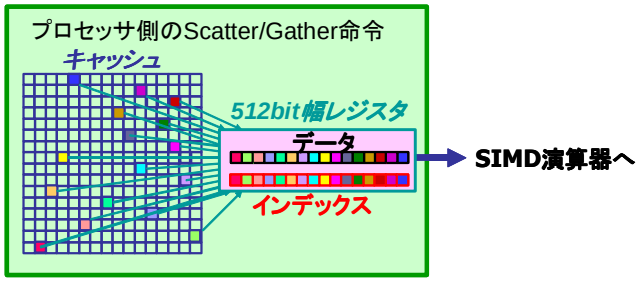

図 1　　プロセッサ側の Scatter/Gather 命令の動作

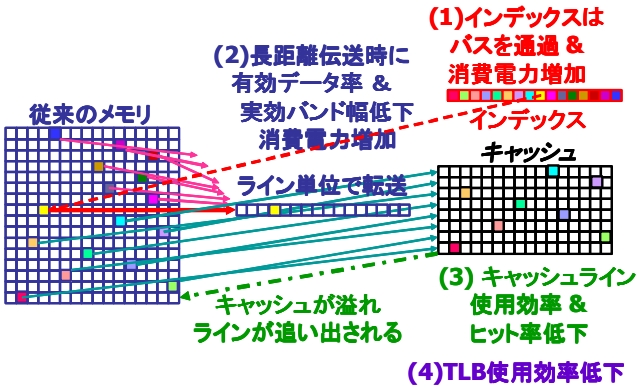

図 2　　プロセッサ側の Scatter/Gather の問題点

図 2 にプロセッサ側の Scatter/Gather の問題点を示す．図示されているような 4 つの問題点が存在する．キャッシュにヒットしている場合は外部メモリアクセスがない．このため，そのような動作ができる利用形態ではメモリバンド幅の消費を抑制できる．しかし，ミス時のリプレースや，プリフェッチによる外部メモリアクセスはライン単位で行なわれるため，ライン内に有効なデータが少ないアクセスパターンの場合は通常のキャッシュと同様にライン内有効データの比率の逆数倍のバンド幅消費と電力浪費が発生してしまうという問題がそのまま温存されている．つまり，上記の性質の観点からは，Intel 社製メニーコアの Scatter/Gather と，ベクトル型スーパーコンピュータの Scatter/Gather とは似て非なるものと言える．

加えて，現状の Xeon® Phi™ においても L2 キャッシュはコア間で共有されていない．このため，疎行列ベクトル積を実行させるような場合は，GPU よりも全体としてはるか

*Intel , Xeon Phi は，米国およびその他の国における Intel Corporation の商標です．

に大きなオンチップキャッシュ容量があるにもかかわらず，512KB の L2 から溢れるようなベクトルサイズでミスが多発するようになる．このため，図 2 の動作モードには Nvidia 社の GPU(L2 が 768KB の Ferimi[14]や 1536KB の Kepler[15])よりも Xeon® Phi™ は先に移行すると考えられる．

3. 提案メモリシステム

3.1 先行研究における提案の概要

ランダム性が高い大きなデータの際は，タイリングなどのキャッシュの挙動を十分に意識したソフトウェア的最適化が困難となり，キャッシュ容量内に有効なデータを置ききれず，問題解決が困難である．特にソフトウェアのみでは以下の二つの問題を解決できない．

(1) ライン内にごく少数の有効データしかないようなアクセスパターンでは，ライン単位の転送により，有効データが少ないゆえのパッケージ間配線バンド幅と電力の浪費が発生する．

(2) 有効データが少ないゆえのキャッシュエントリの浪費と，それに伴うリプレースの多発やヒット率の低下が発生する．

一方，図 3 に示すように田邊らが提案してきた Gather 機能付 Hybrid Memory Cube[9]のようなメモリ側の Gather 機能により，上記の問題を解決することが可能である．図 3 が図 1 と似ていることから明白なように，オフチップな転送部分を除き，その動作もハードウエア量も消費電力も大きな差は無い．それがどこに実装されるかという点においてプロセッサ側ではなくメモリ側に置くということと，Hybrid Memory Cube[16]という三次元実装技術を活用するというところが異なっている．

Graph500 の中核部分の深さ優先探索(BFS)の性能あたりの電力を競う Green Graph500[17]の観点からは，インデックスがパッケージ間配線を通過しない点と，Gather のための細粒度アクセスが積層メモリ間の短いビア配線で転送される点と，使用しないデータがパッケージ間配線を通過しない点が電力効率向上に貢献する．

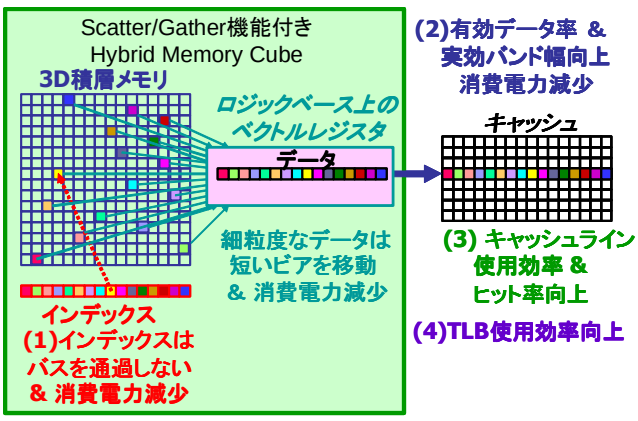

図 3 Scatter/Gather 機能つき Hybrid Memory Cube による間接参照の問題の解決

4. 処理速度と消費電力のモデル

エクサスケールマシンにおいては，処理速度が従来のように電力以外の要因(演算性能やメモリバンド幅など)で制約される場合と，電力で制約される場合が想定できる．本章では Scatter/Gather 命令の有無に関わらずキャッシュアーキテクチャ（ライン単位の外部メモリアクセス）で動作する場合と，提案メモリシステムで動作させる場合について，疎行列ベクトル積における処理速度とメモリアクセスに関する消費電力のモデルを整理する．

4.1 キャッシュの処理速度モデル

キャッシュベースのプロセッサを用いて Scatter/Gather をプロセッサ側で行なうのが通常である．本節のモデリング対象は，そのような場合の疎行列ベクトル積における処理速度である．1FLOPS あたりの必要バンド幅[B/FLOP]の式を作るにあたり，いくつかの値を以下のように定義する．

hit_x：列ベクトル x へのアクセスのヒット率

S：空間的局所性指標[18] (ライン内 32 個中の有効データ数)

I：index1 個のデータサイズ[B]

1FLOPS あたりの必要バンド幅は以下の三つの値の和で表される．

$$BPF_{cache} = \alpha + \beta + \gamma = 2 + I/2 + (1 - hit_x)*128/S \quad [B/FLOP] \quad (1)$$

α：index に必要なバンド幅：I/2[B/FLOP] (連続アクセス＆再利用性なし)

β：A に必要なバンド幅：2[B/FLOP] (連続アクセス＆再利用性なし)

γ：x に必要なバンド幅：有効データのみで 2B/s を出すのに必要なバンド幅

ここで，A は精度改良を取り入れた混合精度を用いる場合の性能を決める単精度浮動小数（4 バイト）を仮定した．また，x はキャッシュ容量より十分に大きく，キャッシュミスのリプレース動作によりキャッシュに取り込まれるものとしている．

上記の γ は 0.5 個の x(2B)をリプレースで持ってくる時に消費されるバンド幅に対応する。ミス 1 回で S 個の x(4B)をリプレースで持ってくるのに 128B を 2 回移動する。ミス率 1 の時 0.5 個の x(2B)をリプレースで持ってくるのに 0.5*256/S[B]を移動する。これはミス率 1 の時 1FLOP あたりに 128/S[B]を移動と同じである。ミス率$(1- hit_x)$の時 1FLOP あたりに $\gamma = (1- hit_x)*128/S[B]$ を移動することになる．よって，主記憶(GPU の場合はデバイスメモリ)のバン

ド幅 W_{cache}[B/s]によって得られる処理速度は以下のようになる．

$$F_{cache} = W_{cache}/(2+I/2 + (1- hit_x)*128/S) \quad [FLOPS] \quad (2)$$

4.2 提案メモリシステムの処理速度モデル

提案メモリシステムではScatter/Gatherをメモリ側で行なう．メモリ側Gatherではindexはメモリ側から動かないのでαに相当するバンド幅消費は Gather スループット W_{gather}[B/s]の中に組込まれる．βはキャッシュの場合同様2[B/s]であり，xはメモリ側 Gather によって連続化されるのでγはβ同様に 2[B/s]であり，全体として以下の式で1FLOPSあたりの必要なメモリバンド幅が表される．

$$BPF_{gather}=4 \text{ [B/FLOP]} \quad (3)$$

Gather スループット W_{gather}[B/s]の場合の得られる処理速度は以下のようになる．

$$F_{gather}=W_{gather}/BPF_{gather} = W_{gather}/4 \text{[FLOPS]} \quad (4)$$

4.3 キャッシュの消費電力モデル

本節および次節の電力モデリングにおいては，単純化のためアドレス計算に必要な電力は組込まないという近似を行なう. Dally[2]によれば今後の半導体システムにおいては計算よりのデータの長距離移動の方がはるかに大きな電力を消費するということがこの近似が正当である第一の根拠である. アドレス計算はキャッシュの場合はCPU上ソフトウェア処理または Scatter/Gather 命令を使用したメニーコア CPU におけるソフトウェア処理にかかる電力は, 提案メモリ上の専用ハードウェアによる処理と同等もしくは大きいと考えられる．ゆえに，アドレス計算に必要な電力は組込まない、という設定は提案手法にとって有利な設定というわけではないということが第二の根拠である．

キャッシュベースでの疎行列ベクトル積実行時のメモリアクセスの消費エネルギーE_{cache}は以下の式(5)でモデル化できる. ここで, キャッシュラインサイズは128B(1024bit), 行列データサイズおよびベクトルデータサイズは32bit, インデックスデータサイズは 64bit， R_{hit} はキャッシュヒット率，E_{hit} はキャッシュヒット時のメモリアクセス１ビットにかかる消費エネルギー，E_{miss} はキャッシュヒット時のメモリアクセス１ビットにかかる消費エネルギー，N_{access} は疎行列ベクトル積実行時のアクセス総数，$b_{average}$ は疎行列ベクトル積実行時の平均アクセスビット数である．E_{hit} はオンチップメモリアクセス１ビットにおける消費エネルギーE_{on}１回分に，E_{miss} はオンチップメモリアクセス１ビットにおける消費エネルギーE_{on}１回分とオフチップメモリアクセス１ビットにおける消費エネルギーE_{off} 2 回分の和に近似できる．メモリアクセス１回に行列値の場合に 32bit, インデックス値の場合に 64bit，ベクトル値の場合に 128B/$S_{locality}$ ($S_{locality}$は空間的局所性) を同じ回数ずつ取得することになるので，平均$(32+64+1024/S_{locality})/3$ ビットを取得する．N_{access} は全非零要素数 N_{nz}×3 回である．$E_{off} \gg E_{on,}$ $S_{locality} \fallingdotseq 1$ の場合，式(6)のように近似できる．Graph500 では $S_{locality}$ と R_{hit} がともに小さいことが予測されるので，E_{cache} はベクトル値のオフチップ転送にかかる消費エネルギーが大きくなることが予想される．

$$E_{cache} = (R_{hit}\times E_{hit} + (1-R_{hit})\times E_{miss})\times N_{access}\times b_{average} \quad (5)$$

$$\fallingdotseq (R_{hit}\times E_{on} + (1-R_{hit})\times (E_{on}+2E_{off}))\times N_{access}\times b_{average}$$

$$= (R_{hit}\times E_{on} + (1-R_{hit})\times (E_{on}+2E_{off}))\times 3N_{nz}\times (32+64+1024/S_{locality})/3$$

$$= (R_{hit}\times E_{on} + (1-R_{hit})\times (E_{on}+2E_{off}))\times N_{nz}\times (32+64+1024/S_{locality}) \quad (6)$$

$$\fallingdotseq 2240(1-R_{hit})\, N_{nz}\, E_{off} \quad (E_{off} \gg E_{on,}\ S_{locality} \fallingdotseq 1 \text{ の場合}) \quad (7)$$

4.4 提案メモリシステムの消費電力モデル

一方， Gather 機能付き Hybrid Memory Cube[8]でプリフェッチを行なう場合の疎行列ベクトル積実行時のメモリアクセスの消費エネルギーE_{gather} は以下の式(3)でモデル化できる．ここで E_{array}，E_{index}，E_{vector} はそれぞれ行列値(32bit), インデックス値(64bit), ベクトル値(32bit)1 要素分を取得するのに消費するエネルギーである．行列値，ベクトル値は全てバーストデータとして基板上の長距離転送によってプリフェッチされ，キャッシュから演算器にオンチップアクセスされる．これらの転送において消費されるエネルギーはそれぞれ 32 E_{on}＋32 E_{off} に近似できる．さらにベクトル値は Gather の過程で Hybrid Memory Cube 内の短距離転送がなされ，その際の 1 要素 Gaather の消費エネルギーは 32 E_{on} に近似できる. インデックス値は基板上の長距離転送はされず, 全て Hybrid Memory Cube 内のビア経由の短距離転送がなされる．よってインデックス値 1 個取得の消費エネルギーは 64 E_{on} に近似できる．

$E_{off} \gg E_{on}$ の場合，式(10)は式(11)のように近似できる．これより式(7)と比べてキャッシュの代わりに Gather 機能付き Hybrid Memory Cube によれば桁違いに消費エネルギーが小さくなることが理解できる．

$E_{off} \gg E_{on,}$ $S_{locality} \fallingdotseq 1$ の場合，キャッシュの代わりに Gather 機能付き Hybrid Memory Cube による消費エネルギーの比率は式(12)で表される．キャッシュヒット率が 15%の状況では約 30 倍，50%の状況では約 17 倍の省電力が期待でき

ることが判った.

$$E_{gather} = N_{nz} \times (E_{array} + E_{index} + E_{vector}) \quad (8)$$

$$\fallingdotseq N_{nz} \times ((32\ E_{on} + 32\ E_{off}) + 64\ E_{on} + (32\ E_{on} + 32\ E_{off} + 32\ E_{on})) \quad (10)$$

$$= N_{nz} \times (160E_{on} + 64E_{off}) \quad (10)$$

$$\fallingdotseq 64\ N_{nz}\ E_{off} \quad (E_{off} \gg E_{on} \text{ の場合}) \quad (11)$$

$$E_{cache}/E_{gather} = 35(1-R_{hit}) \quad (12)$$

5. 評価

5.1 実験環境と評価行列

今回の実験に用いた計算機環境を表 1 に示す．また，評価に用いた行列を表 2 に示す．Graph500 で扱う問題のサイズはグラフの頂点数＝2^{SCALE} であるような SCALE 値を用いて表す[3]．本評価において，ベンチマークを実行するプログラムと同様，グラフ生成において枝数が頂点数の 16 倍となるようなクロネッカーグラフを生成する．次に，生成された枝リストからグラフデータ構造に変換する．その際，生成された疎行列を表 2 に示す．本実験では疎行列の生成には Graph500 の Reference code2.1.4 の Matlab 互換の Octave 版の kronecker_generator.m および kernel_1.m を用いて作成した．kernel_1.m により生成された疎行列をテキストファイルに出力し，Matrix market 形式に変換後，自作プログラムに入力して評価を行った．

表 1 測定環境

CPU	Intel® Xeon®CPU　X5670 @ 2.93GHz
GPU	Nvidia Tesla C2050 (コア数 448) L1 キャッシュ:16KB, L2 キャッシュ:768KB デバイスメモリ: 144GB/s, 3GB
ホスト I/F	PCI express x16 Gen.2 (最大バンド幅 8GB/s)
OS	RedHat Enterprise Linux Client release5.5
CUDA	Cuda3.2

表 2　評価に用いた行列

SCALE 値	非零要素数	行数
11	45,536	2,048
12	97,010	4,095
13	203,826	8,192
14	426,578	16,384
15	883,126	32,768
16	1,818,824	65,536
17	3,730,586	131,072
18	7,609,740	262,144
19	15,481,872	524,287
20	31,401942	1,048,576

5.2 評価実験

Graph500 用疎行列の大きさ（SCALE）を 11 から 20 まで変化させ，GPU 上で疎行列ベクトルを 100 回実行した際の L1 キャッシュヒット率をプロファイラによって測定した結果を図 4 に示す．このヒット率はベクトルへの間接アクセスのものだけではなく，行列値への連続アクセスによるヒットも含まれたヒット率である．行列サイズが大きくなるにしたがってヒット率が減少していくことが良く判る．

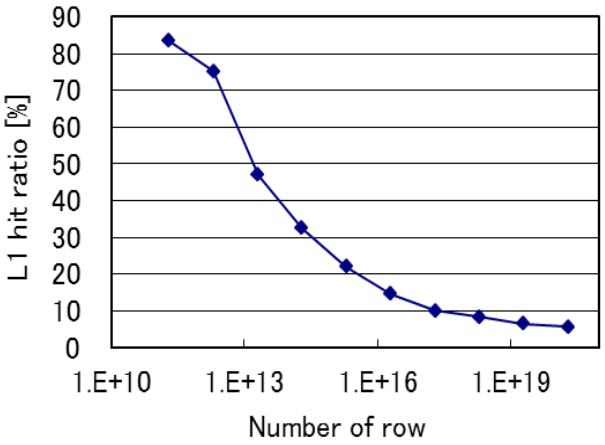

図 4　Graph500 用疎行列の大きさと L1 ヒット率の関係（GPU Nvidia C2050）

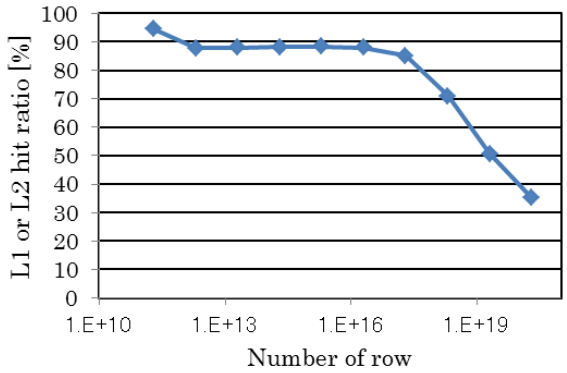

図 5　Graph500 用疎行列の大きさと L1 または L2 にヒットする率の関係（GPU Nvidia C2050）

外部メモリ（デバイスメモリ）へのアクセスになるためには L2 キャッシュでもミスをしなければならない．図 5 は L1 または L2 でヒットしたヒット率をプロファイラの情報を元に図示したものである．L1 と L2 を組合わせたものを 1 つのキャッシュと考えれば，このヒット率も図 4 と似

たような形状で SCALE を大きくしていけば徐々に落ち方が鈍りながらヒット率が落ちていくと考えられる．図 4 の形状を参考にすると，L2 ヒット率は SCALE20 ではまだ落ちきっていないと思われる．SCALE20 は GPU のデバイスメモリには何とか納まる大きさではあるが，Graph500 で実際に提出されている行列の大きさは単一ノードの場合でもさらに 64 倍から 128 倍大きな行列であるため，仮にデバイスメモリ容量がそのような SCALE まで入るならば，さらに L2 のヒット率が落ちて，キャッシュの効果は低下していくと考えられる．

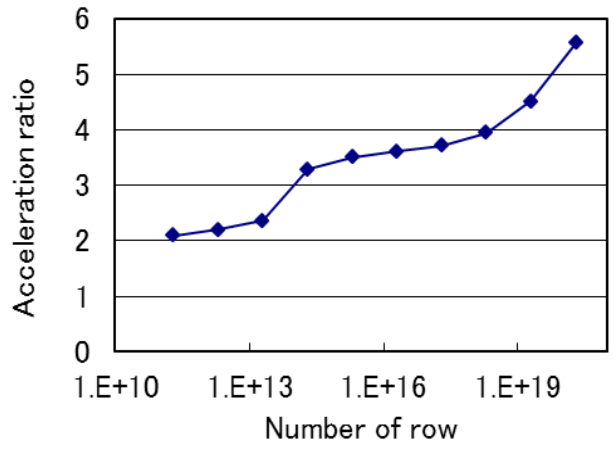

図 6　Graph500 用疎行列の大きさとキャッシュに対するメモリ側 Scatter/Gather 機能による 0 パディングの弊害を含む加速率（GPU Nvidia C2050）

次に，Graph500 用疎行列の大きさ（SCALE）を 11 から 20 まで変化させ，GPU 上で疎行列ベクトルを実行した際のキャッシュに対する Scatter/Gather 機能による加速率を図 6 に示す．ここで，提案メモリを模擬するプログラムではデバイスメモリ上に提案ハードウェアが Gather を完了していて，連続化された配列をアクセスする状況での疎行列ベクトル積の実行時間を測定するものとなっている．値は非数割り込みによるノイズが入らないように適切な初期化をしてあり，計算結果ではなく計算時間が正しく測れるような配慮をしている．ただし，現状のプログラムは Fold 法前処理 [19][20][21]を適用しており，GPU 向けに 0 パディングがなされている．今回の測定では行あたりの非零要素数の平均値の 1.5 倍の位置での折りたたみをした 0 パディングによって，全アクセス数は約 2 倍(2.13～2.25 倍)に増えている．0 は本来，主記憶からではなくレジスタから読むべきものなので，0 値に伴う不要なデバイスメモリアクセスを含んだ形になっている．一方，キャッシュのプログラムでは，0 パディングはヒットにカウントされる．つまり，本測定プログラムは提案手法の性能が半分程度に低く観測される状況設定になっている点に注意されたい．

ここで，性能モデルにパラメータを挿入して検証を試みる．GPU の最大デバイスメモリバンド幅 W_{cache} は 144GB/s である．SCALE20 の index は 4B で表現可能であるので I=4 である．SCALE20 においては L1 または L2 にヒットする率は図 5 に示すように 35.1%である．これは配列データと index の連続アクセス(32 回のアクセス中に 31 回ヒット)によるヒットも含んだヒット率 hit であり，性能モデル式(2)の hit_x を得るには空間的局所性指標 S を用い，式(13)(14)を用いて換算する必要がある．

$$hit=\{hit_x*32/S+31/32*2\}/(32/S+2) \quad (13)$$

$$hit_x=\{(32/S+2)*hit-31/32*2\}*S/32 \quad (14)$$

空間的局所性指標の値は文献[7]の結果から S=1.2 である．式(14)と S と hit から SCALE20 における hit_x=0.155 となる．W_{cache} ， I, S, hit_x を式(2)に代入すると F_{cache}＝1.53[GFLOPS]となる．

一方，提案メモリシステムの評価プログラムにおいては 144GB/s のデバイスメモリバンド幅を配列データと連続化された x で半分ずつ分け合う形で動作するので，Gather スループット W_{gather}＝72GB/s となる．これを式(4)に代入すると，F_{gather}＝18[GFLOPS]となる．

よって SCALE20 における性能モデルから導かれる加速率 F_{gather} /F_{cache}＝11.8 倍となる．この値は図 6 の観測された加速率 5.76 倍の約 2 倍になっている．これは 0 パディングによる前述の効果（観測値が半分程度になる）ことと符合し，性能モデルが妥当であるという一つの証拠が得られた．

図 6 の測定結果の加速率のグラフはジグザグ状に折れ曲がっている．この現象は L1 キャッシュと L2 キャッシュが溢れる SCALE にズレがあることが一因であると考えられる．つまり，SCALE11 から 14 に行列サイズが増えた時は L1 ヒット率が半分以下に落ち込むので加速率の伸びが顕著になる．SCALE14 から 17 に行列サイズが増えた時は，L1 キャッシュのヒット率の変化が鈍る上に L2 キャッシュはヒットしていると考えられるため，加速率の伸びが一旦鈍る．SCALE17 から 20 に行列サイズが増えた時は，L2 キャッシュのミスが始まり，再び加速率が上がり始める．以上のような複数要因の組合わせによりジグザグな曲線が出来上がったと考えられる．

また，L1 キャッシュのヒット率が SCALE20 では 5.8%に過ぎない．L1 または L2 キャッシュにヒットする率も 35.1%である．4 章の電力モデル式(12)に上記を代入すればメモリアクセスの電力比は 20.7 倍の差が出る，SCALE を 20 からさらに大きくしていくと L2 キャッシュが L1 のようにほとんどヒットしなくなるのも時間の問題であると考えられる． L1 または L2 キャッシュにヒットする率が 15%の状況になるとメモリアクセスの電力は 30 倍程度の差が出るようになると考えられる．

6. おわりに

本報告では，Scatter/Gather 命令の有無に関わらずキャッ

シュアーキテクチャ（ライン単位の外部メモリアクセス）で動作する場合と，提案メモリシステムで動作させる場合の疎行列ベクトル積における処理速度と消費電力のモデルを整理した. Scatter/Gather 命令の有無よりも, Scatter/Gather を実装する場所がそれらに大きな影響を与える.

キャッシュアーキテクチャではヒット率が低いとメモリアクセスにかかる時間と消費電力が増加する．キャッシュを用いた場合と提案メモリシステムを用いた場合のメモリアクセスにかかる消費電力は，グラフ解析処理などで見受けられる低いヒット率においては桁違いな差(例えばベクトルアクセスのヒット率 15%の時に約 30 倍)が生じることがわかった．つまり，少なくとも電力の観点から，Scatter/Gather はプロセッサ側ではなく，メモリ側に設けるべきである.

さらに, 処理性能モデルの妥当性を Graph500 ベンチマークに用いられる疎行列を元に検証した. SCALE20 において約半分の加速率が観測される測定プログラムによる観測値は, 性能モデルから得られる 11.8 倍の加速率の約半分になっており, 性能モデルが概ね妥当であることが確認された.

しかし，現状の性能モデルは 1 階層のキャッシュしか組込まれておらず，複数階層のキャッシュが示す複雑な性能曲線は十分に反映できていないという課題が明らかになった. L1 キャッシュのヒット率は SCALE20 という Graph500 で実際に競われているものよりは数桁小さい行列でも 5.8%に過ぎなかった．L1 または L2 キャッシュにヒットする率も 35.1%であった．これが PC の主記憶並みに GPU のデバイスメモリサイズが拡大し，現在の 100 倍のサイズに拡大したら L2 キャッシュも同様な状態に陥ると思われる．そのような状況では，前述のようにメモリアクセスの電力が 30 倍という大きな差が生じうる.

今後の課題は，CPU や主記憶を用いたより大きな疎行列における L2 キャッシュも含めた挙動の解析，階層キャッシュに対する処理性能や電力のモデリング，時間的局所性[22]とヒット率の関係の解析などがある.

謝辞　本研究の一部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) 平木　："[招待講演]将来の HPC アーキテクチャ", ハイパフォーマンスコンピューティングと計算科学シンポジウム 2012 (HPCS'12), pp.163-167, Jan.2012.
2) W. J. Dally : "Power, Programmability, and Granularity: The Challenges of ExaScale Computing", IPDPS2011 keynote (2011). http://techtalks.tv/talks/keynote-power-programmability-and-granularity-the-challenges-of-exascale-computing/54110/
3) X. Yang, S. Parthasarathy, P. Sadayappan : "Fast sparse matrix-vector multiplication on GPUs: implications for graph mining", Proc. VLDB Endowment, Vol.4, No.4, pp.231-242, Jan. 2011.
4) Graph500 : http://www.graph500.org/.
5) Top500 : http://www.top500.org/.
6) 田邊，冨森，高田，城："グラフ解析ワークロードのキャッシュ適合性", 電子情報通信学会コンピュータシステム研究会 vol. 112, no. 237, CPSY2012-42, pp. 67-72,　Oct. 2012.
7) 田邊，冨森，高田，城："疎行列のキャッシュ適合性に基づく Graph500 ベンチマークの特性解析",　情報処理学会研究報告 2012-HPC-138, Feb. 2013.
8) N. Tanabe,　S. Tomimori,　M. Takata,　K. Joe : " Preliminary evaluation of the potential of intracache-line optimization in graph processing ", Euromicro PDP2013, Feb.2013
9) 田邊，堀，Nuttapon，中條 : "Gather 機能を有する Hybrid Memory Cube の FPGA を用いた予備評価", 情報処理学会研究報告 2010-HPC-133, Mar. 2012.
10)　L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, P. Hanrahan : "Larrabee: A Many.Core x86 Architecture for Visual Computing", ACM Trans. Graph. 27, 3, Article 18, Aug. 2008.
11)　George Chrysos : "Knights Corner, Intel's first Many Integrated Core (MIC) Architecture Product　",　Hotchips 24, Aug. 2012. http://www.hotchips.org/wp-content/uploads/hc_archives/hc24/HC24-3-ManyCore/HC24.28.335-XeonPhi-Chrysos-Intel.pdf
12)　Intel　: " Intel® Xeon® Phi™ Coprocessor Data sheet", Nov. 2012.
13)　日本物理学会：　"スーパーコンピュータ",　培風館 1985
14)　NVIDIA : "ホワイトペーパー　NVIDIA の次世代 CUDA™ コンピュートアーキテクチャ　Fermi ™", http://www.nvidia.co.jp/docs/IO/81860/NVIDIA_Fermi_Architecture_Whitepaper_FINAL_J.pdf
15)　NVIDIA : "ホワイトペーパー　NVIDIA の次世代 CUDA™ コンピュートアーキテクチャ　Kepler ™" GK110 ", http://www.nvidia.co.jp/content/apac/pdf/tesla/nvidia-kepler-gk110-architecture-whitepaper-jp.pdfr
16)　Micron Technology, Inc. : " Hybrid Memory Cube : Breakthrough DRAM Performance with a Fundamentally Re-Architected DRAM Subsystem", Hotchips 23, Aug. 2011.
17)　Green Graph500 : http://green.graph500.org/.
18)　冨森，田邊，小郷，高田，城："疎行列のキャッシュへの適合性分類に関する予備評価", 情報処理学会研究報告 2012-HPC-135, Aug. 2012
19)　N. Tanabe,　Y. Ogawa,　M. Takata,　K. Joe : " Scaleable Sparse Matrix-Vector Multiplication with Functional Memory and GPUs", Euromicro PDP2011, Feb.2011
20)　田邊，小郷，小川，高田，城："Gather 機能を有するメモリアクセラレータの疎行列計算への応用", ハイパフォーマンスコンピューティングと計算科学シンポジウム 2012 (HPCS'12), pp.32-41, Jan.2012.
21)　田邊，小郷，小川，高田，城："長行を折畳む疎行列ベクトル積方式と Gather 機能付メモリによる高速化",　情報処理学会 ACS 論文誌, pp. 112-124 Aug. 2012.
22)　冨森，田邊，高田，城："時間的局所性を考慮した疎行列のキャッシュ適合性", 情報処理学会研究報告 2012-HPC-137,　Dec. 2012.