Vol.2015-HPC-150 No.39
2015/8/6

ACP ライブラリの性能最適化に関する検討

野瀬貴史[†1,†2]　安島雄一郎[†1,†2]　佐賀一繁[†1,†2]　志田直之[†1,†2]　住元真司[†1,†2]

ACE （Advanced Communication for Exa）プロジェクトでは，エクサスケールに向けて省メモリ・低遅延を両立する通信ライブラリ ACP (Advanced Communication Primitives)の開発に取り組んでおり、我々は ACP ライブラリの基礎となる ACP 基本層の Tofu インターコネクト 2 向けの性能チューニングを進めている。各種条件を変えながらリモート側不可分操作 1 回のレイテンシを評価した結果、スレッドセーフ時の最良値は 4.26 マイクロ秒、スレッドアンセーフ時の最良値は 3.68 マイクロ秒となった。さらなる最適化として、外乱抑制と将来の通信スレッド廃止に向けて通信スレッドの順序制御スレッドと委譲処理スレッドの 2 つに分割することも検討したが、ハードウェアの仕様に課題がある。

Performance optimization of the ACP library

TAKAFUMI NOSE[†1,†2]　YUICHIRO AJIMA[†1,†2]　KAZUSHIGE SAGA[†1,†2]
NAOYUKI SHIDA[†1,†2]　SHINJI SUMIMOTO[†1,†2]

In the ACE (Advanced Communication for Exa) Project, we are developing the ACP (Advanced Communication Primitives) library that satisfies both small memory consumption and low latency for the Exascale era. We are focusing on the performance tuning of the ACP Basic layer for the Tofu interconnect 2. We evaluated the latency of the atomic operation changing what optimizations are applied and achieved 4.26 microseconds under thread-safe condition and 3.68 microseconds under thread-unsafe condition. For further optimization, we investigated an effect of dividing a communication thread into a command ordering thread and a delegation thread. However, we found some problems on the specification of the hardware.

1. はじめに

ACE （Advanced Communication for Exa）プロジェクト[1]では，エクサスケールに向けて省メモリ・低遅延を両立する通信ライブラリ ACP （Advanced Communication Primitives）[2]の開発に取り組んでいる．既に ACP の基本部分の開発は完了し、ACP 1.1 としてリリースされており、現在は､性能チューニング､分散データライブラリの拡充、アプリケーションからの利用方法の検討、新規 API の検討等を行う段階に移行している｡我々は ACP ライブラリの基礎となる ACP 基本層の Tofu インターコネクト 2[3]向けの性能チューニングを進めており、その検討と評価の状況を報告する。

2. ACP 基本層の概要とその特徴

ACP 基本層[4]は､ACP ライブラリ[5]の最も基礎となるライブラリであり、通信ハードウェアの機能の違いを吸収し抽象化する。ACP 基本層では、各プロセスが登録したメモリ領域に対応付けられる 64bit グローバルアドレスを介し、各プロセスが全プロセスのメモリを参照してデータコピーおよび不可分操作といったグローバルメモリ操作を行うことができるインターフェースを提供する。グローバルアドレスは 64bit となっているため、各種インターコネクトの不可分操作で一般的に実装されている 64bit 長の不可分操作をグローバルアドレス自体に適用させることができ、不可分操作を用いた分散データ構造の実現に適した設計となっている。分散データ構造の更新には CAS (Compare and Swap)命令が多用されるため、通信ライブラリの不可分操作レイテンシは重要である。

ACP 基本層の特徴的な機能として、リモート間コピー機能とグローバルメモリ操作の順序関係の指定機能がある。リモート間コピーは、acp_copy 関数で指定するデータのコピー元とコピー先の両方を、acp_copy 関数を呼び出したプロセス以外、つまりリモートプロセスに属するグローバルアドレスにできる機能である｡図 1 にリモート間コピーの概略を示す。リモートプロセス間のデータコピーは直接行われるため、同等の機能を RDMA の Get と Put の組み合わせで行う（図 2）よりも転送時間およびメモリの両方の効率が良い。ACP におけるグローバルメモリ操作はハンドルによって順序関係を指定することができる。つまり、特定の通信が完了するのを待ち、完了を契機として開始される通信を ACP ライブラリに対して指示することができる｡順序を指定するサンプルコードとその概略を図 3 と図 4 に示す。ランク 1 から 2, 3 へのデータ転送はリモート間通信であるので、Bcast のようなデータ転送を、他のプロセッサの介在なくランク 0 が行えていることがわかる。

†1 富士通株式会社　次世代テクニカルコンピューティング開発本部
Fujitsu Limited., Next Generation Technical Computing Unit
†2 独立行政法人科学技術振興機構　戦略的創造研究推進事業
Japan Science and Technology Agency (JST),
Core Research for Evolutional Science and Technology (CREST)

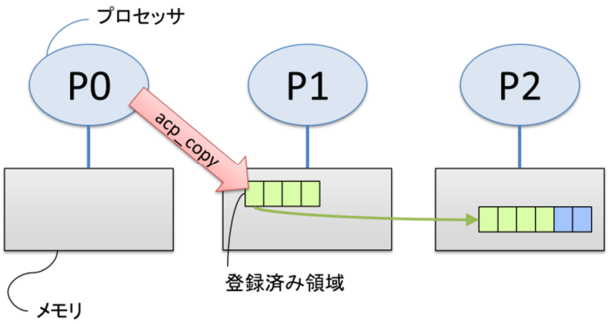

図 1 ランク 1 からランク 2 へのリモート間コピー

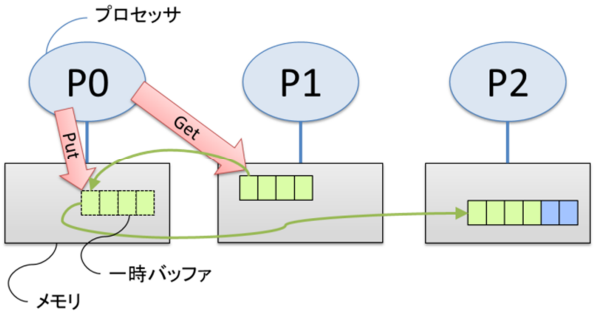

図 2 リモート間コピーを Put と Get の組み合わせで実現する例

```
/* ga_0, ga_1, ga_2, ga_3 はそれぞれランク 0～3 の持つ
   バッファのグローバルアドレス */
/* acp_copy の返り値は通信のハンドルであり、
   第 4 引数には待ち合わせる通信のハンドルを指定する。
   ACP_HANDLE_NULL の場合は待ち合わせず直ちに通信が
   開始される */
h0 = acp_copy(ga_1, ga_0, bufsize, ACP_HANDLE_NULL);
h1 = acp_copy(ga_2, ga_1, bufsize, h0);
h2 = acp_copy(ga_3, ga_1, bufsize, h0);

/* ACP_HANDLE_ALL を指定すると先行するすべての通信を
   待ち合わせる*/
acp_complete(ACP_HANDLE_ALL);
```

図 3 通信間に順序を指定するサンプルコード

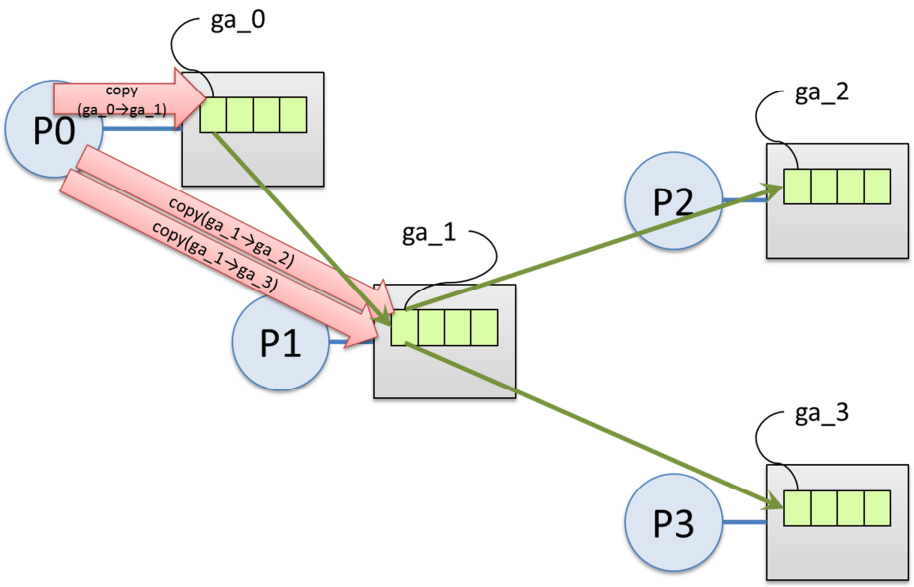

図 4 図 3 のコードで実現される通信の模式図

3. ACP 基本層実装の概要

3.1 全プラットフォーム共通の特徴

ACP 基本層には UDP、InfiniBand、Tofu インターコネクト 1/2 向け実装が存在するが、それぞれの詳細については[6], [7] および[8]を参照されたい。ここでは性能を検討する上で考慮すべき特徴について述べる。ACP 基本層の機能は将来のハードウェア化を考慮して設計されているが、現在実装が存在する UDP, InfiniBand, および Tofu インターコネクト 1/2 ではハードウェアに ACP の機能と対応するものが全て存在しているわけではないため、一部がソフトウェア実装となっている。特に、リモート間コピーは現在そのような機能をサポートするハードウェアが存在しないため、すべてのプラットフォームにおいてソフトウェア実装となっている。ソフトウェア実装となる機能のサポートと、グローバルメモリ操作の順序制御のため、現在の各プラットフォーム向けの実装は全てメインスレッドとは別に通信スレッドが存在している。

3.2 Tofu 1/2 版の概要

Tofu 1/2 版の実装の概要について述べる。メインスレッドと通信スレッドとの間にはコマンドキューが存在しており、メインスレッドで ACP の関数が呼び出されると対応するコマンドがキューに投入され、通信スレッドがそれを受け取ると通信が開始される。キューのエントリは主として FREE、QUEUED、ORDER、ORDER_END、EXECUTING、DELEGATED、FINISHED という状態を持ち、それぞれ表 1 で示した意味の状態であることを示す。メインスレッドと通信スレッドがコマンドキューエントリを介してどのようにやりとりするかの基本的な流れを図 5 に示す。NIC は Tofu 1/2 に搭載されたネットワークインターフェースを表している。Tofu1 では不可分操作がハードウェアに実装されていないため、完全にソフトウェアで実装されている。このため、ACP の不可分操作関数が呼び出された時にはコマンド委譲が発生し、不可分操作のコマンドは DELEGATED のルートを通る。一方、Tofu2 ではハードウェア機能を呼び出すため、DELEGATED ではなく EXECUTING のルートを通る点が Tofu1 版と異なる。リモート間コピーは両者ともに実装されていないため、共通して委譲によるソフトウェア実装のルートを通る。

4. 性能最適化の検討

ACP 基本層の Tofu2 版の性能最適化にあたっては、2 つの観点が考えられる。1 つはレイテンシの削減であり、これは ACP に限らずどのような通信ライブラリにおいても重要な観点である。もう 1 つは、ACP 固有の事情として、コマンド委譲に伴い発生する外乱の抑制という観点が挙げられる。なお、スループットについては、Tofu 1/2 版では十分な性能が確保できているため、検討しない。

まず、レイテンシの削減について検討する。Tofu 向け ACP ライブラリは最適化が十分進んだ段階ではないため、無駄なコードや必ずしも適切でない設計が含まれる。特にデバッグ用途のコードが不要であるのは明らかである。Tofu のハードウェアに対する命令であるディスクリプタの生成処理も、可読性を優先した状態となっており、最適化の余地がある。また、ACP 基本層の各関数はスレッドセーフとなっており、コマンドキューへの書き込み時にはロックがかかるが、状況によっては不要である。ACP ではプロセス数が増大してもメモリ消費が抑えられるため、大規模であっても Flat MPI に相当する実行モデルが MPI に比べ実行しやすい。よって、スレッド数よりもプロセス数で並列度を稼ぐケースが多くなると考えられ、このような状況下ではスレッドセーフ化を解除しレイテンシを削減するのは有効である。一方で、スレッドセーフが要求されるケースが消滅するわけではないため、スレッドセーフを維持しつつレイテンシを削減する方策も別途検討が必要である。コマンドキューに未完了命令が存在しない状況下では、メインスレッドから通信スレッドをバイパスして直接 Tofu インターコネクトの NIC に通信命令を発行することが可能であり、レイテンシの隠蔽が期待できる。具体的な方法を図 6 に示す。

表 1 コマンドキューエントリの状態

状態名	意味
FREE	エントリはキューに投入されていない。
QUEUED	エントリはキューに投入された直後である。
ORDER	投入されたコマンドが待ち合わせるハンドルに ACP_HANDLE_ALL が指定された場合、待ち合わせるハンドルに具体的なハンドルが再設定された状態である。具体的なハンドルが指定されていた場合、再設定はされない。また、ACP_HANDLE_NULL の場合はこの状態は省略され ORDER_END になる。
ORDER_END	待ち合わせる通信が終了し、自身の通信を開始できる状態である。
EXECUTING	Tofu 通信を実行中である。
DELEGATED	他ノードに対してコマンドを委譲中である。この場合 EXECUTING の状態にはならない。
FINISHED	コマンドの処理が全て終了した。

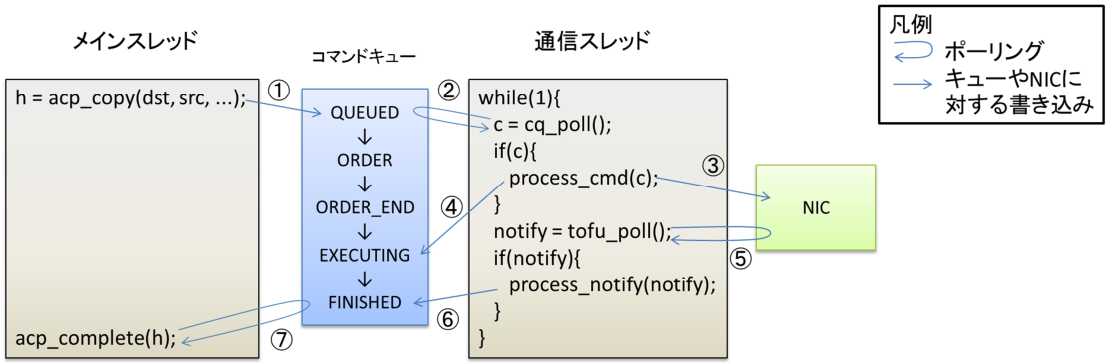

図 5 メインスレッドと通信スレッドによるコマンド処理の概略

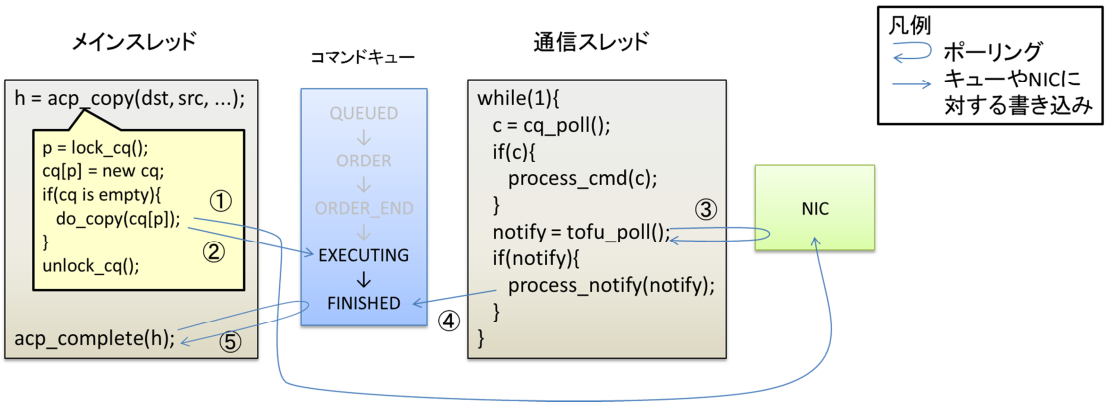

図 6 特定の状況下では通信スレッドをバイパスして NIC に指示できる

次に、外乱の抑制について検討する。ACP 基本層に通信スレッドが必要となるのは、独立した 2 つの理由によっている。1 つはローカルで投入されたコマンド同士の順序関係を解決するためであり、もうひとつは外部から委譲されたコマンドの待ち受けと応答のためである。2 つの処理を分離することで、委譲による外乱を抑制することができる。具体的な実装を以下で検討する。現状のコマンド委譲機構は図 7 に示すような構造になっている。委譲が必要なコマンドを実行する時は、通信相手のデリゲーションバッファにコマンドの内容を送信し、相手の通信スレッドはその書き込みの完了通知を検知して委譲された処理を行う。通信スレッドは NIC からの完了通知が自分の行ったデータ転送のものなのか、コマンド委譲に関わる制御通信のものなのかを判別し、適宜処理を振り分けている。ここで、ハードウェアの制御キューは複数存在するので、通常のデータ転送に用いる制御キューとは別に委譲コマンド用の制御キューを確保し、委譲コマンド処理用のスレッドを新たに専用に割り当てることで、外部からの要求に伴う通信スレッドの負荷を分離することができると考えられる。ただし、Tofu インターコネクト 2 ではこの実装は成立しない。なぜならば、ハードウェアの制御キューはそれぞれ独立しており、仮想アドレス空間を共有していないためである。デリゲーションバッファには委譲されたコマンドに関係するグローバルアドレスが書き込まれる。Tofu 版 ACP においては、このグローバルアドレスに仮想アドレスを構成するステアリングタグと呼ばれる識別子とオフセットの情報が格納されているが、この仮想アドレスは通信スレッドが用いる制御キューに関連付けられたもので、委譲待ち受けスレッドが用いる制御キューとは関連がないため、直接用いることができない。将来のハードウェアでは制御キュー同士でアドレス空間を共有する設定ができることが望ましい。

上記で検討したコマンド委譲機構の分離は、ACP の機能のハードウェア化に際しても有効である。通信スレッドからコマンド委譲の待ち受け機能を分離すると、通信スレッドの役割は通信の順序制御のみとなる。ハードウェアに順序制御の機能を追加することで、通信スレッドを廃止することができ、レイテンシ改善に大きく貢献すると予想される。当然、ACP の機能を全てハードウェア化することでも性能の改善は達成されるが、全通信に対して効果のある通信スレッドの排除に比べ、コマンド委譲機構のハードウェア化は効果のある範囲が狭く、優先度は比較的低い。コマンド委譲機構のみをソフトウェアに分離することで、ハードウェアが実装する機能を限定することができ、設計に選択肢が生まれる。ただしこの場合、ハードウェアには自分の発行した通信命令の完了を待ち合わせるのみならず、別途ソフトウェアから発行された応答通信を待ち合わせる機能が必要であり、このコストは別途さらなる検討が必要である。

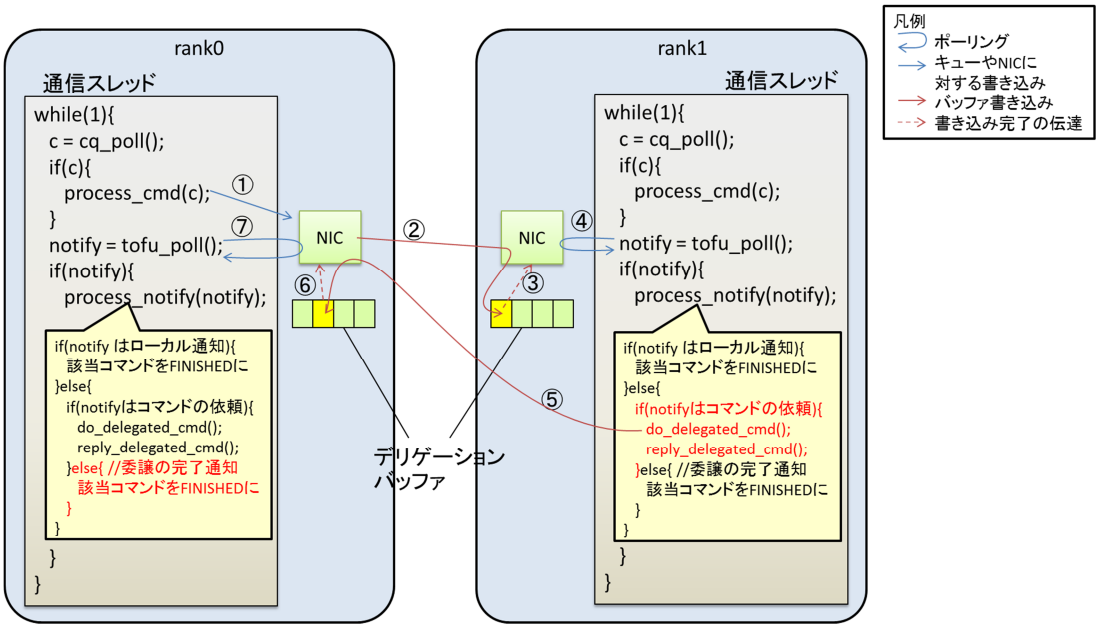

図 7 コマンド委譲機構の概略

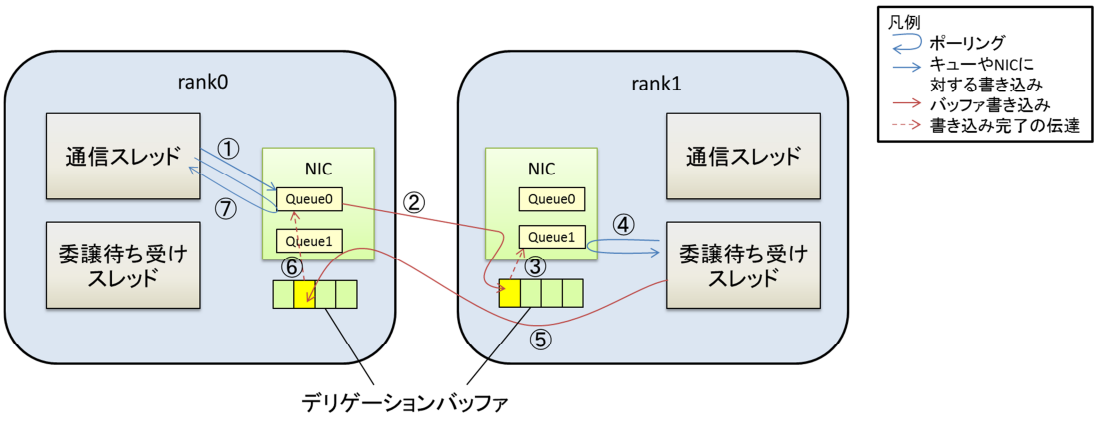

図 8 委譲待ち受けスレッドの分離案

5. 性能評価

4 節で検討した最適化のうち、無駄なコードの除去、ディスクリプタ生成処理の最適化、スレッドセーフの解除、通信スレッドのバイパスを Tofu2 版 ACP ライブラリに施し、適用する最適化を変えながら、4 バイトの不可分加算である acp_add4 と完了待ちの acp_complete の組を 1000 回繰り返すことで性能評価を行った。評価環境は表 2 の通りである。コンパイルオプションは-O3 である。[8]における報告時からマシン環境が更新されたため、ACP との比較用の、富士通 MPI の拡張 RDMA インターフェースに含まれる不可分操作とその完了待ち 1 回あたりの性能も再測定した結果、リモート操作は 2.26 マイクロ秒、ローカル操作 1.74 マイクロ秒に改善されていることが分かっている。測定条件とその結果を表 3 に示す。条件のハードウェア不可分操作は、[8]で実装した、Tofu2 の機能を利用する改造項目であり、残りが今回のものとなる。無駄なコードの除去は全項目に対して施している。前回と比べ、無駄なコードの除去以外何も最適化を行っていない条件 2 と 5 が大幅に改善されている。スレッドセーフの On/Off の観点から、条件 1 と 2、条件 3 と 5、条件 4 と 6 を分析すると、リモート操作とローカル操作ともに改善の傾向が見られる。ディスクリプタの最適化の観点で条件 3 と 4、条件 5 と 6、条件 7 と 8 を比較すると、こちらも改善の傾向が見られるが、概ね 0.1 マイクロ秒程度に留まる。

表 2 評価環境諸元

マシン名	PRIMEHPC FX100
CPU	SPARC64 XIfx
メモリ	32GB
ネットワーク	Tofu インターコネクト 2, 12.5GB/s
OS	Linux version 2.6.32
コンパイラ	富士通 C/C++ コンパイラ Version 2.0.0
MPI	富士通 MPI ライブラリ Version 2.0.0

表 3 測定結果

条件 No.	条件 ハードウェア不可分操作	ディスクリプタ最適化	スレッドセーフ	通信スレッドバイパス	結果[usec] Remote	Local
1					8.57	1.43
2			✓		8.61	2.17
3	✓				3.91	3.36
4	✓	✓			3.68	3.23
5	✓		✓		4.44	3.64
6	✓	✓	✓		4.31	3.51
7	✓		✓	✓	4.39	3.87
8	✓	✓	✓	✓	4.26	3.77

通信スレッドのバイパスは、条件 5 と 7、条件 6 と 8 を比較すると、リモート操作では 0.05 マイクロ秒の改善が見られるが、ローカル操作では効果が出ていない。この原因は調査中である。以上をまとめると、スレッドセーフ時には全ての最適化を適用した条件 8 のリモート操作が最も性能が良く、スレッドアンセーフ時には条件 4 が最も良い。Tofu2 はプロセッサと不可分操作が協調的に動作するため、ローカルの不可分操作についてはプロセッサで処理を行ってもよく、Tofu2 の機能を用いるよりもより性能の高い条件 1 または 2 の実装を用いれば良い。富士通 MPI の拡張 RDMA インターフェースはスレッドアンセーフなため、条件 4 と比較するのが適当であるが、依然としてリモート操作で 1.42 マイクロ秒の差があることが分かる。また、この差は、条件 1 のローカル操作の性能値と近い。プロセッサ単体でアトミック加算を行う時のコストは数十ナノ秒であるため、ACP の通信スレッドが存在する事による影響はリモート・ローカルに共通して 1.42 マイクロ秒程度と見積もれる。

6. まとめ

本稿では ACP ライブラリの基礎となる ACP 基本層の Tofu インターコネクト 2 向けの性能チューニングの検討を進め、そのいくつかの手法を適用した結果、リモート操作に関してはスレッドセーフ時の最良値は 4.26 マイクロ秒、スレッドアンセーフ時の最良値は 3.68 マイクロ秒となった。また、ローカル操作についてはそれぞれ 2.17 マイクロ秒と 1.43 マイクロ秒となった。本論文ではパフォーマンスカウンタ等の情報に基づく最適化には踏み込んでいないため、今後検討していく。

さらに、外乱抑止の観点から、コマンド委譲機構の通信スレッドからの分離方法を考察したが、Tofu2 の制御キューの仮想アドレス空間の仕様に課題があることが判明した。また、この分離方法は将来 ACP の機能のハードウェア化に実装上の選択肢を与えるものであるが、リモート間通信等の委譲コマンドをハードウェア実装しなくても良い代わりに、ソフトウェアから生成された応答を待ち合わせる機構が必要となる。

参考文献

1) ACE Project, http://ace-project.kyushu-u.ac.jp/main/jp/index.html
2) http://ace-project.kyushu-u.ac.jp/main/jp/05_downloads/index.html
3) Ajima, Yuichiro, et al. "Tofu Interconnect 2: System-on-Chip Integration of High-Performance Interconnect." Supercomputing. Springer International Publishing, 2014.
4) 安島 雄一郎, 佐賀 一繁, 野瀬 貴史, 三浦 健一, 住元 真司: ACP 基本層の設計思想とインターフェース, 情報処理学会研究会報告 Vol. 2014-HPC-143, No. 9, pp. 1-6, 2014.
5) 住元真司, 安島雄一郎, 佐賀一繁, 野瀬貴史, 三浦健一, 南里豪志: エクサスケール通信向け ACP スタックの設計思想, 情報処理学会研究会報告, Vol. 2014-HPC-143, No. 8, pp. 1-7, 2014.
6) 佐賀 一繁, 安島 雄一郎, 野瀬 貴史, 三浦 健一, 住元 真司: ACP 基本層の実装と初期評価, 情報処理学会研究会報告, Vol. 2014-HPC-143, No. 10, pp.1-6, 2014.
7) 森江 善之, 南里 豪志, 安島 雄一郎, 本田 宏明, 曽我 武史, 小林 泰三, 住元 真司: InfiniBand による ACP 基本層の実装と評価, 情報処理学会研究会報告, Vol. 2015-HPC-148, N0. 33, pp.1-6, 2015.
8) 野瀬 貴史, 安島 雄一郎, 佐賀 一繁, 志田 直之, 住元 真司: Tofu インターコネクト 2 上での ACP 基本層の実装と性能評価, 情報処理学会研究会報告, Vol. 2015-HPC-148, No. 32, pp.1-6, 2015.