Vol.2018-HPC-167 No.16
2018/12/17

AI橋渡しクラウドABCIにおけるLinpack benchmarkの最適化と性能評価

本田　巧[1]　笠置　明彦[1]　福本　尚人[1]　大辻　弘貴[1]　土肥　義康[1]　田原　司睦[1]　中島　耕太[1]

概要：近年のHPCシステムは, HPCアプリケーションに加え, 高い演算性能が必要なディープラーニングへの需要も増加している. そのため, 高い演算性能と電力効率を有するGPUを搭載したノードで構成されるPCクラスタが増加している. GPUの搭載でノードの演算性能が大幅に向上した一方で, ノード間通信に用いられるInfiniBandやCPU-GPU間のデータ転送に用いられるPCIeは相対的に性能向上が緩やかであるため, システム全体の性能バランスの観点では, 通信性能がボトルネックになりやすい. そのため, GPUクラスタにおける大規模実行では通信処理の最適化が重要になる. 本論文では, 最新GPUクラスタ向けに開発したHPLと通信の最適化, また, 大規模GPUクラスタにおけるLinpackベンチマークの特性について報告する. 本研究の性能評価は, 4352基のNVIDIA Tesla V100を有するGPUクラスタ “AI橋渡しクラウド (ABCI)” で行った. 最適化を行ったHPLを用いて性能評価を行ったところ, 全ノードの実行で19.88PFLOPSの性能を達成し, ABCIは2018年6月のTop500において世界5位の実行性能を達成した.

Performance evaluation and optimization of Linpack benchmark on ABCI

Takumi Honda[1]　Akihiko Kasagi[1]　Naoto Fukumoto[1]　Hiroki Ohtsuji[1]　Yoshiyasu Doi[1]
Tsuguchika Tabaru[1]　Kohta Nakashima[1]

Abstract: The demand for improving computing power increases in not only HPC field but also DL field. Thus, recent PC clusters are equipped with GPUs which have high computing power and energy efficiency. In contrast, communication performance such as InfiniBand and PCIe has been improved slowly, compared with computing power, and the data communication easily becomes a bottleneck in a case of execution of applications on a large GPU clusters. In this paper, we report optimization techniques which are applied for improving Linpack performance and the performance evaluation on ABCI (*AI Briding Cloud Infrastracture*). ABCI is a petascale GPU cluster which has 4352 NVIDIA Tesla V100 and 2176 Intel Xeon Gold 6148. Our HPL achieved 19.88 PFLOPS, and the score made ABCI the fifth fastest supercomputer on Top500 in June, 2018.

1. はじめに

従来のシミュレーションなどのアプリケーションだけでなく, 近年注目を集めているディープラーニング分野でもニューラルネットワークの学習に膨大な計算量が必要となっている. 高い翻訳精度を達成しているGoogle's Neural Machine Translationでは, ネットワークの学習に96基のNVIDIA Tesla K80を用いて6日を要しており, より高い演算性能を有したHPCシステムによる処理の高速化の需要は高まっている.

一方で, 近年構築されるHPCシステムは, 価格性能比が良いこともあり, 多数のサーバを高速なネットワークで結合するPCクラスタが主流になっている. PCクラスタでは, 高演算性能と電力効率のメニーコアアクセラレータGPU (Graphics Processing Unit) を搭載したGPUクラスタが増加している. この傾向は, HPCシステムの世界ランキングTop500[1]にも表れており, 2018年6月に発表されたリストでは, 上位10システムのうち5つのシステムが

[1] 株式会社富士通研究所
Fujitsu Labratories Ltd.

GPU クラスタであり, 新たに増えた総演算性能の約 56%が GPU によって得られたものである. 2018 年 6 月の Top500 で新たに国内トップの HPC システムになった ABCI (AI 橋渡しクラウド)[2] も, 4352 基もの NVIDIA Tesla V100 を搭載した国内最大の GPU クラスタである.

本論文では, 最新 GPU を搭載した GPU クラスタ向け HPL (High-Performance Linpack)[3] と, ABCI における HPL の性能向上のために適用した最適化技術を報告する. Linpack ベンチマークは, 密行列を係数行列とする連立一次方程式を解くアプリケーションであり, 性能指標として Top500 でも用いられている. 本研究では, CPU クラスタ向けの Linpack の MPI 並列実装である HPL をベースに, GPU クラスタ向けの HPL を開発した. GPU クラスタ向け HPL の最適化として, プロセス間通信と CPU-GPU 間のデータ転送の最適化を行った. 最適化を行った我々の HPL を用いて ABCI の Linpack 計測を行ったところ 19.88PFLOPS の性能を達成し, ABCI は 2018 年 6 月の Top500 において 5 位にランクインし国内 1 位のシステムとなった.

本論文の構成は以下のとおりである. 2 章では, ABCI の構成について述べる. 3 章では, Linpack ベンチマークの実装の 1 つであり, 本研究で基とした HPL について説明する. 4 章では, 開発した最新 GPU クラスタ向け HPL の実装と最適化手法について説明する. 5 章では, 我々の開発した HPL の評価結果について述べ, 6 章で結論を述べる.

2. ABCI

ABCI (AI 橋渡しクラウド) は, 高性能計算システム, 大容量ストレージシステムで構成されており, 各システムのノード間は InfiniBand により接続されている. 以降では, 本論文に関連の深い高性能計算システムについて述べる.

2.1 高性能計算システム

ABCI の高性能計算システムは, 計算ノード 1088 台, マルチプラットフォームノード 10 台, インタラクティブノード 4 台, 管理サーバ・ゲートウェイノード 15 台などで構成されている [2]. 計算ノードは PRIMERGY CX2570 M4 であり, 計算ノード群は図 1 に示すように 34 ノード搭載したラック 32 台で構成されている.

2.1.1 計算ノード

ABCI の計算ノードは, 図 2 に示すように Intel Xeon Gold 6148 を 2 基搭載しており 384GB のメモリを共有している. アクセラレータとしては, HPC 向けの GPU である NVIDIA Tesla V100 を 4 基搭載している. 図 2 からも分かるように, 同じ CPU に接続されている 2 基の GPU は PCI スイッチを用いて PCIe Gen3 ×16 を共有している. また, 4 基の GPU は互いに 2 本の NVLink2 で接続されており, GPU 間の高速なデータ転送が可能である. CPU と GPU の

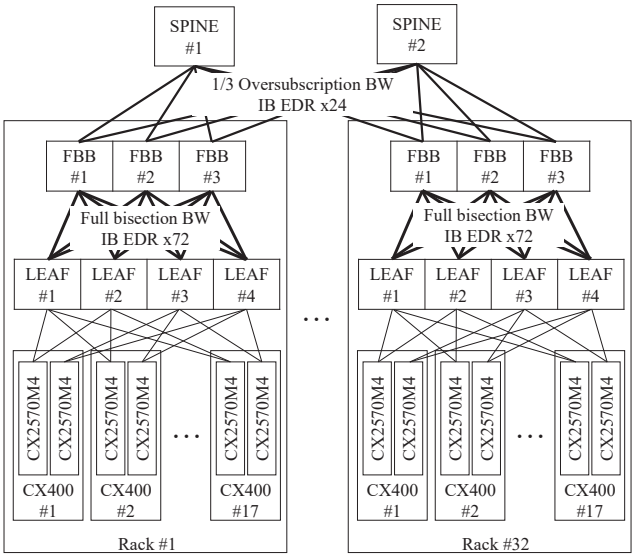

図 1 ABCI の計算ノードのインターコネクト

理論倍精度演算性能はそれぞれ約 1.5TFlops, 7.8TFlops であり, ノードあたりの理論倍精度演算性能は約 34.2TFlops である. また, 計算ノードは 2 つの IB HCA (InfiniBand Host Channel Adapter) を持ち, 2 本の InfiniBand EDR が異なる Leaf スイッチに接続されている.

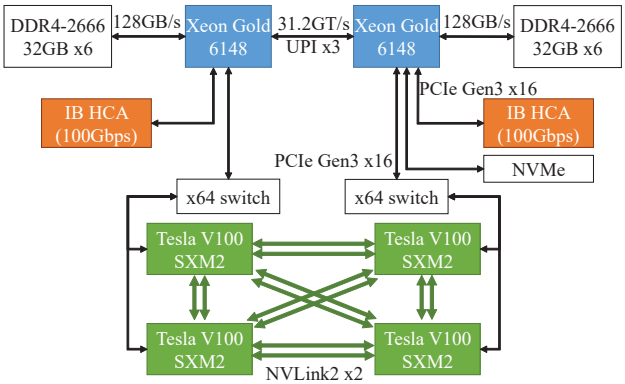

図 2 CX2570M4 の構成

2.1.2 計算ノードのインターコネクト

同一ラック内の計算ノード間は, 図 1 に示すように 7 つのスイッチによる 2 段構成でフルバイセクションのバンド幅 7200Gbps 相当で接続されている. 一方で, 異なるラック間のスイッチの接続はオーバーサブスクリプション構成であり, ラック間の計算ノード間は 1/3 である 2400Gbps 相当のバンド幅である.

3. HPL

本章では Linpack ベンチマークの MPI 並列実装である HPL について説明する. Linpack ベンチマークは, 非対称実行列を係数行列とする連立一次方程式をガウスの消去法を用いて解くアプリケーションで, Top500 に用いられるベ

ンチマークであり, HPC システムの演算性能測定に広く用いられている. HPL は, HPC システムなどの分散メモリ型並列計算機向けに作られた MPI 並列実装の Linpack ベンチマークプログラムである. HPL では $N \times N$ の係数行列を乱数で生成し, $P \times Q$ プロセスによる分散並列処理で連立一次方程式を解く. HPL の計算は, LU 分解と後退代入操作で構成されており, 計算時間の大半を LU 分解の計算時間が占めるため, LU 分解処理の高速化が HPL の高速化につながる.

3.1 シングルプロセスの LU 分解の流れ

はじめに, シングルプロセス ($P \times Q = 1$) での LU 分解処理について説明する. HPL の LU 分解は, 列パネル分解, 行交換操作, 行パネル更新, 残行列更新の 4 つの操作で構成されている. これらの操作の流れを図 3 に示す.

列パネル分解では, 入力行列の最も左の NB 列 (列パネル) の LU 分解を行う. ブロックサイズ NB は, 入力パラメータとして与えられる任意の値である. この操作は, ピボット選択, 行交換, 行列積など処理で構成されている. LU 分解後の列パネルのうち, 上部の $NB \times NB$ の行列を行列 $\boldsymbol{U}$, 残りの行列を行列 $\boldsymbol{A}$ とする.

行交換操作では, 列パネル分解のピボット選択で行った行交換を列パネル以外の部分行列に対して行う. この操作により, 行列全体でピボット選択による行の入れ替えが完了する.

行パネル更新では, 最も上部の NB 行 (行パネル) に対して, 行列 $\boldsymbol{U}$ を係数行列とする連立一次方程式を解く. この操作により, 行パネル (行列 $\boldsymbol{B}$) の LU 分解が完了する. 一般的に, 行パネル更新には, 数値計算ライブラリである BLAS (Basic Linear Algebra Subprograms)[4] の倍精度三角行列ソルバー (DTRSM) を用いる.

残行列更新では, 列パネル分解に伴う更新を残行列 $\boldsymbol{C}$ に反映する. 列パネル分解と行パネル更新で得られた行列 $\boldsymbol{A}$, $\boldsymbol{B}$ を用いて, 行列演算 $\boldsymbol{C}-\boldsymbol{AB}$ を計算し, 演算結果で残行列 $\boldsymbol{C}$ を更新する. 残行列更新には, 一般的に, BLAS ライブラリの倍精度行列積 (DGEMM) が用いられる.

これらの 4 つの操作を行うと, 図 3 に示すように列/行パネルの LU 分解が完了する. 得られた行列 $\boldsymbol{C}$ を次の入力行列とし, 同様の操作を繰り返し適用することで行列全体の LU 分解が完了する.

列パネル分解, 行パネル更新, 残行列更新のそれぞれの総計算量は表 1 に示すようになっており, $NB \ll N$ であるため残行列更新が LU 分解の計算時間の大半を占める. そのため, 残行列更新の行列積計算の性能が LU 分解の演算性能を大きく左右する [5].

3.2 マルチプロセスの LU 分解の流れ

マルチプロセスの場合, 生成されたプロセスは $P \times Q$ の 2 次元のプロセス格子を形成する. 図 4 に 3×3 のプロセス格子と各プロセスが保持する部分行列の例を示す. 図 4 の $P_i (i = 0, 1, ..., 8)$ はプロセスを表す. ガウスの消去法による解法では, 行列の左上から順に LU 分解が完了するため, 単純に入力行列をプロセス数で分割すると, プロセス毎の計算量が大きく異なりロードインバランスが発生してしまう. このロードインバランスを緩和させるため, HPL では図 4 のように分割した $NB \times NB$ の小行列をサイクリックに各プロセスに割り当てている. 図 4 の各小行列内の数字は小行列を保持するプロセス番号を表している.

表 1 各操作の計算量

操作	計算量
列パネル分解	$O(N^2 \times NB)$
行パネル更新	$O(N^2 \times NB)$
残行列更新	$O(N^3)$

マルチプロセスでの LU 分解も, シングルプロセスと同様の操作であるが, 入力行列が複数のプロセスで分散保持されているため, 2 種類のプロセス間のデータ通信が発生する.

1 つは, 列パネル分解と行交換操作の Allgather 通信である. 行パネルを構成する行データは同じ列に属するプロセスに分散している. 行パネルは全てのプロセスの残行列更新で必要となるため, Allgather 通信で全てのプロセスが行パネルを構成する行データを得る.

もう 1 つは, 列パネル分解結果のブロードキャストである. 列パネル分解を行うのは行列の一番左の列パネルを保持する 1 列分のプロセスのみである. しかし, 全てのプロセスが行パネル更新や残行列更新で列パネルの情報が必要となるため, 列パネル分解を行ったプロセスが, 同じ行に属するプロセスに結果をブロードキャストする. HPL では, リングアルゴリズム [6] に基づいた複数のブロードキャストが実装されており計測するシステムに適した実装を選択できるようになっている.

この 2 種類の通信の総通信量は $O(N^2(P+Q))$ である [7]. 総計算量は $O(N^3)$ であるため, 行列サイズ N が大きくなるほど HPL の実行時間のなかで通信が占める割合が小さくなり実行効率が向上する傾向がある.

4. 最新 GPU クラスタ向け HPL

本章では, 開発した GPU クラスタ向け HPL について説明する. はじめに, 行列データの格納場所について述べ, プロセスの処理の流れと適用した最適化手法について述べる.

4.1 行列データの格納場所

ABCI の計算ノードは CPU と GPU を搭載しており, CPU メモリと GPU メモリの 2 種類のメモリがある. そのため, 入力行列を格納するメモリの選択によって HPL は 2

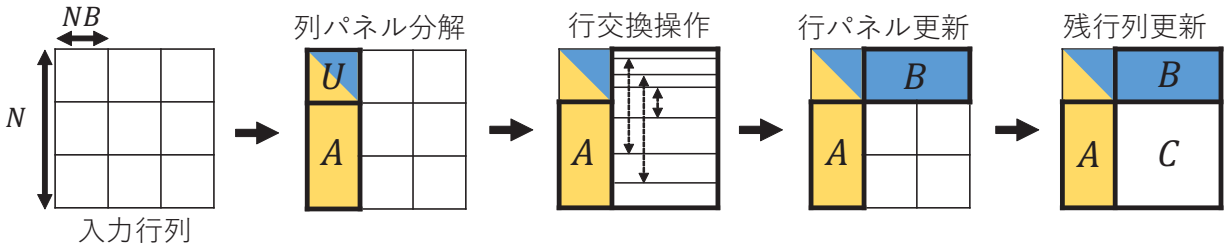

図 3　HPL の LU 分解処理の流れ

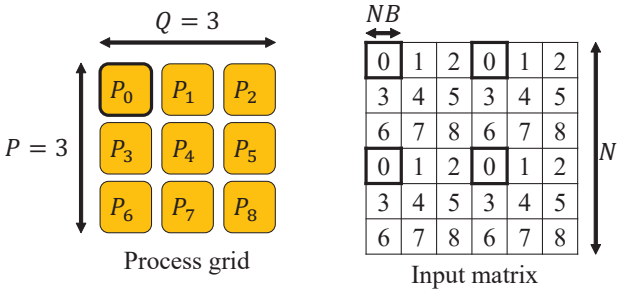

図 4　プロセス格子と行列の割り当て

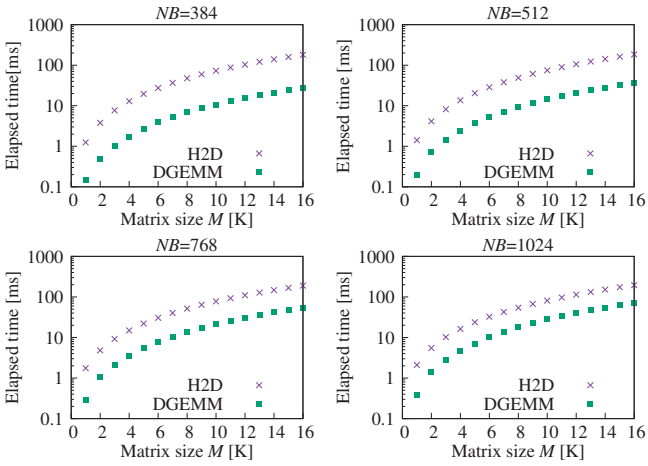

図 5　Tesla V100 における DGEMM の計算時間とデータ転送時間の比較

種類に分類される.

CPU メモリに入力行列を格納する場合, CPU メモリのサイズが行列サイズを制限する. ABCI のノードの CPU メモリは GPU メモリより大きいため, CPU メモリに格納することで行列サイズを大きくすることができる. 一方で, ノードの演算性能の約 9 割が GPU の演算性能によるものであり, 最も計算量が多い残行列更新を GPU で行わなければ GPU の演算性能を活かすことができない. しかし, GPU で処理を行うためにはデータを GPU メモリに転送する必要がある. そのため, CPU メモリ上に GPU メモリサイズを超える入力行列を格納した場合, GPU での残行列更新の度に多くのデータを転送しなければならず, データ転送に使われる PCIe への負荷が高くなりボトルネックとなりやすい.

GPU メモリに入力行列を格納する場合, 行列サイズは GPU メモリのサイズで制限される. GPU メモリのサイズは CPU メモリより小さいため, 行列サイズを大きくすることによる実行効率向上を図ることができない. また, 通信時間が全体に占める割合が多くなるため, 通信部分の最適化が重要となる. しかし, GPU メモリ上に入力行列があるため, 残行列更新に伴うデータ転送は CPU メモリに行列を格納する場合と比較し少なくなり PCIe への負荷を軽減することができる.

入力行列を格納するメモリを決定するために, CPU メモリに入力行列を格納した場合の残行列更新のデータ転送時間と GPU の DGEMM の計算時間を比較する. 評価には, ABCI の計算ノードと同じ構成のサーバを用いた. ソフトウェア環境は CUDA 9.1[8] を利用し, BLAS ライブラリは cuBLAS 9.1[9] を用いた. 残行列更新では, $M \times NB$ 行列の行列積を計算する. また, CPU メモリに入力行列が格納されている場合は 2 つの $NB \times M$ 行列と $M \times M$ 行列のデータ転送が必要となる. ここで M は残行列の一辺を表す. ブロックサイズである NB は, 列パネル分解やデータ通信と残行列更新のロードバランスを考慮し, 384 や 512 など行列サイズと比較して小さい値が採用されることが多い. そこで, $NB = 384, 512, 768, 1024$ における転送時間と計算時間を評価する. 評価実験の結果を図 5 に示す. 図 5 の H2D は 2 つの $NB \times N$ 行列 A, B と $M \times M$ 行列 C の CPU メモリから GPU メモリへの転送時間を表している. DGEMM は cuBLAS の DGEMM ルーチンによる行列演算 $AB + C$ の計算時間を表している. 図 5 より, NB と M の値に関係なく DGEMM の計算時間よりデータ転送時間が大きくなっている. この結果から, CPU メモリに入力行列を格納し残行列更新の度にデータ転送を行う場合, PCIe がボトルネックとなることは明らかである. また, 図 2 より ABCI の計算ノードでは 2 基の GPU が PCI スイッチを介して PCIe を共有しているため, 複数 GPU にデータ転送を行う場合はより多くの転送時間が必要になる. よって, GPU メモリに入力行列を格納する方針を採用する.

4.2　各プロセスの処理の流れ

我々の HPL では, 1 つのプロセスが 1 つの GPU を管理

するため, GPU 数とプロセス数 ($P \times Q$) が等しくなる. 全てのプロセスが同様の操作を実行するため, 以降では 1 プロセスの動作について説明する.

表 2　HPL の各操作と割り当て

列パネル分解	CPU
ブロードキャスト	CPU
行交換操作	CPU & GPU
行パネル更新	GPU
残行列更新	GPU

各プロセスは, 列パネル分解, ブロードキャスト, 行交換操作, 行パネル更新, 残行列更新の 5 つの操作を繰り返し実行していく. 入力行列は GPU メモリにあるため, CPU-GPU 間のデータ転送が増加しないように各操作を表 2 に示すように CPU と GPU で分担する. GPU が計算量が多い行パネル更新と残行列更新を担当し, CPU は GPU のタスク管理とプロセス間通信が含まれる列パネル分解, 行交換操作を担当する.

4.2.1　列パネル分解

列パネル分解は, 比較的小さな行列積計算と同じ列のプロセス間でのデータ通信を再帰的に実行する. 計算量の小さな処理とデータ通信が繰り返されるため, 列パネル分解は列パネルを CPU メモリ上のバッファにコピーし CPU で実行する.

4.2.2　ブロードキャスト

列パネル分解後, プロセス格子の各行で分解結果のブロードキャストを行う. ブロードキャストでは, HPL に実装されているリングアルゴリズムを用いている. 列パネル分解を行ったルートプロセスからいくつかのプロセスにデータが送られ, 受信したプロセスはさらに次のプロセスにデータを送る. また, 列パネル分解の結果は GPU で行う残行列更新で必要になるため, 各プロセスは受信後に GPU にデータ転送を行う.

4.2.3　行交換操作

行交換操作は, 図 6 に示すように行パネルを構成する行データの抽出とプロセス間のデータ通信の 2 ステップで実行される. はじめに, 各プロセスは GPU メモリ上の行列データから行パネルを構成する行を抽出し CPU メモリ上のバッファに転送する. 各プロセスの行データの抽出が完了すると, 行パネルのデータが同じ列の各プロセスの CPU メモリ上に分散していることになる. 行パネルは残行列更新に必要となるため, 同じ列のプロセス間で Allgather 通信を実行し, 同じ列の全てのプロセスが行パネルのデータを得る.

4.2.4　行パネル更新

各プロセスが行パネルの複製を持つため, 全てのプロセスが行パネル更新を実行する. CPU メモリ上にある行パネルを GPU に転送し, GPU で更新を行う. 更新が完了すると, 残行列更新に必要なデータが GPU 上に揃うことになる.

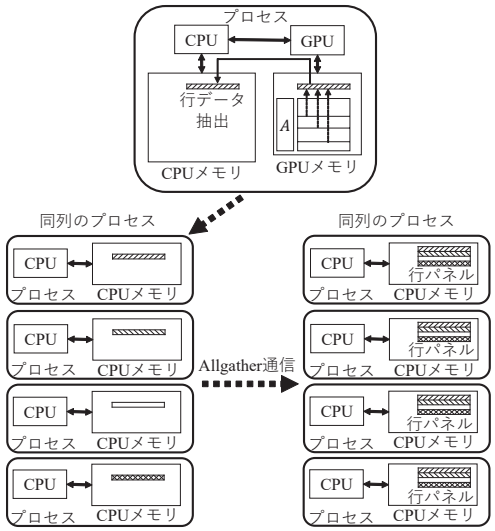

図 6　行交換操作の流れ

4.2.5　残行列更新

残行列更新は, GPU メモリ上にある列パネルと行パネル, 残行列による DGEMM を行うことで完了する. 更新した行列データは GPU メモリ上にあるため, 次の反復で列パネル分解を行うプロセスは CPU メモリに列パネル部分のデータを転送する.

4.2.6　残行列更新のブロック化

一度に全ての残行列を更新する場合, 行交換操作の Allgather 通信が完了するまで GPU の行パネル更新と残行列更新を行うことができない. そのため, 図 7 a に示すように行交換操作の Allgather 通信を実行している間 GPU はアイドル状態になってしまい性能低下につながる. この問題を解決するために, 文献 [10] でも採用されている残行列更新のブロック化を行う. 図 7 b に示すように, 残行列を一定列ずつのブロックに分割しブロック毎に残行列更新を行う. ブロック毎に行交換操作, 行パネル更新, そして, 残行列更新を行うことで, 残行列更新と次のブロックの行交換操作をオーバーラップさせ通信時間を隠蔽する.

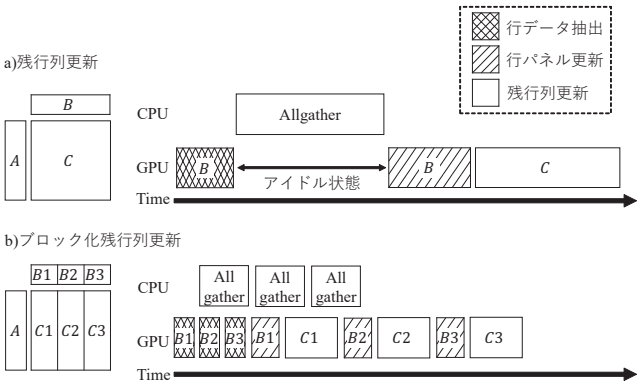

図 7　ブロック化残行列更新の流れ

4.3 通信の最適化

我々の HPL では, GPU メモリに入力行列を格納するため行列サイズが比較的小さくなり, プロセス間通信時間が占める割合が大きくなる. また, ABCI のような大規模 GPU クラスタでは生成するプロセス数の増加により総通信量が増加するため, 通信の最適化が HPL の性能向上において重要となる.

通信の最適化として, ブロードキャストのパイプライン化を行った. ブロードキャストはリングアルゴリズムで行っており, 列パネル分解を行ったプロセスから最も離れたプロセスはデータを受信するまでに行方向のプロセス数 Q に比例する時間がかかってしまう. 列パネルのデータを受信しなければ, 次の反復の行交換や行パネル更新, 残行列更新を開始することができないため, 残行列更新が完了するまでにデータを受信できなければプロセスは通信待ち状態になり性能低下につながる. そこで, 列パネルのデータを一定サイズに分割しデータ通信をパイプライン化する. 一部のデータを受信すると次のプロセスへのデータ通信を開始できるため, 送信と受信のオーバーラップが可能になる. 列パネルをパイプライン的に送信することで, 全てのプロセスが列パネルを受信するまでにかかる時間を短縮し, 通信待ちによる性能低下を軽減する.

元々の HPL のブロードキャストのデータ通信は, MPI の同期通信の MPI_Send 関数と MPI_Recv 関数で実装されている. 同期通信では, 送信と受信が逐次的に実行されるためパイプライン化による送信と受信のオーバーラップが実現できない. そのため, ブロードキャストのパイプライン化に伴い, 非同期通信の MPI_Isend 関数, MPI_Irecv 関数を用いて非同期化を行った. 通信の非同期化を行ったことにより列パネルのデータ送信と GPU メモリへ転送もオーバーラップさせることができるようになるため転送時間の隠蔽も可能になる.

5. 性能評価

本章では, ABCI における開発した GPU クラスタ向け HPL の性能を報告する. ソフトウェア環境は表 3 に示すとおりである. ABCI の HPL 計測では, MPI ライブラリの導入確認も行ったため 2 種類の OpenMPI[11] を用いた. 5.2 節でグラフ掲載している結果は MPI-1.10.2 を用いた実行の結果である.

5.1 スケーラビリティ評価

4.1, 4.2 節に基づき開発した HPL のスケーラビリティを評価する. 図 8 に異なるノード数における HPL の実行効率を示す. 実行効率は, GPU の理論演算性能 (7.8TFlops) と使用した GPU 数の積を 100%としたときの HPL の達成 Flops の割合である. 本評価で用いる HPL には, 通信の最適化は適用していない. 我々の HPL では, $P < Q$ のプロセス格子が $P > Q$ の格子より性能が高くなる傾向があるため, 図 8 の計測には $P < Q$ となるプロセス格子を採用している. この傾向は, 行交換操作におけるプロセス間通信のレイテンシが関係していると考えられる. $P > Q$ のプロセス格子の場合, 同列のプロセス数が増加するため行交換操作にかかる時間が増加する. 1 回のブロック残行列更新が終わるまでに行交換操作が完了しなければ GPU がアイドル状態になってしまう. そのため, 行交換操作はレイテンシを小さくする必要があり, 列方向のプロセス数が増加するとボトルネックとなりやすい.

表 3 ソフトウェア環境

MPI	OpenMPI-1.10.2, 1.10.7
CUDA	CUDA 9.1
BLAS	Intel MKL 2018 Update 2, cuBLAS 9.1
Compiler	icc 18.0.0, nvcc 9.1.85
Compile options	-fomit-frame-pointer -O3 -funroll-loops
(icc)	-fopenmp -restrict -xCORE-AVX512
Compile options	--compiler-options -fno-strict-aliasing
(nvcc)	--arch=compute_70
	--code=compute_70,sm_70

図 8 より, ノード数が増加すると実行効率が著しく低下することが分かる. 1 ノードの HPL の実行効率は約 68.9%であるのに対して, 図 8 の計測で最もノード数が多い 900 ノード実行の効率は約 45.1%まで低下した. ノード数増加による実行効率低下は, プロセス間通信にかかる時間の増加が原因であると考えられる. 我々の HPL では CPU と GPU が非同期に処理を実行しており, 行交換操作だけでなくブロードキャストも残行列更新等の GPU タスクの実行で隠蔽できるように設計している. しかし, 我々の HPL は行列を GPU メモリに格納するため, 行列サイズは小さくなり通信時間が残行列更新などの計算時間に対して占める割合は大きくりやすい. また, ノード数の増加によりプロセス格子が大きくなることで総通信量も増加する. プロセス格子の形状が $P < Q$ であるため, 行方向のプロセス数が多くなりやすく, ブロードキャストにかかる時間が増加し通信の隠蔽が困難になったと考えられる.

5.2 全ノードでの性能評価

ABCI の全 1088 ノードを用いた異なるプロセス格子による性能の変化と通信の最適化による性能向上の評価を行う. 全ノード実行では GPU の上限周波数を設定している. GPU の周波数設定をしない場合, 周波数のばらつきが大きくなることによるロードインバランスが生じるため, 上限周波数を設定し GPU 間の周波数のばらつきを軽減している.

5.2.1 異なるプロセス格子形状における性能評価

図 9 は異なるプロセス格子における HPL の性能を示している. 本計測においても, 通信の最適化を適用していない

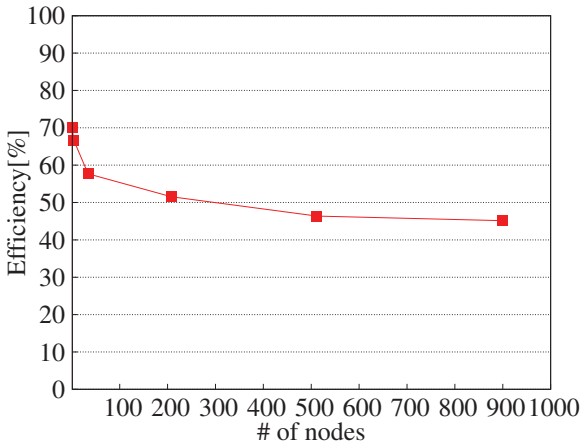

図 8　異なるノード数における HPL の実行効率評価

HPL を用いた．行列サイズ N は 291806, ブロックサイズ NB はこれまでの計測で最も性能が良い 288 を採用した．また, 元々の HPL に実装されているブロードキャストのアルゴリズムに関してもこれまでの計測で最も性能がよい 1ringM を選択した．我々の HPL は, プロセス格子 64×68 のときに 15.63PFlops, 32×136 のときに 18.21PFlops を達成した．プロセス格子 16×272 の実行も行ったが明らかな性能低下がみられたため計測を途中で打ち切った．通信コストが最も小さくなる正方形のプロセス格子で性能が低くなったのは, 5.1 節と同じく行交換操作がボトルネックになったと考えられる．また, プロセス格子 16×272 は行交換操作のレイテンシがより小さくなるが, 総通信量が増加するため性能が低下したと考えられる．これらの結果より, 以降の計測は 32×136 のプロセス格子で行った．

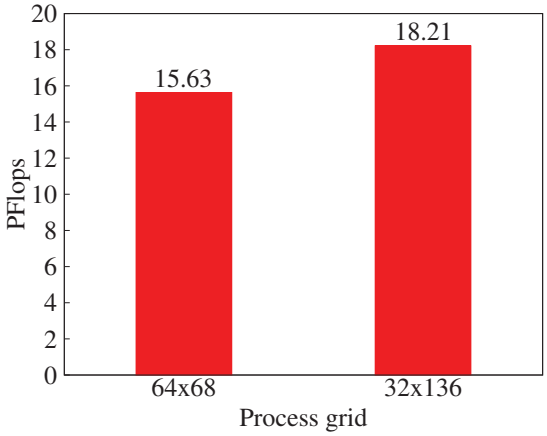

図 9　異なるプロセス格子の性能比較

5.2.2　通信最適化の評価

4.3 節で述べた通信の最適化を適用することでの性能向上を評価する．全ノード実行における通信の最適化の有無による性能の変化を図 10 に示す．行列サイズ, ブロックサイズとブロードキャストアルゴリズムは 5.2.1 節と同じ設定である．通信の最適化適用前の最高性能は図 10 の based-HPL に示す 18.21PFlops であった．一方で, 通信の最適化を適用した場合の性能は図 10 の opt-HPL に示す 19.04PFlops であり, 通信の最適化を適用することで 4.5% の性能向上が得られた．

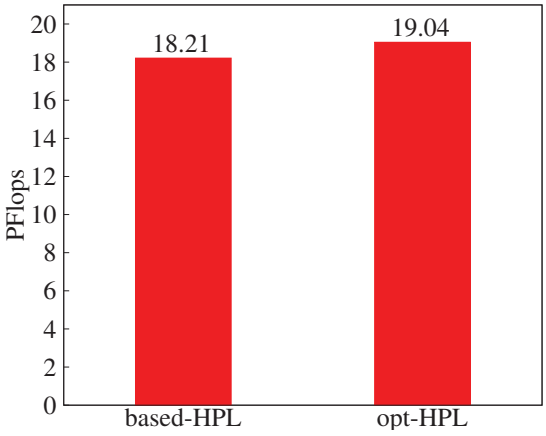

図 10　通信最適化の評価

ブロードキャストの最適化を行ったため, ブロードキャストの実装の違いによる性能比較を行う．図 11 は HPL に実装されているリングアルゴリズムベースのブロードキャスト実装である 1ring, 1ringM, 2ring, 2ringM における我々の HPL の性能を表している．行列サイズとブロックサイズは, 5.2.1 節と同じ設定である．図 11 より, 1ring が 19.34PFlops と最も性能が高く通信の最適化と相性が良かった．

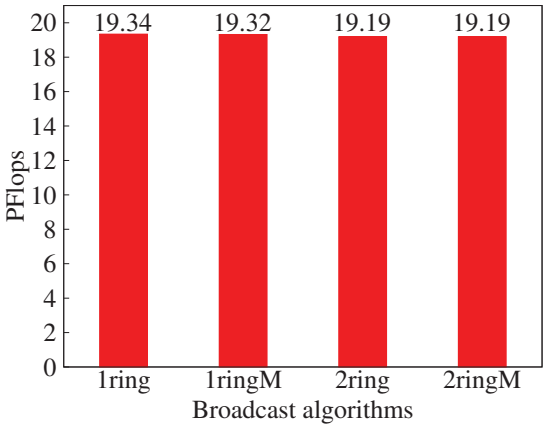

図 11　異なるブロードキャストアルゴリズムの性能比較

ブロードキャストに 1ring を選択し, パイプラインの分割サイズや残行列更新のブロックサイズ等のパラメータ探索を行った．その結果, 我々の HPL は ABCI の全ノード計測において 19.88PFlops の性能を達成した．

6.　おわりに

本論文では, 大規模 GPU クラスタにおける Linpack ベンチマーク性能向上のための最適化技術と ABCI における性能評価の結果について報告した．最新の GPU クラスタ

では, GPU の演算性能の向上に伴い CPU-GPU 間のデータ転送がボトルネックになりやすくなっており, この傾向は演算がボトルネックであると知らている Linpack ベンチマークにおいても起こることを示した. そして, このボトルネックを回避するために GPU メモリを主なデータ格納場所とした GPU クラスタ向け HPL を開発した. また, 大規模 GPU クラスタでの実行において, 我々の HPL はノード間通信が占める割合が大きくなりやすいため, 通信の最適化としてブロードキャストのパイプライン化と非同期化を行った. 我々の HPL は, ABCI の全 1088 ノードによる実行において 19.88PFlops の性能を達成し, 2018 年 6 月の Top500 において ABCI は 5 位にランクインした.

謝辞 本研究における成果は, 国立研究開発法人 産業技術総合研究所の協力のもと, ABCI (AI 橋渡しクラウド) の全系を用いて実施された. また, 東京工業大学学術国際情報センターの額田彰准教授には, 大規模計測に関する貴重なアドバイスを頂きました. ご貢献に感謝申し上げます.

参考文献

[1] TOP500 Supercomputer Sites, https://www.top500.org/.
[2] 小川宏高，松岡 聡，佐藤 仁，高野了成，滝澤真一朗，谷村勇輔，三浦信一，関口智嗣: 世界最大規模のオープン AI インフラストラクチャ AI 橋渡しクラウド（ABCI）の概要，情報処理学会研究報告ハイパフォーマンスコンピューティング（HPC），Vol. 2018-HPC-165, No. 19, pp. 1–7 (2018).
[3] Petitet, A., Whaley, R. C., Dongarra, J. and Cleary, A.: HPL – A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers, http://www.netlib.org/benchmark/hpl/.
[4] BLAS, http://www.netlib.org/blas/.
[5] 成瀬 彰，住元真司，久門耕一: Xeon プロセッサ向け Linpack ベンチマーク最適化手法とその評価，情報処理学会論文誌コンピューティングシステム（ACS），Vol. 45, No. SIG 11(ACS 7), pp. 62–70 (2004).
[6] 成瀬 彰，中島耕太，住元真司，久門耕一: マルチコア PC クラスタ向け All-to-all 通信アルゴリズムの提案と評価，情報処理学会論文誌コンピューティングシステム（ACS），Vol. 3, No. 3, pp. 166–177 (2010).
[7] 遠藤敏夫，松岡 聡，橋爪信明，長坂真路，後藤和茂: ヘテロ型スーパーコンピュータ TSUBAME の Linpack による性能評価，情報処理学会研究報告ハイパフォーマンスコンピューティング（HPC），Vol. 2006, No. 87, pp. 43–48 (2006).
[8] NVIDIA: CUDA ZONE, https://developer.nvidia.com/cuda-zone.
[9] NVIDIA: cuBLAS, https://developer.nvidia.com/cublas.
[10] 遠藤敏夫，額田 彰，松岡 聡: スーパコンピュータ TSUBAME 2.0 における Linpack 性能 1 ペタフロップス超の達成，情報処理学会論文誌コンピューティングシステム（ACS），Vol. 4, No. 4, pp. 169–179 (2011).
[11] OpenMPI, https://www.open-mpi.org/.