Vol.2015-HPC-150 No.35
2015/8/6

密結合並列演算加速機構TCAによる並列GPUコードの性能予測モデル

松本 和也[1,a)]　塙 敏博[2]　藤田 典久[3]　桑原 悠太[3]　朴 泰祐[1,3]

概要：筑波大学計算科学研究センターでは，GPU クラスタにおけるノード間に跨る GPU 間通信のレイテンシの改善を目的とした密結合並列演算加速機構 TCA (Tightly Coupled Accelerators) を独自開発している．TCA はノード間のデータ通信を低レイテンシで実現するが，TCA を使うことで必ずしも性能を向上させられるわけではなく，どのような特徴を持ったアプリケーションにおいて TCA を用いることが有効であるかはまだ明確ではない．そこで，本研究においてはデータ通信において TCA を用いた場合の性能を予測するモデルの構築を行い，TCA が有効となる条件を調査する．本稿では，性能予測モデルについて現時点までに検討していることを述べる．Point-to-point 通信と Broadcast 通信についての性能モデル式を構築した結果を示す．そして，一次元分割による Conjugate Gradient 法（CG 法）実装を例として，性能モデルをアプリケーションに対してどのように考えて適用していくのかについてを記述する．

1. はじめに

近年，高い演算性能とメモリバンド幅性能を持つ GPU を汎用的な計算に活用する GPGPU（General Purpose GPU）に関する研究が盛んに行われている．GPU は消費電力あたりの演算性能という点においても CPU を上回り，GPU を計算加速機構として搭載した GPU クラスタも高性能計算分野において普及している．しかし，GPU クラスタで並列処理計算を行うためには，複数ノードに跨る GPU 間データ通信が必要であり，その通信性能は GPU の演算性能と比べて充分に高いとは云えない．計算ノードを跨がる GPU 間の通信はホストを経由して行う必要があるため，その通信レイテンシはホストメモリ間で通信を行う場合よりも大きくなる．また，小さなデータを頻繁に GPU 間でやりとりするようなアプリケーションにおいては，通信のバンド幅性能よりもレイテンシの小ささが重要になることがある．

そこで筑波大学計算科学研究センターでは，ノードを跨ぐ通信に関わるレイテンシの改善を目指して密結合型並列演算加速機構 TCA (Tightly Coupled Accelerators) をを提唱し，その概念の実証系として GPU クラスタ向け通信機構 PEACH2 (PCI Express Adaptive Communication Hub ver.2) を独自開発している [1], [2]．TCA はノードを跨ぐ GPU などのアクセラレータ間の直接通信を可能にするインターコネクション・ネットワークに関する技術概念である．2013 年 10 月からは，GPU クラスタ HA-PACS[3] の拡張部として，TCA 実証環境である HA-PACS/TCA が稼働している．なお，PEACH2 に基づく TCA 実装は正しくは TCA/PEACH2 と呼ばれるが，本稿で単に TCA と記した場合はこの TCA/PEACH2 を指す．

TCA については Ping-pong 通信の性能評価 [2] を始めとして、いくつかの性能評価が行われている [4], [5], [6], [7], [8]．実際のアプリケーションに TCA を利用した場合の有効性については，姫野ベンチマーク [7]，格子 QCD のライブラリ QUDA[4]，Conjugate Gradient 法（CG 法）[5]，FFT[8] に対してこれまでに行なってきた．これらの性能評価により，TCA はノードを跨ぐ GPU 間通信を低レイテンシで実現できることが確かめられ，通信時間が計算時間よりボトルネックとなる場合において，TCA は有効であることが示されている．

特に短メッセージ通信におけるレイテンシ削減を主眼としており，TCA はアプリケーションのストロング・スケーリング性能の向上させることが期待される技術であるが，TCA により必ずしも性能を向上できるわけではない．GPU クラスタにおけるアプリケーションの並列計算では，並列化によりプロセス毎の演算量が通常は減るために，一種のメニーコア・プロセッサである GPU を効率的に利用して演算できなくなる場合がある．それに加えて，計算に参加する計算ノード数（プロセス数）が多くなると，通常は

[1] 筑波大学計算科学研究センター
[2] 東京大学情報基盤センター
[3] 筑波大学大学院システム情報工学研究科
[a)] kzmtmt@ccs.tsukuba.ac.jp

通信パターンが複雑になり通信時間が増えてしまう．その結果，並列化による演算時間の削減量に比べて並列化による通信時間の増加量が大きくなってしまい，並列化の効果がなくなる場合においては性能が改善されなくなる．その一方で，現状の TCA は PCI-Express（PCIe） Gen2 技術を利用しているためにその通信バンド幅性能は InfiniBand と比べて低く，限りがある．そのため，並列化が性能向上に与える効果が大きいような，総演算量が多い問題の計算の通信に TCA を利用すると，InfiniBand を利用した場合よりも計算性能が低くなってしまう場合がある．

本研究においては，どのようなアプリケーション，及びどのような問題サイズの範囲において TCA が有効なのかを明らかにするために，性能予測モデルの構築を試みる．厳密な性能予測は目的とはせず，おおよその性能を予測し TCA を適用することの有用性の有無について事前に予測できるようにすることを目的とした性能モデルを考える．本稿では，性能予測モデルについて現時点で検討していることを記述する．Point-to-point 通信と Broadcast 通信についての性能モデル式を構築し結果を示す．そして，一次元分割による Conjugate Gradient 法（CG 法）実装を例に性能モデルをアプリケーションに対してどのように適用していくのかという観点で考察した結果を述べる．

2. 密結合並列演算加速機構 TCA

2.1 TCA と PEACH2

密結合並列演算加速機構 TCA（Tightly Coupled Accelerators) は，アクセラレータ間（演算加速機構間）の直接結合に関する通信機構技術のことであり，その詳細は文献 [1], [2], [9] に詳しい．本節では，本稿を理解するために必要な TCA の概要情報を説明する．

PEACH2 (PCI Express Adaptive Communication Hub version 2) は，TCA を実現するためのインタフェース・チップである [2]．PEACH2 は，PCI-Express（PCIe）をアクセラレータ間通信に利用する．PCIe は，シリアルバス・インタフェースであり GPU ボード，Ethernet ボード，InfiniBand (IB) ボードなどの外部機器をホストコンピュータに結合するために広く使われている．PEACH2 同士を PCIe 外部ケーブルにより結合することにより，クラスタシステムを構築することができる．

PEACH2 ボードは，最大 4 GB/s のバンド幅でデータ通信を行う PCIe Gen2 x8 ポートを 4 つ持つ．1 ポートはホストとの接続に用い，残りの 3 ポートは他のノードの PEACH2 と接続するために用いられる．

PEACH2 は，PIO と DMA の 2 つの通信方式を備えている [2]．PIO 通信は，CPU の store 操作によりリモートノードへのデータ書き込みを行う．通信レイテンシが極めて小さいため，小さなデータの通信に向いている．なお，PEACH2 は CPU 間の PIO 通信のみを提供する．それに対して DMA 通信機能は，PEACH2 チップに搭載されている DMA コントローラにより実現される．DMA 通信は，データの読み込み元と書き込み先の PCIe アドレスおよび書き込むデータのサイズを記述したディスクリプタに沿って行われる．PEACH2 は Chaining DMA 機能を備えており，複数のディスクリプタをポインタ連結しておけば，先頭のディスクリプタに対する通信開始の命令を送ることで連続した DMA 処理が可能である．DMA 通信のレイテンシは PIO 通信のレイテンシと比べて大きいが，DMA の実測バンド幅は PIO のバンド幅より大きく，大きなデータの通信では DMA を用いる方が高速な通信が可能である．

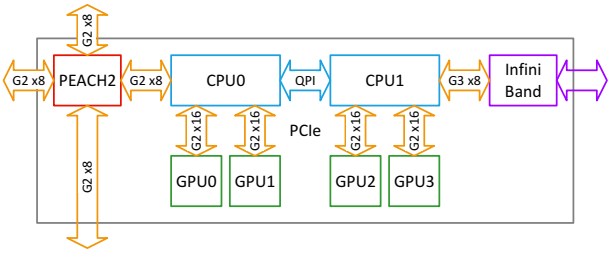


図 1 HA-PACS/TCA のノード構成
Fig. 1 Node configuration of HA-PACS/TCA

2.2 HA-PACS/TCA

HA-PACS（Highly Accelerated Parallel Advanced System for Computational Sciences）は，筑波大学計算科学研究センターで開発・運用されている，アクセラレータ技術に基づく大規模 GPU クラスタシステムである．HA-PACS は，2012 年 2 月に運用が開始されたベースクラスタ部と，2013 年 10 月に運用が開始された TCA 部（HA-PACS/TCA）から成る．HA-PACS ベースクラスタはコモディティ製品により構成されているのに対し，HA-PACS/TCA にはコモディティ製品に TCA を通信機構として加えた構成である．HA-PACS/TCA は TCA アーキテクチャの実証環境システムであり，PEACH2 ボードの実験環境でもある．本研究では HA-PACS/TCA のみを用いる．

表 1 に HA-PACS/TCA の構成仕様を記し，HA-PACS/TCA の計算ノードの構成を**図 1** に示す．それぞれの計算ノードは 2 ソケットの Intel Xeon E5-2680v2 (IvyBridge-EP) CPU と 4 つの NVIDIA K20X (Kepler GK110) GPU を演算装置として持つ．HA-PACS/TCA は 64 ノードから構成されるが，その 64 ノードは 16 ノードずつ 4 つのサブクラスタに分けられる．16 ノードのサブクラスタは**図 2** に示すように 2×8 トーラスのトポロジで PEACH2 により接続されている．また，HA-PACS/TCA の全 64 ノードは，2 ポートの InfiniBand QDR によるフルバイセクションバンド幅の Fat Tree ネットワークによってもつながれている．

TCA の PEACH2 は PCIe Gen2 x8 技術を利用してい

表 1　HA-PACS/TCA システムの構成仕様
Table 1　System Specification of HA-PACS/TCA

ノード構成	
マザーボード	SuperMicro X9DRG-QF
CPU	Intel Xeon E5-2680 v2 2.8 GHz × 2 (IvyBridge 10 cores / CPU)
メモリ	DDR3 1866 MHz × 4 ch, 128 GB (=8 × 16 GB)
ピーク性能	224 GFlops / CPU
GPU	NVIDIA Tesla K20X 732 MHz × 4 (Kepler GK110 2688 cores / GPU)
メモリ	GDDR5 6 GB / GPU
ピーク性能	1.31 TFlops / GPU
インターコネクト	InfiniBand: Mellanox Connect-X3 Dual-port QDR TCA: PEACH2 board (Altera Stratix-IV GX 530 FPGA)
システム構成	
ノード数	64
インターコネクト	InfiniBand QDR 108 ports switch × 2 ch
ピーク性能	364 TFlops

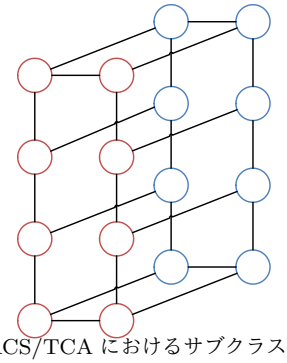


図 2　HA-PACS/TCA におけるサブクラスタの TCA トポロジ
Fig. 2　TCA topology of a sub-cluster on the HA-PACS/TCA

るので 4 GB/s が理論ピークバンド幅であるが，dual-rail InfiniBand QDR は PCIe Gen3 x8 で接続されており，ほぼ倍の理論ピーク性能で通信が可能である．このため，TCA による通信は，一定長以下のメッセージであればその低レイテンシ性が活かされるが，これを越えるメッセージ長の通信ではバンド幅で InfiniBand の方が優位となる [2]．

3. 性能予測モデル

本節については，本研究で構築を進めている性能予測モデルについて述べる．本研究では，MPI で書かれた並列コード内の通信部分を TCA/PEACH2 を利用する形に移植した場合について性能予測モデルの構築を図る．本研究においては，TCA/PEACH2 を用いることが，MPI/IB (InfiniBand による MPI) を用いた場合と比べ，どのような条件または範囲において性能的に有利になるのかを明らかにすることを目的としてモデルを構築する．そのため，各ネットワークにおけるモデルによる予測性能と実測性能との誤差はそれほど重要視せず，比較的簡単に適用できておおよその性能を予測でき，2 つのネットワークの特性・使い分けの指標となるものを検討する．

並列計算において，並列化の対象となる部分全体の計算時間は，単純な場合には演算時間と通信時間の合計時間であると考える．なお，本研究では演算と通信をオーバーラップした場合の性能モデルについては扱わない．TCA は低レイテンシ通信を特徴としており，それを活かしたアプリケーションのストロング・スケーリング性能の向上に効力を発揮することが期待されている．そのため，本研究では演算と通信をオーバーラップさせることによる性能的な利点がほとんど無いような場合を対象とする．

通信時間に関しては，通信に関するモデル式を構築しそのモデル式により性能予測できるようにする．最も基本的な通信性能である通信レイテンシと最大実効バンド幅は，Point-to-point 通信の性能を測定した結果から求めた値を使用する．Collective 通信については 16 ノードを超えた場合の性能も予測できるようなモデル式の構築を行う．最大バンド幅性能で劣る TCA/PEACH2 がその低レイテンシという特徴を活かすことで MPI/IB を上回る性能を得られるのはどのような通信パターンを持つ場合なのかをわかるようにする．

演算性能に関しては，通信性能と比べても予測が困難だと云える．アプリケーションの各部分における演算数・メモリアクセス量などを計測しそれを基に演算性能についてのモデルを構築することを検討している．また，GPU で演算を行うためには GPU 用の演算カーネル（CUDA カーネル）をプログラム内で実行する必要があるが，そのカーネルの起動に関わるコストがアプリケーションのストロングスケーリングを阻む要因になる場合がある [6]．本研究においては，その起動コストを実機で測定し演算性能のモデル式に反映させる．アプリケーションの並列化により性能が向上するのは、並列化によって削減できる演算時間より並列化によって増加する通信時間が小さい場合であるので，ある程度まとまった量の演算が必要な場合でないと並列化の効果がない．性能モデルを利用することにより，演算量の観点からそれぞれのアプリケーションに対する並列化の利点が望める下限の問題サイズの範囲を特定できるようにする．

4. 通信性能の予測

本節おいては HA-PACS/TCA における Point-to-point 通信の性能と Broadcast 通信の性能予測モデル式の構築結果について述べる．性能の測定条件を**表 2** に示す．性能の測定には HA-PACS/TCA の 1 サブクラスタ（最大 16 ノードまで）を用いる．本研究では 1 ノードあたり 1 GPU のみを用いており，これ以降の記述における利用プロセス数は利用ノード数と一致する．

表 2　性能の測定条件

OS	CentOS Linux 6.4 Linux 2.6.32-358.el6.x86_64
GPU プログラミング環境	CUDA 6.5
C コンパイラ	Intel Compiler 15.0.2 (Composer XE 2015.2.164)
MPI 環境	MVAPICH2 GDR 2.1a

TCA/PEACH2 を用いた実装との比較のために，本稿では InfiniBand 向けの MPI 実装の一つである MVAPICH2-GDR 2.1a（以下 MV2GDR）[10] による性能も適宜示す．MV2GDR は，TCA と同様に GPU-Direct for RDMA（GDR）技術 [11] が使われている．GDR により GPU メモリと InfiniBand ボードとの間で直接アクセスが可能となり，InfiniBand を経由した小サイズデータ通信の際のレイテンシが通常の MVAPICH2 と比べて改善されている．また，MV2GDR は GDRCopy[12] ライブラリを利用することで更に低レイテンシな通信が可能となる．GDRCopy はホスト CPU からの読み書きによって GPU メモリコピーを低レイテンシで行うためのライブラリである（GDRCopy も GDR 技術を使用している）．MV2GDR を用いる際に通信プロトコル・スイッチするデータサイズを調節するためのパラメタをチューニングすることで性能を改善できる．本稿では**表 3** に示すように設定して測定を行った場合の性能を記す．なお，Intel SandyBridge 及び IvyBridge アーキテクチャにおいては 2 ソケットの QPI を跨いだ PCIe デバイス間での通信性能が著しく低下することが知られている [2]．よって，QPI を跨がない場合の通信性能を測定するために，本研究においては TCA/PEACH2 による通信の測定時には GPU0 と GPU1 を，そして MPI/IB による通信の測定時に GPU2 と GPU3 を利用するように CUDA_VISIBLE_DEVICES 環境変数を設定して性能を測定する．

4.1　Point-to-point 通信の性能

基本的な通信性能である Point-to-point 通信（P2P 通信）のバンド幅性能の測定結果を**図 3** に示す．これはノードを跨ぐ GPU-to-GPU 間通信の性能で，最大バンド幅性能は TCA/PEACH2 が 2.8 GB/s で MPI/IB が 5.4 GB/s である（4 MB 以上のサイズでは TCA/PEACH2 の DMA 通信は正常に動作しないため未掲載）．64 KB までのデータサイズでは TCA/PEACH2 の方が速いが，128 KB 以上のサイズでは MPI/IB に性能逆転される．また，最小レイテンシは TCA/PEACH2 が 2.1 μsec であり MPI/IB が 3.0 μsec である．

この最大バンド幅性能の逆数 α[s/B] と最小レイテンシ β[s] とにより P2P 通信の通信時間 T_{P2P} は n[B] をデータサイズとするとき，

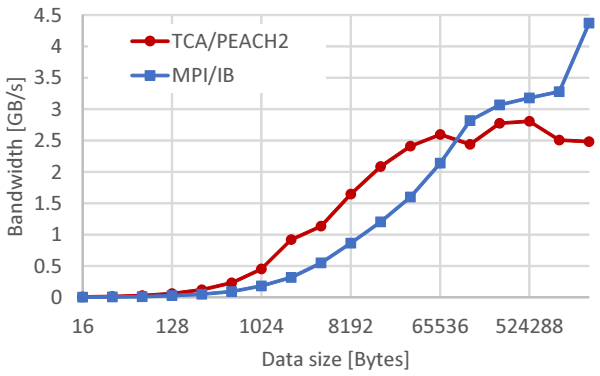


図 3　Point-to-point 通信のバンド幅性能（100 回測定の平均時間）

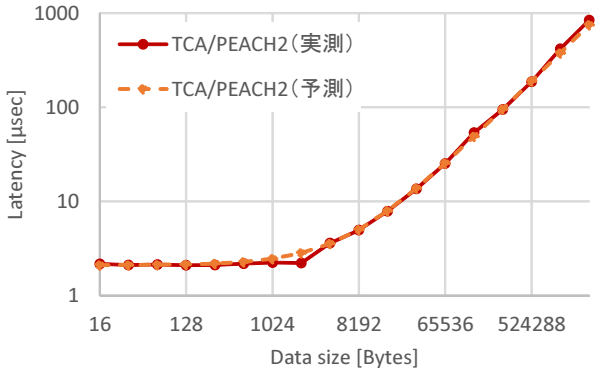


図 4　Point-to-point 通信の実測時間と予測時間（実測時間は 100 回測定の平均）

$$T_{P2P} = \alpha n + \beta \tag{1}$$

という式により予測できる．TCA/PEACH2 による P2P 通信の実測時間と式 1 による予測時間を**図 4** に示す．通信時間の実測時間と予測時間の相対誤差は以下の式 2 により求めたとき，データサイズが 4096 Bytes 以上の場合において誤差は最大で 11%で大きな誤差はないと云える．

$$\text{予測誤差} = \frac{|\ \text{実際の通信時間} - \text{予測通信時間}\ |}{\text{実際の通信時間}} \times 100[\%]. \tag{2}$$

なお，MPI/IB による P2P（Send and Receive）通信実装に関しては，データサイズにより通信方式を切り替えており，単純に式 1 により性能予測できないためサイズによって分けて予測を行う必要がある．

4.2　Broadcast 通信の性能

P2P 通信よりは複雑な Collective 通信の一つである Broadcast 通信の性能予測モデル式を考える．Broadcast の通信アルゴリズムとしては，**表 4** のようにデータを送信するプロセス数を毎通信ステップごとに倍にしていき $\log_2 p$ で通信を完了するアルゴリズムを実装している（p はプロセス数）[13]．実装では TCA/PEACH2 の Chaining DMA 機能を利用し，送信が必要なプロセスにおいて DMA

表 3　プログラム実行時の設定

タイプ	環境変数設定
TCA/PEACH2	CUDA_VISIBLE_DEVICES=0,1 MV2_ENABLE_AFFINITY=0 MV2_USE_CUDA=1 numactl –cpunodebind=0 –localalloc
MPI/IB	CUDA_VISIBLE_DEVICES=2,3 MV2_ENABLE_AFFINITY=0 MV2_USE_CUDA=1 MV2_USE_GPUDIRECT=1 MV2_NUM_PORTS=2 MV2_GPUDIRECT_LIMIT=524288 MV2_USE_GPUDIRECT_GDRCOPY=1 MV2_USE_GPUDIRECT_GDRCOPY_LIMIT=16384 MV2_USE_GPUDIRECT_GDRCOPY_NAIVE_LIMIT=16384 numactl –cpunodebind=1 –localalloc

表 4　Broadcast 実装の通信パターン（8 プロセス使用時）

\ Rank	0	1	2	3	4	5	6	7
通信前	x							
Step 1	$x \to 4$							
Step 2	$x \to 2$				$x \to 6$			
Step 3	$x \to 1$		$x \to 3$		$x \to 5$		$x \to 7$	
通信後	x	x	x	x	x	x	x	x

通信の開始命令を 1 回だけ発行すれば済むようにしているので，この Broadcast 通信は

$$T_{Bcast} = \log_2 p \cdot (\alpha n + \beta)$$

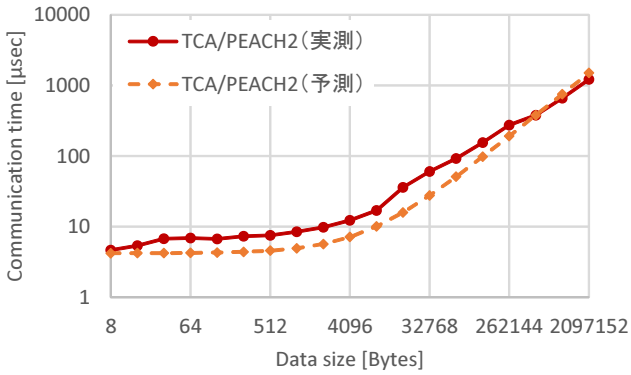
というモデル式により通信時間の予測を試みる．

TCA/PEACH2 による Broadcast 通信の実測時間と式 1 による予測時間を図 **5** に示す．Broadcast 通信の相対誤差は図 **6** に示されるように P2P 通信より大きく，最大で 50%以上の誤差があり予測性能が実測性能より倍以上遅くなる場合がある．モデルは大雑把な性能を知るためであり正確さはそれほど重視しないが，どれくらいの誤差を許容するのかを検討する必要がある．また，Broadcast 通信の場合はノードマッピングを考慮すれば TCA/PEACH2 の 2x8 リングのトポロジにおいても経路衝突なしで通信を実現できるが，Allgather 通信を含む幾つかの Collective 通信では経路衝突は避けられない．その場合には経路衝突も加味した性能モデルを考えることが求められる．

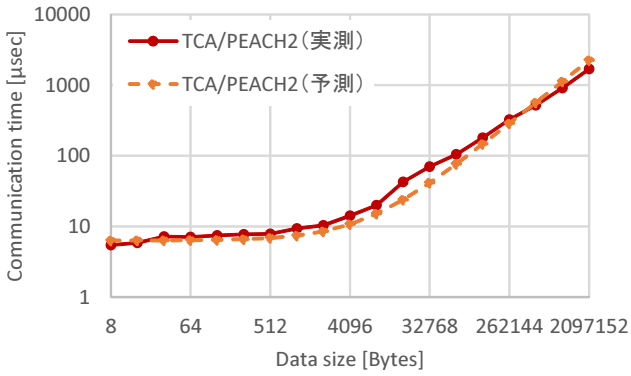
5. GPU へのオフローディング起動と通信周辺のコスト

GPU クラスタにおける並列 GPGPU では，GPU への演算オフローディングと，MPI 等によるノード間通信の両方が必要になる．その際に重要なのは，GPU 演算と通信起動のスイッチングオーバヘッドである．TCA/PEACH2 は NVIDIA 社の CUDA 環境をベースに開発を行っているため，ここでは CUDA 環境においてこのコストを見積もる．

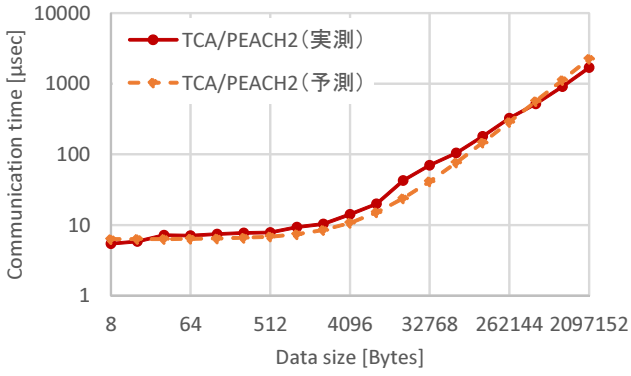
CUDA を用いた並列 GPGPU 処理では，GPU にオフローディングされる処理は CUDA カーネル関数として実装されるが，MPI 等の並列通信は CPU 側でないと実行できないため，最低限，通信が発生する度にカーネル関数を終了して CPU 側に制御を戻す必要がある．通信完了後に再度カーネル関数（先ほどの関数かもしれないし，別の継続するカーネル関数かもしれない）を呼び出す．この処理



(a) p = 4



(b) p = 8



(c) p = 16

図 5　Broadcast 通信の実測時間と予測時間（p はプロセス数を表し，実測時間は 100 回測定の平均）

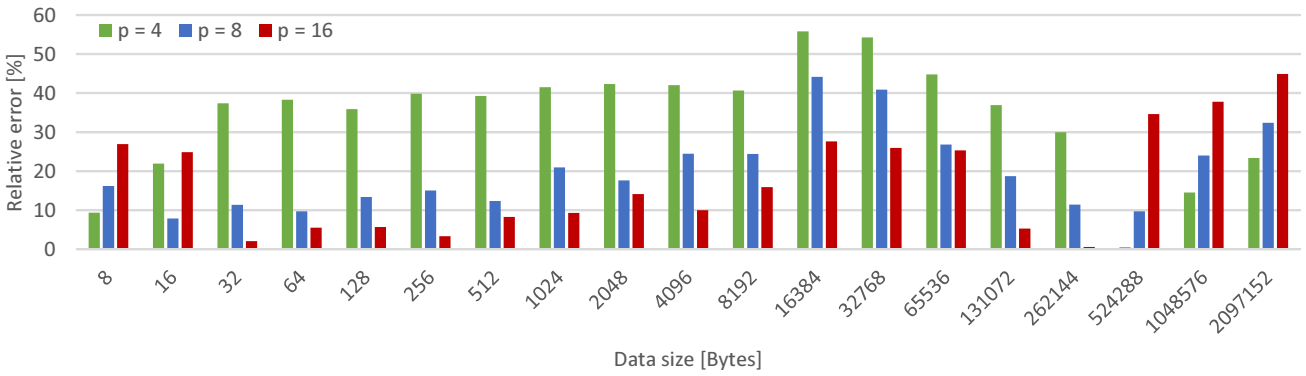


図 6 Broadcast 通信の実測時間に対する予測時間の相対誤差

を繰り返すことにより並列 GPGPU が実現される．従って，カーネル関数の起動コストは，ストロング・スケーリングを目指す並列処理において，カーネル関数内の実行時間が短くなるにつれ大きなオーバヘッドとなり，この見積もりは重要である．

CUDA カーネル関数の起動オーバヘッドを NVIDIA 社の K20X を対象に実測した場合，CUDA 6.5 環境では平均で 3.23 μsec かかることが我々の既存研究でわかっている [14]. 一般的には，このカーネル起動オーバヘッドのみに注意が行きがちだが，上記のような実行モデルでは，カーネル関数終了後，MPI あるいは TCA によるノード間通信が発生する．その際，CPU あるいは PCIe 上の別デバイスから GPU のデバイスメモリの内容が正しくアクセスされるためには，通信を実行する前に `cudaStreamSynchronize()` 等により，同期をとる必要がある．通信のためにカーネル関数から戻ってくるのであれば，このコストとカーネル関数起動コストを合計で考えなければならない．K20X 環境でこれを測定したところ，そのコストはカーネル関数起動コストよりはるかに大きく，平均で 17.2μsec 程度もかかることがわかった．性能モデルを考える際にはこのコストを考慮する必要がある．

6. 性能モデルのアプリケーションへの適用

TCA/PEACH2 における並列 GPGPU 処理の例として，一次元分割による CG 法 [6] を性能モデルにどのように考えて適用していくのかについて考える．CG 法は，対称正定値行列を係数行列とする連立一次方程式 $Ax = b$ を解くための反復法である．本稿において，連立一次方程式は

$$Ax = b$$

と記す．ここで A は $N \times N$ の対称正定値行列であり，x および b は N 次元ベクトルである．本研究では，行列データの格納形式は，Compressed Row Storage (CRS) 形式（CSR: Compressed Sparse Row 形式とも呼ばれる）[15] を用いる．浮動小数点演算は倍精度の実数に対して行う．なお，CG 法は前処理を行うことで収束性能を高められる可能性があるが，ここでは省略する．

CG 法を一次元分割により並列化するために，疎行列 A を行方向にほぼ均等にプロセス数でデータ分割し，かつベクトル x, b も同割合で均等に分割し各プロセスに初期データとして持たせる．つまり，プロセス数を p と記述し $n = \lfloor N/p \rfloor$ とするとき，各プロセスは $n \times N$ の A の部分行列および n 次元の b と x の部分ベクトルを持つ（ただし最大ランクのプロセスは $(N - (p-1)n) \times N$ 行列および $(N - (p-1)n)$ 次元ベクトルを持つ）．このようにデータ分割を行うことにより，CG 法の並列アルゴリズムは図 7 のように記述できる．

CG 法の主な演算は，疎行列ベクトル積計算（SpMV: Sparse Matrix-Vector multiply），内積計算（DOT product），ベクトル加算（AXPY）である．**図 7** のアルゴリズムは毎反復において（$k \geq 2$），1 回の SpMV（図 7 の行 15），3 回の DOT（行 6, 16, 20[*1]），3 回の AXPY（行 12, 18, 19）を行う．これらの行列とベクトルに対する 3 つの演算は基本的な演算であり，CUDA による NVIDIA 社の数値計算ライブラリでも提供されている．SpMV は cuSPARSE ライブラリ [16] に，DOT と AXPY は cuBLAS ライブラリ [17] にそれぞれ cusparseDcsrmv, cublasDdot, cublasDaxpy ルーチンとして実装されている．本研究では，各 GPU 内の処理ではそれらの cuSPARSE と cuBLAS ルーチンを利用する．

各反復において，図 7 の並列アルゴリズムは，以下のようなデータ通信が必要である．

- SpMV 計算 (図 7 の行 15) を行う前に，全プロセスが各プロセスに均等に分散されているベクトルデータ p_l を集める必要がある (Allgather).
- プロセス毎の DOT 計算（行 6, 16, 20）の後に，そのローカルなベクトル内積の総和を計算し，全プロセスがその総和を持つ必要がある (Allreduce).

この Collective 通信に Allgather 実装と Allreduce 実装を

[*1] ベクトルの 2-ノルムは内積計算を用いて計算する．

1: $\boldsymbol{x} := \mathrm{Allgather}(\boldsymbol{x}_l)$
2: $\boldsymbol{r}_l := \boldsymbol{b}_l - \boldsymbol{A}_l\boldsymbol{x}$
3: $d_t := \boldsymbol{r}_l^T\boldsymbol{r}_l$
4: $norm0 := \mathrm{sqrt}(\mathrm{AllreduceSum}(d_t))$
5: **for** $k := 1, 2, \cdots$ **do**
6: $\quad\rho_t := \boldsymbol{r}_l^T\boldsymbol{r}_l$
7: $\quad\rho := \mathrm{AllreduceSum}(\rho_t)$
8: $\quad$**if** $k = 1$ **then**
9: $\quad\quad\boldsymbol{p}_l := \boldsymbol{r}_l$
10: $\quad$**else**
11: $\quad\quad\beta := \rho/\rho\mathrm{prev}$
12: $\quad\quad\boldsymbol{p}_l := \beta\boldsymbol{p}_l + \boldsymbol{r}_l$
13: $\quad$**end if**
14: $\quad\boldsymbol{p} := \mathrm{Allgather}(\boldsymbol{p}_l)$
15: $\quad\boldsymbol{q}_l := \boldsymbol{A}_l\boldsymbol{p}$
16: $\quad\alpha_t := \rho/(\boldsymbol{p}_l^T\boldsymbol{q}_l)$
17: $\quad\alpha := \mathrm{AllreduceSum}(\alpha_t)$
18: $\quad\boldsymbol{x}_l := \alpha\boldsymbol{p}_l + \boldsymbol{x}_l$
19: $\quad\boldsymbol{r}_l := -\alpha\boldsymbol{q}_l + \boldsymbol{r}_l$
20: $\quad d_t := \boldsymbol{r}_l^T\boldsymbol{r}_l$
21: $\quad norm := \mathrm{sqrt}(\mathrm{AllreduceSum}(d_t))$
22: $\quad$**if** $norm/norm0 < \varepsilon$ **then**
23: $\quad\quad$break
24: $\quad$**end if**
25: $\quad\rho\mathrm{prev} := \rho$
26: **end for**

図 7 CG 法の並列アルゴリズム．各変数の下付き文字 “l” および “t” は，プロセス毎にローカルに持つ部分データおよび一時データであることをそれぞれ表す．

利用する．Allgather は各プロセスが持っているデータブロックを他のプロセスとやりとりし，Allreduce は倍精度の場合は 8 バイトという非常に少量のデータを他プロセスとやりとりする．以上の特徴から，Allgather は TCA の GPU 間 DMA 通信による実装を利用し，Allreduce は TCA の CPU 間 PIO 通信による実装を利用する [*2]．

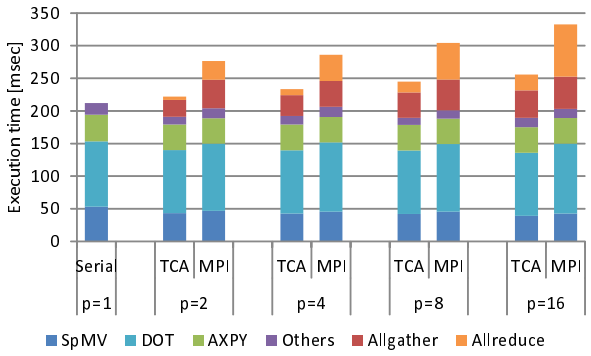
CG 法実装を The University of Florida Sparse Matrix Collection[18] から取得した**表 5** に示す疎行列に対して評価を行うと，それぞれの処理の処理時間の内訳は**図 8** のようになる．この内訳はプロセスランク番号 0 の結果で，TCA/PEACH2 を通信に用いた場合の結果と MPI/IB を通信に用いた場合の結果を併記している．なお，実際の使用において，CG 法は解が収束するまで反復する必要があるが，反復回数を 1000 回に固定して性能を測定した結果である [*3]．

3 つの行列の中でサイズが一番小さい行列（nasa2910）の場合では，演算量が少なすぎるために並列化しても演算時間をほとんど削減できていない．プロセス数によらず演算時間に一定になってしまっており，節 5 で述べたオーバヘッド・コストによる時間がほとんどを占める．最も大きな行列（nd24k）に関しては，並列化により演算時間を削減することができている．しかし，その演算時間が削減は SpMV によるものが主であり，他の AXPY と DOT に関してはほとんど演算時間を減っていない．このことから，それぞれの演算に対して演算密度や演算特性を理解した上で適切なモデルを構築して実測に対するフィッティングを

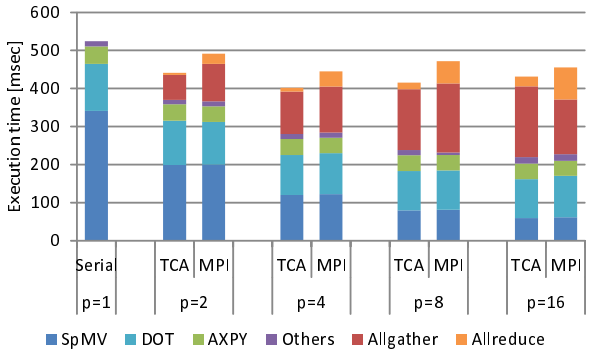
[*2] cublasDdot は計算したローカルな内積を CPU 側にも返すことができるので，それにより返されたスカラー値を CPU 間 Allreduce を利用して内積を計算する．
[*3] 本研究は CG 法における 1 反復当たりの処理時間の評価を目的としており，収束するか否かは問題としない．性能評価における反復当たりのバラ付きによる誤差をなくすための十分な反復回数として 1000 回を選んだ．

表 5 性能評価に用いた疎行列の特性

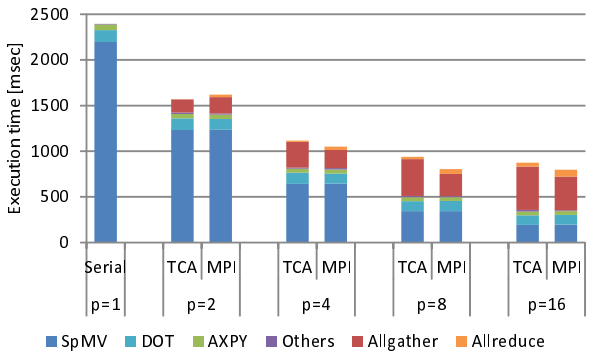
行列名	行数（N）	非零要素数（nnz）	nnz/N
nasa2910	2,910	174,296	59.9
smt	25,710	3,753,184	146.0
nd24k	72,000	28,715,634	398.8



(a) nasa2910



(b) smt



(c) nd24k

図 8 CG 法実装の各処理実行時間の内訳 (ランク 0 の内訳)

行っていく．

この CG 法実装の通信パターンとしては，Allreduce にかかる時間は行列サイズによらず一定であるが，Allgather は行列サイズが大きくなるとその通信時間が増加する．そのため，小さいサイズの行列に対しては TCA/PEACH2 は性能的に有利であるが，サイズが大きくなると MPI/IB に性能で逆転される．この逆転されるサイズを性能モデルにより予測できるようにする．節 4 のようにモデル式を構築しそれにより通信時間の予測を行う．先行研究において [6]，TCA/PEACH2 を用いて Allgather に関しては Recursive Doubling 法，Allreduce は Dissemination 法で実装が実現済みであるが，これらは使用ノード数を増やしたとき通信経路で衝突を起こす場合がある方法である．この 2 つの通信に関するモデルは詳細なデータに基づいて構築を行っているところである．

以上の演算時間と通信時間に関する性能に対するモデルの適用により，実際に TCA/PEACH2 がストロング・スケーリングという観点において有利となる問題サイズを明らかにする．

7. おわりに

本研究は TCA による低レイテンシ通信が性能向上に貢献するのは，どのようなアプリケーションにおいてどのような問題サイズの範囲においてなのかを明確にすることを目的としている．本稿では，そのために構築を試みている性能予測モデルについて現時点で検討していることを記述した．現状では性能予測モデルをアプリケーションに対して実際に適用するところまでは至っておらず，今後，CG 法等のこれまで TCA/PEACH2 で実装されたアプリケーションを対象にモデルを完成させていく予定である．

謝辞 本研究の一部は JST-CREST 研究領域「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」，研究課題「ポストペタスケール時代に向けた演算加速機構・通信機構統合環境の研究開発」による．また，HA-PACS/TCA システムの利用は筑波大学計算科学研究センター学際共同利用プログラム（課題名「密結合演算加速機構アーキテクチャに向けた GPGPU アプリケーション」）による．

参考文献

[1] 塙 敏博，児玉祐悦，朴 泰祐，佐藤三久：Tightly Coupled Accelerators アーキテクチャに基づく GPU クラスタの構築と性能予備評価，情報処理学会論文誌. コンピューティングシステム， Vol. 6, No. 3, pp. 14–25 (2013).

[2] Hanawa, T., Kodama, Y., Boku, T. and Sato, M.: Tightly Coupled Accelerators Architecture for Minimizing Communication Latency among Accelerators, *Proceedings of the 27th IEEE International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW 2013)*, IEEE, pp. 1030–1039 (2013).

[3] 朴 泰祐，佐藤三久，塙 敏博，児玉祐悦，高橋大介，建部修見，多田野寛人，蔵増嘉伸，吉川耕司，庄司光男：演算加速装置に基づく超並列クラスタ HA-PACS による大規模計算科学，情報処理学会研究報告，Vol. 2011-HPC-130, No. 21, pp. 1–7 (2011).

[4] 藤井久史，藤田典久，塙 敏博，児玉祐悦，朴 泰祐，佐藤三久，藏増嘉伸，Clark, M.：GPU 向け QCD ライブラリ QUDA の TCA アーキテクチャ実装の性能評価，情報処理学会研究報告，Vol. 2014, No. 43, pp. 1–9 (2014).

[5] Matsumoto, K., Hanawa, T., Kodama, Y., Fujii, H. and Boku, T.: Implementation of CG Method on GPU Cluster with Proprietary Interconnect TCA for GPU Direct Communication, *Proceedings of the 29th IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW 2015)*, IEEE, pp. 647–655 (2015).

[6] 松本和也，塙 敏博，児玉祐悦，藤井久史，朴 泰祐：密結合並列演算加速機構 TCA を用いた GPU 間直接通信による Collective 通信の実装と性能評価，2015 年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 (HPCS2015) 論文集，情報処理学会，pp. 120–128 (2015).

[7] 藤井久史，塙 敏博，児玉祐悦，朴 泰祐，佐藤三久：TCA アーキテクチャによる並列 GPU アプリケーションの性能評価，情報処理学会研究報告，Vol. HPC-140, No. 37, pp. 1–6 (2013).

[8] 藤井久史，塙 敏博，児玉祐悦，朴 泰祐：GPU 向け FFT コードの TCA アーキテクチャによる実装と性能評価，情報処理学会研究報告，Vol. HPC-148, No. 12, pp. 1–9 (2015).

[9] Kodama, Y., Hanawa, T., Boku, T. and Sato, M.: PEACH2: An FPGA-based PCIe network device for Tightly Coupled Accelerators, *ACM SIGARCH Computer Architecture News*, Vol. 42, No. 4, pp. 3–8 (2014).

[10] Panda, D. K.: MVAPICH2-GDR (MVAPICH2 with GPUDirect RDMA), The Ohio State University (online), available from ⟨http://mvapich.cse.ohio-state.edu/overview/⟩ (accessed Jul 9, 2015).

[11] NVIDIA: NVIDIA GPUDirect, (online), available from ⟨https://developer.nvidia.com/gpudirect⟩ (accessed Jul 9, 2015).

[12] NVIDIA Corporation: GDRCopy, GitHub (online), available from ⟨https://github.com/NVIDIA/gdrcopy⟩ (accessed Jul 9, 2015).

[13] Grama, A., Karypis, G., Kumar, V. and Gupta, A.: *Introduction to Parallel Computing*, Addison-Wesley, 2nd edition (2003).

[14] 桑原悠太，塙 敏博，児玉祐悦，朴 泰祐：GPU クラスタにおける GPU 間セルフ通信機構に関する提案，情報処理学会研究報告，Vol. HPC-148, No. 17, 情報処理学会，pp. 1–8 (2015).

[15] Saad, Y.: *Iterative Methods for Sparse Linear Systems*, SIAM, 2nd edition (2003).

[16] NVIDIA: cuSPARSE Library, (online), available from ⟨http://docs.nvidia.com/cuda/cusparse/index.html⟩ (accessed Jun 14, 2015).

[17] NVIDIA: cuBLAS Library, (online), available from ⟨http://docs.nvidia.com/cuda/cublas/index.html⟩ (accessed Jun 14, 2015).

[18] Davis, T. A. and Hu, Y.: The University of Florida Sparse Matrix Collection, *ACM Transactions on Mathematical Software*, Vol. 38, No. 1, pp. 1:1–1:25 (online), available from ⟨http://www.cise.ufl.edu/research/sparse/matrices⟩ (2011).