Vol.2020-HPC-177 No.22
2020/12/22

低レイテンシ uTofu インターフェースを用いた格子QCD計算における通信の高速化

金森 逸作[1,a)]　中村 宜文[1,b)]　似鳥 啓吾[1,c)]　辻 美和子[1,d)]　向井 優太[2,e)]　三吉 郁夫[2,f)]
松古 栄夫[3,g)]　石川 健一[4,h)]

概要：格子 QCD は，隣接通信を多用する典型的な HPC 計算であり，線形ソルバー内での縮約計算の頻度も高い．そのため，スーパーコンピュータ「富岳」開発において，ハードウェア・システムソフトウェア・アプリケーションソフトウェアが共同して開発にあたるコデザインの対象の一つになっている．本講演では，コデザインの成果を踏まえて実現した，富岳向けの格子 QCD 用疎行列線形ソルバーにおける通信の高速化について報告する．隣接通信には低レイテンシの uTofu インターフェースを用いており，MPI 持続通信を用いるよりも小さな通信オーバーヘッド，きめ細かな通信リソースの割り付けを実現している．また内積計算に必要な少数要素の縮約についても，Tofu バリアと呼ばる機能で高速化を実現している．

Acceleration of communication with low latency uTofu interface in LQCD application

Abstract: Lattice QCD is a typical HPC application which uses frequent neighboring communications as well as reductions in linear solvers. For this reason, in the development of supercomputer Fugaku, it is one of the target applications for codesign, where hardware developers, system software developers, and application software developers work in close collaboration with each other. In this talk, we report some outcomes from the codesign activities: acceleration of communication we achieved in iterative linear solver for the lattice QCD application. By using the low latency interface uTofu , we realized a smaller overhead for neighboring communications than that with MPI persistent communications, and a refined assignment of the communication resources. We also realized acceleration of reduction with small numbers of elements for inner products by using the Tofu barrier feature.

1. はじめに

スーパーコンピュータ「富岳」[1] は，全 158,976 ノードからなるシステムで理化学研究所計算科学研究センターに設置されている．その開発では，計算機の開発者だけではなくハードウェア・システムソフトウェア・アプリケーションソフトウェアの開発者が共同して開発にあたるコデザインの手法が取り入れられている [2]．富岳の利用が想定される数あるアプリケーションの中から典型的なものがターゲットアプリケーションとして選ばれ，それらの実効性能が上がるように開発が進められてきた．

ターゲットアプリケーションの一つが，格子 QCD（Lattice QCD, LQCD）である．LQCD は，物質の構成要素であるクォークの相互作用を記述する量子色力学（Quantum ChromoDynamics, QCD）を計算機で扱いやすい形式で記述したものであり，その発展は大型計算機の発展と軌を一にしてきた．計算時間の大部分を，反復法を用いた疎行列（構造格子上のステンシル計算）の線形ソルバーに費やす．

[1] 理化学研究所計算科学研究センター
RIKEN R-CCS
[2] 富士通株式会社
Fujitsu Limited
[3] 高エネルギー加速器研究機構
High Energy Accelerator Research Organization (KEK)
[4] 広島大学先進理工系科学研究科
Graduate School of Advanced Science and Engineering, Hiroshima University
[a)] kanamori-i@riken.jp
[b)] nakamura@riken.jp
[c)] keigo@riken.jp
[d)] miwako.tsuji@riken.jp
[e)] mukai.yuta@fujitsu.com
[f)] miyoshi.ikuo@fujitsu.com
[g)] hideo.matsufuru@kek.jp
[h)] ishikawa@theo.phys.sci.hiroshima-u.ac.jp

ⓒ 2020 Information Processing Society of Japan

並列計算の観点からは，隣接通信・大域的縮約演算の頻度が高いアプリケーションの典型例になっている．LQCD アプリケーションとのコデザインの成果として富岳に取り入れられたものには，低レイテンシの通信インターフェースである uTofu（以下，uTofu API）の提供，Tofu バリア機能を用いた縮約演算を「京」の 1 要素から 3 要素までに拡張などがある[*1]．

本発表では，コデザインの成果を踏まえて京コンピュータから増強された通信機能が，LQCD 向けのソルバーライブラリである qws (QCD Wide Simd library) の性能向上に果たした役割を報告する．qws は，C 言語および C++ 言語で書かれており，github にて公開されている [3]．名前から分かるとおり qws は SIMD 幅の広いアーキテクチャ用にデータ構造が設計されており，演算部分の性能は十分に高い．一方で，京コンピュータから増強されたとはいえ富岳の通信部分の性能向上はノードあたりのインジェクションバンド幅で 2 倍にとどまり（演算は単精度ノーマルモードで 48 倍），演算で通信部分を隠蔽しきれなくなっている．そのため通信部分の高速化なしには，富岳で性能を出すのが難しい．本報告では qws のベンチマークを通じて qws に実装した以下の富岳向けの改善要素の効果をそれらの要素のオンオフにより検証する．

- 隣接通信アルゴリズムの工夫
- プロセスランクマップと通信方向の割り付けの工夫
- 通信命令発行の並列化
- キャッシュインジェクション機能の利用
- Tofu バリア通信機能の利用

これらの機能の詳細については本文中で説明する．

関連した仕事では，コデザインの中でも富岳でのマイクロプロセッサ (A64FX[4]) の開発に関わるものに [2] がある．LQCD 向けに特化した計算機の開発は，QCDPAX [5], CP-PACS [6], QPACE [7] が知られている．特定のアーキテクチャ向けの LQCD アプリケーションの高速化については，たとえば [8], [9], [10] がある．また，LQCD 分野の国際会議でも最新の動向が毎年報告されている（[11] など）．

2. Tofu インターコネクト D と uTofu インターフェース

富岳のノード間接続には Tofu インターコネクト D（以下，TofuD）が用いられている．TofuD は京コンピュータに用いられた Tofu インターコネクトを強化したもので，6 次元メッシュ/トーラスと呼ばれるトポロジを持つ．図 1 に 6 次元メッシュ/トーラスの概念図を示す．X，Y，Z の 3 軸の長さはシステムの構成によって可変，A，B，C の 3 軸の長さはそれぞれ 2，3，2 で固定である．**表 1** に TofuD の諸元 [12] を記載した．富岳の有する約 15 万ノードの間

[*1] 他にも OS のジッター・ゼロを目指す McKernel があるが，本発表では触れない．

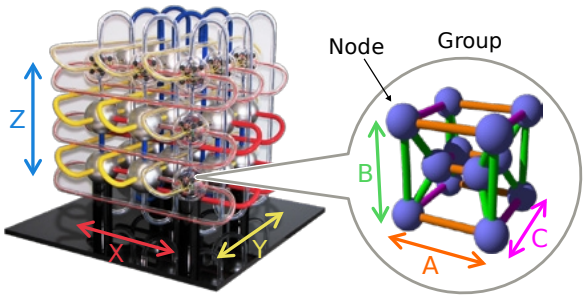

図 1 6 次元メッシュ/トーラスの概念図.
Fig. 1 Diagram of 6-dimensional mesh/torus.

で低レイテンシ・広バンド幅の通信を実現するために，隣接する全 10 ノードに 2 レーンのリンクで接続されている．また各ノードは TNI と呼ばれる Remote Direct Memory Access (RDMA) エンジンを 6 個備え，最大 6 方向へ同時に送信可能である．さらに Tofu バリアと呼ばれる高速な同期/縮約演算機構を備える．Tofu バリアの縮約演算機能は，LQCD アプリケーションとのコデザインの結果，浮動小数点数の 3 つの独立した総和演算を 1 度の通信で CPU を介さずに行えるようになっている．

この TofuD の性能を最大限に引き出す API が uTofu である．uTofu は TofuD を操作する低レベルな C 言語の API であり，富岳およびその商用機である富士通 PRIMEHPC FX1000 でもサポートされている．uTofu を使用することで，MPI より低レイテンシな通信，TNI の明示的な指定，TofuD の機能の明示的な利用ができるといった利点がある．また，uTofu は MPI と併用することも可能である．

TNI はそれぞれ複数の CQ (control queue) というインターフェースを持ち，CQ が異なれば複数のスレッドから同時に操作を受け付けることができる．uTofu API では指定した TNI/CQ に通信発行などの指示を出すことができるため，複数の通信を負荷が均等になるよう 6 個の TNI に分散したり，TNI の操作をスレッド並列化したりすることができる．

本報告では，TofuD のキャッシュインジェクションという機能を uTofu API から明示的に利用している．キャッシュインジェクションは，いくつかの条件を満たしたとき，受信データを主記憶だけではなく最終レベルキャッシュにも書き込む機能である．最終レベルキャッシュにも受信データを書き込むことで，後続する CPU による受信データへのアクセスのキャッシュミスを防ぐことができる．

Tofu バリアも uTofu API で操作することができるが，富士通 MPI でも条件を満たす通信は自動で Tofu バリアを用いるため，本報告の同期，縮約演算は MPI を使用している．

ⓒ 2020 Information Processing Society of Japan

表 1　TofuD の諸元 [12].
Table 1　Sepcifccation of TofuD[12].

ノードあたりのレーン数 x リンクバンド幅	6.8 GB/s
ノードあたりの TNI 数	6
ノードあたりのインジェクションバンド幅	40.8 GB/s

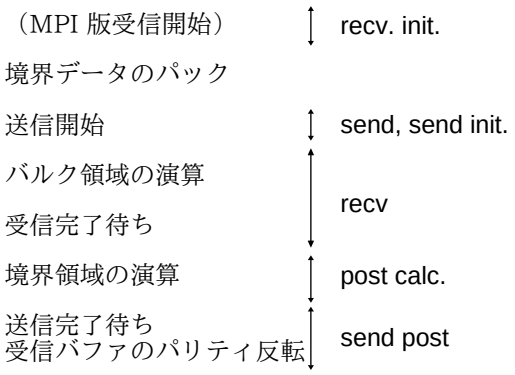

図 2　乗算の手順．受信バッファを 2 重化したダブルバッファリングを用いるので，バッファを切り替える操作も必要になる．右側の名称は次節での図 4，図 5 の凡例に対応する．
Fig. 2　Steps for the matrix multiplication.

3. 実装

3.1 アプリケーションの概略

qws では LQCD の大規模疎行列係数連立方程式のソルバーを実装している．その疎行列の構造は 4 次元構造格子上のステンシル計算となっている．各格子点の自由度は複素数が 12 成分のベクトルで，8 個の隣接格子点から，これらの 12 成分に作用する行列をかけた上で足しこむ．これを，領域分割を用いた前処理（乗法シュヴァルツ法）と組み合わせた単精度の行列 A に対して線形方程式 $Ax = b$ を双共役勾配安定化法（BiCGStab 法）で解く．行列 A の演算量は格子点あたりに換算すると 39406 FLOP で，B/F 比は 1.37 である．領域分割前処理を組み合わせることで，隣接プロセスへの通信頻度を削減している．1 プロセスあたり隣接した 2 領域を配置しているので，行列 A の作用でプロセス間を跨ぐ隣接通信は，プロセスあたり同時に（8 ではなく）7 個となる．BiCGStab 1 反復あたりの A の作用は 2 回で，A の作用 1 回あたりの隣接通信は 10 回である．隣接通信に着目した A の作用の手順を**図 2** に示す．

3.2 隣接通信のアルゴリズムとライブラリ

受信バッファが一つの単純な隣接通信では，受信したデータが利用前に上書きされることを避ける必要があり，受信バッファが新しいデータで上書き可能かを確認するための追加の通信が生じる．コデザインの過程でこの確認コストが無視できないとわかったので，本実装では，受信バッファを二重化してこの確認を省くダブルバッファリングアルゴリズムを用した．

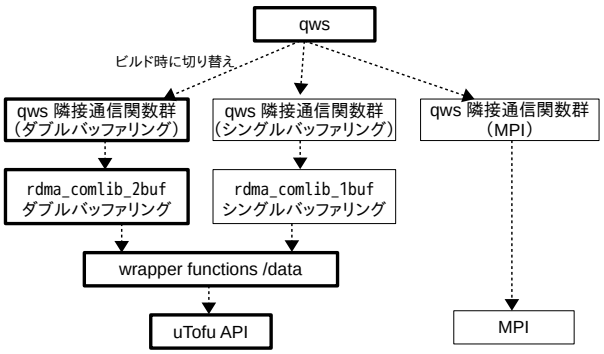

図 3　低レベルの API を隠蔽する通信ルーチン群.
Fig. 3　Routines to hide low level APIs.

通信部分は，**図 3** にあるように uTofu API をラップする隣接通信用のルーチン群を作ることで，qws から直接 uTofu API を呼ぶことを避けている．ラッパーでは，通信に必要な情報— CQ へのハンドラ，送信・受信バッファおよびそのサイズ，RDMA で送るデータの書き込み先のアドレス，受信完了確認用のタグの値など — を一つの構造体で管理している．その構造体へのポインタを引数にとる C 言語関数群がラッパー関数になる．さらにこのラッパー関数群を用いて，ダブルバッファリング（またはシングルバッファリング）の隣接通信を行うクラス（図中 `rdma_comlib_2buf` または `rdma_comlib_1buf`）を実装している．qws ビルド時に隣接通信ルーチン群を切り替えることで，uTofu API を用いない通常の MPI 通信版へ容易に変更できる．なおこれらのルーチン群は，ラッパー関数の階層，隣接通信クラスの階層，どちらでも qws 本体から切り離して単独で利用可能である．

uTofu 利用時のデータ受信完了の確認には，送受信バッファの末尾に付加したタグを用いる．uTofu のストロング・オーダリングの機能を用いることで，末尾のデータの到着で全てのデータの到着が保証されている．そのため送信側で送信毎にタグの値を更新し，受信側ではタグ値の変化の検出をもって受信完了とする．送信完了の確認は，TNI 上の送信完了キューをポールすることで実現している．これらはラッパー関数群内で実装している．

3.3 ランクマップと TNI の割り付け

大規模並列環境下の LQCD ではネットワークのトポロジが 4 次元トーラスであることが望ましい．TofuD インターコネクトの 6 次元トーラス・メッシュネットワークは，2 軸ずつ組み合せて容易に 3 次元トーラスは作れるが，そのままでは 4 次元トーラスにはならない．qws では，まず Tofu の Z 軸を 3 ノード毎のスライスに分けて 2 次元メッシュを作る（連続した方向を Z_c, ストライド 3 の方向を Z_d とする）．次節で詳細に調べるノード形状は $(AX, BY, CZ) = (4, 6, 18)$ で，B 軸は，環状に閉じている．

ここから (A,Z_c), (C,Z_d), B, (X,Y) の組み合わせでそれぞれ閉じた軸を 4 つ作り，$6 \times 6 \times 3 \times 4$ の 4 次元トーラスとした[*2]．

これらのもとで問題サイズを固定すれば，反復あたりの隣接ノードへの通信量が決まる．特定の TNI に通信が集中しないように，各ノード毎に，通信方向に応じて利用する TNI を指定している．プロセス内の格子サイズを $32 \times 6 \times 4 \times 3$ としたときの全 6 TNI 用いたときの TNI 割り付けと，その際の通信量を**表 2** 及び**表 3** にまとめた．ノード内通信も TNI を経由するため同一ノードへのループバックも計上している．各 TNI 毎の通信量については，4 TNI に制限した場合（次節の utofu base が該当）の値も記載した．

3.4 通信発行のスレッド並列化

uTofu API を用いた通信の発行は，スレッド並列に行うことができる[*3]．これにより，4 次元トーラス上での隣接通信発行の時間を短縮できる．実装では，4 次元軸に関する `omp for` ループ内で `utofu_put` 関数をラッパー経由で呼び出し，隣接通信を発行している．

3.5 キャッシュインジェクション

uTofu API を用いた RDMA put 通信では，主記憶だけではなく最終レベルキャッシュにも直接データを書き込むことができる．そのためには put 発行時のフラグに加えて受信バッファがキャッシュに収まる大きさであり，かつ，そのアラインメントが 256 バイトに取れていること，バッファには（put 通信を除く）書き込みがないこと，が必要である．また受信時にバッファがキャッシュヒットしている必要もある．

本測定条件ではノードあたりの問題サイズが小さいため，作業領域を全て最終レベルキャッシュに収めることができる．そのためキャッシュインジェクションを用いることで，データ受信時に受信バッファもキャッシュに残すことができる．そうすることで，受信データを用いる計算部分の処理時間を短縮できる．

3.6 Tofu バリアを用いた高速な通信

qws の一反復あたりでは，1 要素の Allreduce が 1 回，2 要素が 1 回，3 要素が 2 回ある．これらの Allreduce に対しては，Tofu バリア機能を用いた高速な通信アルゴリズムが用いられている．

表 2 各 LQCD 軸方向の，通信メッセージ長と利用 TNI．LB は同一ノード内へのループバックで，右端の通信量はノード内プロセス数や通信の多重度（TZ_d 方向への 3 ホップ）を加味している．プロセス内の格子サイズは，$32 \times 6 \times 4 \times 3$ としている．
Table 2 Message size and TNI for each QCD axis.

QCD プロセス座標	Tofu 軸	TNI	メッセージ長さプロセス (Byte)	リンク・ノード内への通信量 (Byte)
QX	TA	0/1	3456	13824
	TZ+	0		13824
	TZ−	1		13824
QY+	TC	0/1	9216	18432
	TZ+	0		55296
	TZ−	1		55296
	LB	2		19432
QY−	TC	0/1	9216	18432
	TZ+	0		55296
	TZ−	1		55296
	LB	3		19432
QZ+	TB	2	13824	27648
	LB	2		27648
QZ−	TB	3	13824	27648
	LB	3		27648
QT+	TX	4	18432	73728
	TY	4		73728
QT−	TX	5	18432	73728
	TY	5		73728

表 3 隣接通信 1 回あたりの，各 TNI への通信量．
Table 3 Data size to each TNI for 1 set of neighboring communications.

TNI	通信量（Byte）	4 TNI に制限時の通信量 (Byte)
0	69120	115200
1	69120	115200
2	73728	101376
3	73728	101376
4	73728	—
5	73728	—

4. 測定結果

4.1 測定条件

前節で述べた機能の効果を調べるために，ベースライン (**utofu base**) として

- 隣接通信で用いる TNI 数を 6 個ではなく 4 個に制限
- 隣接通信発行のスレッド並列化なし
- 受信バッファのキャッシュインジェクションなし

としたものを用意した．そしてこれらの各機能の有無を組み合わせた以下のものを用意した．期待される効果も合わ

[*2] ノード内には $1 \times 2 \times 2 \times 1$ の 4 プロセス割り当てで，全体では $6 \times 12 \times 6 \times 4$ の 1728 MPI プロセスになる．
[*3] CQ ハンドラ取得時にオプション指定が必要．

ⓒ 2020 Information Processing Society of Japan

表 4 測定結果．BiCGStab 500 反復（隣接通信 10,000 回）の経過時間で単位は秒．all は全区間，all reduce は `MPI_Allreduce` に要した時間．その他の区間については図 2 を参照．

Table 4 Measured elapsed time in second for 500 iterations of BiCGStab (10,000 times of neighboring communications).

	all	post calc.	recv.	send	send post	recv. init.	all reduces	all reduces*
utofu base	0.4567	0.0492	0.1731	0.0314	0.0277	—	0.0161	0.0363
w/ 6 TNI	0.4344	0.0491	0.1490	0.0303	0.0331	—	0.0164	0.0369
w/ threaded put	0.4535	0.0494	0.1766	0.0178	0.0284	—	0.0164	0.0348
w/ cache inject.	0.4436	0.0394	0.1688	0.0314	0.0278	—	0.0161	0.0362
full	0.4129	0.0396	0.1446	0.0182	0.0331	—	0.0167	0.0339
full [1buf]	0.4378	0.0398	0.1353	0.0131	0.0506	0.0118**	0.0191	0.0368
FJMPI [1buf]	0.5066	0.0492	0.1544	0.1045	0.0101	0.0442	0.0162	0.0350
FJMPI [2buf]	0.5454	0.0491	0.2184	0.0836	0.0098	—	0.0262	0.0508
MPI [1buf]	0.6249	0.0406	0.3031	0.0967	0.0411	0.0205	0.0170	0.0693
MPI [2buf]	0.6259	0.0402	0.3166	0.0999	0.0398	—	0.0175	0.0668

*計測直前のバリア同期を省略
** uTofu シングルバッファリングに対しては send init.

せて記載する．

w/ 6 tni 全 6 TNI を用いたもの：受信待ち時間 (recv.) の短縮

w/ th 隣接通信の発行を，スレッド並列化したもの：送信に関する時間 (send, send post) の短縮

w/ c.inj キャッシュインジェクションを有効にしたもの：受信したデータを用いた計算時間 (post calc) の短縮

full これら全てを有効にした場合：上記 3 つ全ての時間短縮

さらに，ダブルバッファリングの効果を検証するため

full 1buf full をシングルバッファリング仕様にしたものも用意した．ダブルバッファリングに効果があれば，送信に関する時間が full 1buf より full の方が短いと期待される．

uTofu API を利用した効果の比較のために，MPI 持続通信を用いた以下の実装も用意した．

fjmpi[1b] uTofu API を用いずに，`FJMPI_Prequest_`で始まる富士通拡張インターフェースを用いたもの（シングルバッファリング版）

fjmpi[2b] 上記をダブルバッファリングにしたもの

mpi[1b] 通常の MPI を用いた実装，シングルバッファリング版

mpi[2b] 上記のダブルバッファリング版

富士通拡張インターフェース版は，通常の MPI 版で用いた `MPI_Send_init`, `MPI_Recv_init`, `MPI_Start` をそれぞれ `FJMPI_Prequest_init`,... と置き換えたものである．

測定には富岳共用前評価環境を用いた．測定時の言語環境は `tcsds-1.2.27b` で，コンパイル時のオプションは

```
-Kfast,restp=all,optmsg=2,ocl,preex,noprefetch,\
noswp  -Nline,lst=t -Nnofjprof -Nfjomplib \
-Kilfunc=loop -Krdconv=2
```

である．C++言語で書かれた部分については `-std=gnu++11` 及び一部のファイルは更に `-Knosch_pr` を加えている．

MPI は，ノードあたり 4 プロセス（1 CMG あたり 1 プロセス）割り当てている．また実行バイナリは，第 2 階層ファイルシステムではなく，16 ノード毎に実装されているシステムディスク (SSD) 上の `/tmp` にコピーして実行した．実行時間については，個別の測定区間に対するタイマーのみを有効にしたバイナリを 1 回だけ実行したときに，もっとも時間がかかったスレッドの実行時間を採用している．1 回の実行は BiCGStab 500 反復で，隣接通信を 10,000 回含んでいる．なお Allreduce の実行時間測定の直前には `MPI_Barrier` を挿入し，ノード間の同期をとってから計測している．

4.2 測定結果

qws 500 反復の実行時間を**表 4** に記載した．また実行時間全体の比較を図 4 にまとめる．全ての機能を盛り込んだ full がもっとも実行時間が短いことが分かる．

各機能の効果を見るために，受信待ち，送信発行，受信データを利用した演算部の内訳を図示したのが図 5 である．以下では順に時間短縮効果をみていく．

利用する TNI を 4 から 6 に増やすことで，レイテンシが無視できる理想的な状況ではデータの受信待ちに要する時間が 2/3 に短縮される．しかし，実際の短縮幅は 14% であった（図 5 上段）．表 3 にあるように，各 TNI の通信量は，最大 115,200 バイトから 73,728 へ 36% 削減されたのに比べて小さな短縮幅にとどまっている．これはメッセージ長が短いため，レイテンシが無視できないからだと理解できる．

スレッド並列化の利用によって送信発行に要する時間

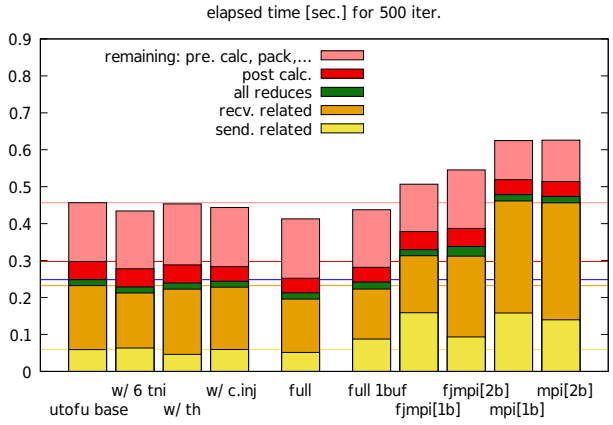

図 4 BiCGStab ソルバー 500 反復に要した実行時間の内訳．隣接通信を 10,000 回含んでいる．ラベルの詳細は本文参照．左端（utofu base）がベースラインで，その右 3 つがキャッシュインジェクション，通信発行のスレッド並列化，TNI を 6 個利用，各機能を個別に有効にしたもの，中央の full が全ての機能を有効にしたもの．

Fig. 4 Details of elapsed time for 500 iteration of BiCGStab solver, which contains 10,000 sets of neighboring communications.

（send）が 40%以上削減された（図 5 中段）．一方で隣接通信に用いる TNI を 4 個から 6 個に増やすことで，送信完了確認 (send post) の時間が若干伸びている．シングルバッファリングだと，受信バッファが利用可能になったという通知の送信（send post 内），通知の到着待ち（send init）が加わるためダブルバッファリングより遅くなる．通知の送信には，utofu_put_piggyback8 関数（送れるデータサイズは小さいが，主記憶を経由せずに TNI にデータを送るため utofu_put より低遅延）を，スレッド並列化せずに用いた．全般的に uTofu API を用いた方が MPI 版（富士通拡張版を含む）のものよりも所要時間が短く，通信発行の低レイテンシ化に成功している．なお MPI シングルバッファリング版では，受信に関する MPI_Start （または FJMPI_Prequest_start）が受信バッファが利用可能であることを保証している．

キャッシュインジェクションについても，明確に効果を見てとれる（図 5 下段）．受信したデータを用いた計算の実行時間が，20%（1 回あたり約 1 μs）短縮されている[*4]．この短縮幅は，ハードウェアプリフェッチの効果が出るまでのキャッシュミスが無くなった効果と考えて矛盾はなく，キャッシュインジェクションの効果といえる．

ノード間の足並みの乱れは，Allreduce 実行時に顕在化する．足並みの乱れと縮約のコストを切り分けるために，Allreduce の計測時の直前に MPI_Barrier で同期を取っているが，この同期を外すと MPI を用いた実装のほうが時間がかかる傾向があることがわかる（表 4 最右列）．理由については，MPI 関数の実装，測定時の OS ジッターなど様々な要素が絡むので，ここでは立ち入らない．

[*4] MPI 版においても同程度の高速化があるように見えるが，MPI 関数の実装の詳細に依存するのでここでは立ち入らない．

表 5 Allreduce において Tofu のバリア通信機能を “用いる” 場合と “用いない” 場合の，BiCGStab 1 反復あたりの実行時間の比較．

Table 5 Elapsed time of BiCGStab per iteration, comparision between Allreduce w/ and w/o the Tofu barrier feature.

ノード数	ノード形状	実行時間 [ミリ秒] 用いない	用いる	実行時間 比率
432	4x6x18	1.111	0.821	1.35
576	4x6x24	1.130	0.823	1.37
720	4x6x30	1.146	0.828	1.38
864	4x6x36	1.153	0.830	1.39
1008	4x6x42	1.160	0.835	1.39
1152	4x6x48	1.175	0.837	1.40
2304	4x12x48	1.218	0.847	1.44
9216	16x12x48	1.298	0.857	1.51
11520	20x12x48	1.380	0.861	1.60
13824	24x12x48	1.371	0.862	1.59
27648	24x24x48	1.448	0.874	1.66
27648	48x12x48	1.454	0.878	1.66

表 5 は，Allreduce において Tofu のバリア通信機能を用いる場合と用いない場合に，qws の BiCGStab 1 反復に要した時間を比較したものである．前述の full の設定で，ランクマップも同様のものを用いている（最下段のみ，B 軸と X 軸が入れ替わっている）．432 ノードから 27648 ノードにおいて Tofu のバリア通信機能の有無により実行時間に約 1.4 倍から 1.7 倍の差があり，Tofu のバリア通信機能を用いたほうが有意に速い．全系利用時には約 2 倍程度の性能差が出ると思われる．なお 1 要素の Tofu バリア通信機能は京の Tofu インターコネクトから存在したので，コデザインの正味成果は全ての Allreduce を 1 要素 Tofu バリア通信に置き換えることで見積もれる．このとき実行時間（机上値）は 432 ノードで約 5%，27648 ノードで約 10%の増加になる．

5. まとめ

スーパーコンピュータ「富岳」の開発では，コデザインを通じてアプリケーション側の要望が多く実現されている．その中から，LQCD 向けの疎行列線形ソルバーライブラリ qws における通信の高速化を報告した．

LQCD アプリケーションでは，頻繁な隣接通信と Allreduce が必要で，通信の低レンテシ化が高速化には必須である．隣接通信の低レイテンシ化は，uTofu API とダブルバッファリングの組み合わせで実現できた．TofuD インターコネクトの通信バンド幅を無駄なく使うには適切な

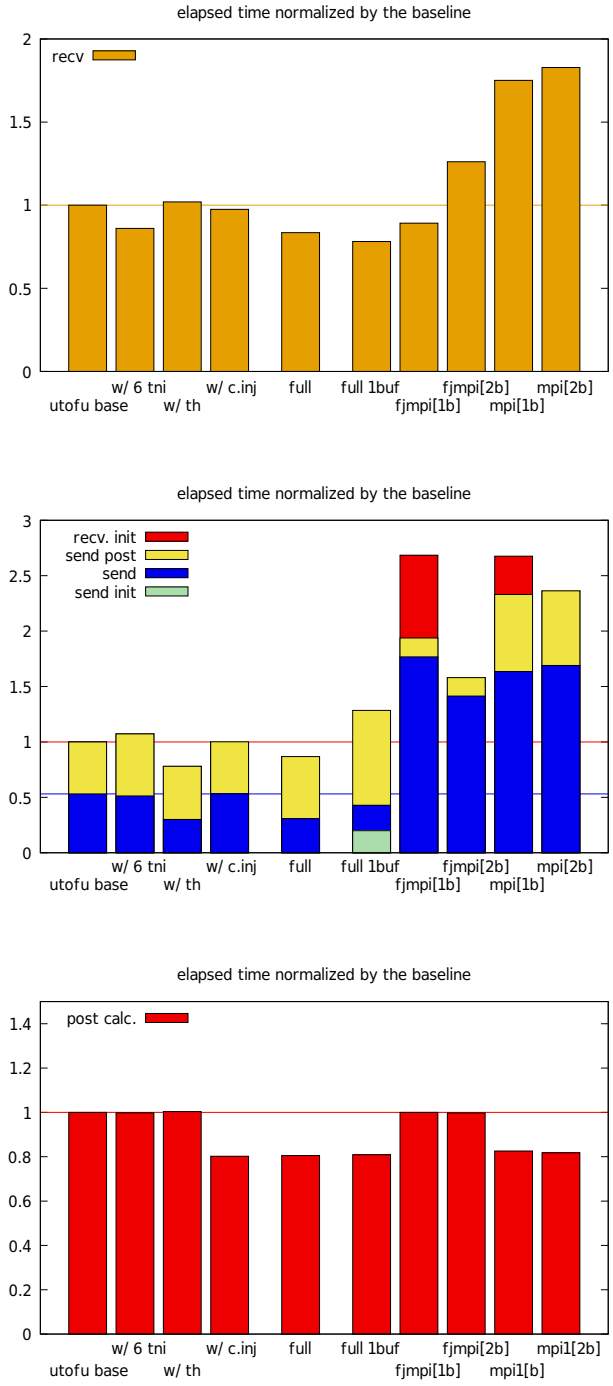

図 5 ベースラインで規格化した項目ごとの経過時間．短いほどよい．上から順に，受信待ちに要する時間（6 TNI を用いる効果），送信発行に要する時間（送信のスレッド並列化とダブルバッファリング利用の効果），受け取ったデータを用いる演算時間（キャッシュインジェクションの効果）．左端がベースラインで，右側 4 つは uTofu API を用いない実装．

Fig. 5 Relative elapsed time of each feature normalized with the baseline, lower is better. From top to bottom: timing for receiving (effect of 6 TNIs), for issuing send (threaded put and double buffering), and for computation with received data (cache injection).

TNI を用いる必要があるが，これも uTofu API の利用で実現できた．Allreduce における Tofu バリア通信機能も効果が大きい．Allreduce の実行時間が占める割合は Tofu バリア通信機能を用いれば 3.5% （432 ノード実行時）に過ぎない．これを 3 要素まではなく旧来の 1 要素までの Tofu バリア通信と比較した場合，机上値で全実行時間を 5% ほど短縮したことになり，全系だと短縮幅（予測値）は 10%を超える．これらはいずれもコデザインの成果といえよう．

なお富岳共用前評価環境における評価結果は，スーパーコンピュータ「富岳」の共用開始時の性能・電力等の結果を保証するものではない．

謝辞 本研究は，文部科学省『ポスト「京」で重点的に取り組むべき社会的・科学的課題に関するアプリケーション開発・研究開発」重点課題「宇宙の基本法則と進化の解明」』および，文部科学省『「富岳」成果創出加速プログラム「シミュレーションで探る基礎科学：素粒子の基本法則から元素の生成まで」』の一環として実施されたものです．また，本研究の一部は，スーパーコンピュータ「京」／スーパーコンピュータ「富岳」の計算資源の提供を受け実施しました．フラッグシップ 2020 の関係者，とくにコデザインワーキンググループの LQCD の関係者に感謝します．また uTofu や Tofu バリアの機能について情報を提供いただいた富士通の開発者に感謝します．

参考文献

[1] 理化学研究所計算科学研究センター：スーパーコンピュータ「富岳」について，（オンライン），入手先 ⟨https://www.r-ccs.riken.jp/jp/fugaku⟩ （参照 2020-11-12）．
[2] M. Sato, Y. Ishikawa, H. T. Y. K. T. O. M. T. H. Y. M. A. N. S. I. M. K. H. A. F. A. A. K. M. and Shimizu, T.: Co-design for A64FX manycore processor and “Fugaku”, *SC '20: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Article No. 47*, pp. 1–15 (online), DOI: 10.5555/3433701.3433763 (2020).
[3] Y. Nakamru, Y. Mukai, K.-I. I. I. K.: qws, (online), available from ⟨https://github.com/RIKEN-LQCD/qws⟩ (accessed 2020-11-15).
[4] Fujitsu Limited: A64 FX Microachitecture Manual, (online), available from ⟨https://github.com/fujitsu/A64FX⟩ (accessed 2020-11-05).
[5] Y. Iwasaki, T. Hoshino, T. S. Y. O. T. K.: QCDPAX: A parallel computer for lattice QCD simulation, *Computer Physics Communications*, Vol. 49, No. 3, pp. 449–455 (online), DOI: 10.1016/0010-4655(88)90005-7 (1988).
[6] 岩崎洋一：CP-PACS プロジェクト報告書，「専用並列計算機による「場の物理」の研究」研究成果報告書，筑波大学計算物理学研究センター（オンライン），入手先 ⟨https://www2.ccs.tsukuba.ac.jp/cppacs/file/report97.pdf⟩ （参照 2020-11-19）．
[7] Baier, H. et al.: QPACE: A QCD parallel computer based on Cell processors, *PoS*, Vol. LAT2009 (online), DOI: 10.22323/1.091.0001.
[8] 杉崎由典，南一生，庄司文由，中村宜文，藏増嘉伸，横川

ⓒ 2020 Information Processing Society of Japan

三津夫，寺井優晃：スーパーコンピュータ「京」における格子 QCD の単体性能チューニング，情報処理学会論文誌コンピューティングシステム（ACS），Vol. 6, No. 3, pp. 43–57 (2013).

[9] T. Boku, K. Ishikawa, Y. K. and Meadows, L.: Mixed Precision Solver Scalable to 16000 MPI Processes for Lattice Quantum Chromodynamics Simulations on the Oakforest-PACS System, *2017 Fifth International Symposium on Computing and Networking (CANDAR)*, pp. 362–368 (online), DOI: 10.1109/CANDAR.2017.69 (2017).

[10] I. Kanamori, H. M.: Practical Implementation of Lattice QCD Simulation on SIMD Machines with Intel AVX-512, (online), DOI: 10.1007/978-3-319-95168-3_31 (2018).

[11] Clark, K.: GPUs For Lattice Field Theory,(online), available from ⟨https://indico.cern.ch/event/764552/contributions/3428328/attachments/1865778/3067959/lattice_2019.pdf⟩ (accessed 2020-11-15).

[12] Y. Ajima, T. Kawashima, T. O. N. S. K. H. T. S. S. H. Y. I. T. Y. K. U. and Inoue, T.: The Tofu Interconnect D, *Proc. of the IEEE*, Vol. Cluster 2018, pp. 646–654 (online), DOI: 10.1109/CLUSTER.2018.00090 (2018).

ⓒ 2020 Information Processing Society of Japan