インテル® MPI ライブラリー: Hadoop* エコシステムをサポート

MPI で計算負荷の高いタスクを Hadoop* の MapReduce よりも高速に処理

Mikhail Smorkalov インテル コーポレーション ソフトウェア開発エンジニア

長年にわたり **MPI** は、分散処理モデルの主流として利用されてきました。しかし、大量の入力データの処理を必要とするワークロードを扱うハイパフォーマンス・コンピューティング (HPC) により、新しいアプローチとフレームワークが登場しました。最も一般的なものは、Apache* Hadoop* MapReduce[1] であり、特に Hadoop* ソフトウェア・スタック (すべての関連ツールとフレームワークを含む) が挙げられます。

Vanilla Hadoop* は複数のモジュールで構成され、[2] 効率良く解くことができる問題の範囲には一定の制約があります。例えば、MapReduce は、反復間で中間結果をストレージにダンプする必要があるため、反復アルゴリズムには適していません。この欠点を改善するため、反復間で効率良くインメモリー・データをキャッシュできる Apache Spark* や Apache Storm* のような、ほかの Hadoop* フレームワークが開発されました。さらに、クラスター管理フレームワークの YARN* は、MapReduce 以外にも対応しているため、これまで MPI が主流であった分野でも Hadoop* を利用することができます。

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

タスクによっては、Hadoop* スタックを利用することでより簡単に効率良く処理することができ、計算負荷が高く、ノード間通信が多い機械学習でも Hadoop* プラットフォームの採用が増えつつあります。ただし、複数の研究において、[3、4] Hadoop* フレームワークよりも MPI のほうが高速であることが示されています。MPI は HPC 分野にしか適していないと誤解されている方がいらっしゃるかもしれません。この記事は、Hadoop* エコシステムで一般的に使用されているツールを補完するために MPI を利用する利点と課題を明らかにします。

新しい HPC の課題

歴史的に、高性能計算は気象モデル、物理、化学のような、比較的少量の入力を基に大量の計算を行う (場合によっては、大量のデータを出力する)、計算負荷の高い分野で利用されてきました。MPI はそのような問題に最適で、さまざまな通信パターンに加えて、多くの場合、ハードウェア機能を効率良く利用する MPI 実装も提供します。

しかし、新しい分野では、テラバイトやペタバイトのデータを入力し、大量のデータを扱うアプリケーションの開発が求められています。通常、ノード間通信をあまり必要としないため、効率良く、信頼性の高い、スケーラブルな入力/出力 (I/O) が特に重要になります。Hadoop* では、HDFS* によりこれに対応しています。

データ解析タスクには長い時間がかかるため、透過的なフォールト管理が必須です。さらに、ストリーミング・タスクは永続的に実行可能なため、ノードクラッシュは避けられません。Hadoop* ソフトウェア・スタックは、ジョブ全体を再起動しなくても、タスクを円滑に引き渡すことができるメカニズムを採用しています。失敗したサブタスクのみ正常なノードに渡されます。また、Hadoop* スケジューラーは、タスクをできるだけデータの近くで実行するため、最適なデータの局所性が得られます。

これらすべての機能が、データ・サイエンティストにとって Hadoop* プラットフォームを魅力的なものにし、Hadoop* エコシステムの解析フレームワークとライブラリーの開発を促進しました。フレームワークは、さまざまなプログラミング言語 (Python*、Java*、Scala* など) に対応しているため、簡単に導入することができます。従来の MPI 実装は、MPI 標準で定められている C と Fortran のみに対応していますが、Java* や Python* インターフェイスを提供する MPI 実装もあります。

MPI レパートリー

最初に、MapReduce プログラムの実装に利用できる、豊富な MPI 通信パターンについて考えてみましょう。Map フェーズは MPI_Scatter(v) 関数を利用して実装できます。Reduce フェーズには MPI_Reduce[5] か、リダクションを実行する前にすべてのデータが揃っていなければならない場合は MPI_Alltoall(v) とマージ操作を利用します。MPI 実装は通常、集合操作ごとにいくつかのアルゴリズムがあり、スケール、メッセージサイズ、ハードウェア・アーキテクチャーに応じて最適なものを選択します。そのため、Hadoop* で使用される主な操作は MPI とよく似ています。ビッグデータに不可欠なフォールト管理とデータ管理の機能はどうでしょうか? MPI にそれらに相当する機能はあるのでしょうか?

Hadoop* は、データの局所性、つまり、できるだけプロセスに近いストレージからデータを取得することによって優れたスケーラビリティーを達成します。一方、HPC は、ネットワーク経由でアクセスする並列ファイルシステムを利用するため、大量の入力データを扱う場合、ネットワークの帯域幅とロードバランスが非常に重要になります。ただし、最近の並列ファイルシステム (Lustre* や GPFS* など) は、システムサイズの増加に伴い、ほぼ線形のスケーラビリティーを提供すると言われています。つまり、スケーラブルなデータ管理に利用できます。インテルが提供する Lustre* は、Lustre* ファイルシステムを利用する Hadoop* で最適なデータアクセスを実現します。[6] Hadoop* アプリケーションの実行で、常に完全なデータの局所性を保持することは不可能ですが、並列ファイルシステムはリモートデータへの高速な並列アクセスを可能にします。

Hadoop* でも HDFS* 以外のファイルシステムを利用しているケースがあります。[7、8] HPC クラスターで利用されているストレージシステムの並列性と高帯域幅インターコネクトは、Hadoop* にも利点をもたらします。例えば、[8] Lustre FS* は、コモディティー・ノード上のローカル・ドライブ・アクセスよりも高速なデータアクセスを提供します。

さらに、MPI-IO には、柔軟にチューニング可能な集合 I/O 操作が含まれています。例えば、集合バッファリングにより、ノード上の 1 つのプロセスのみが FS への書き込み/読み取りを行うようにし、I/O の同時実行を減らすことができます。

フォールトトレラントは、Hadoop* プラットフォームの最大の利点の 1 つと言え、ハードウェア障害からほぼ透過的に回復することができます。同時に、フォールトトレラントは MPI 標準の弱点でもあります。実装によっては、特殊なワークフローに従ってフォールトトレラントの MPI プログラムを記述することも可能でしょうが、[9] そのためにはアプリケーション・エンジニアによる多大な労力を必要とします。MPI 4.0 で導入される可能性があるユーザーレベルのフォールトトレラントは、透過的なフォールト管理を提供しませんが、少なくとも MPI 開発者の負担を軽減できるでしょう。MPI で利用可能な唯一の透過的なフォールト管理メカニズムは、チェックポイントです。[10] グローバル・スナップショットに基づいて、すべてのタスク (正常なものも含む) がチェックポイントから再起動されます。これは、Hadoop* と比較した MPI 実装の数少ない欠点の 1 つです。ストリーミング処理アプリケーションのような実行時間の長いサービスには MPI は適していません。

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

Hadoop* エコシステムのもう 1 つの利点は、さまざまな分野にわたる広範なコミュニティーの存在です。ファイル、リレーショナル・データベース、ストリームなど、さまざまなデータソースへの簡単なアクセスを提供する多数のツールとテクノロジーを利用できます。一方、MPI は、パフォーマンス重視の比較的低水準のテクノロジーであるため、API で多種多様な構文は用意されていません。ただし、インテル® Data Analytics Acceleration Library (インテル® DAAL)[11] のようなテクノロジーにより、さまざまなデータソースからデータを読み取り、変換し、処理する、データフロー全体を効率良く実装する方法 (広範なアルゴリズムを含む) が提供されています。これは、MPI によりパフォーマンスを向上したいと思いながらも、すべてのデータフローを自身で実装しなければならないため諦めていた開発者にとっては朗報と言えます。

また、Vanilla Hadoop* は、シャッフルフェーズでノード間通信に TCP/IP を利用するため、高速インターコネクトの利点を得ることは容易ではありません。ただし、ノード間通信に RDMA を利用することでこの問題を緩和できます。[12]

多くの方は、タスク処理ツールを選択する際に、スケーラビリティーに優れた Hadoop* を検討するかもしれませんが、MPI も適していると言えます。例えば、**インテル® MPI ライブラリー**は、最大 340,000 プロセスまでスケーリングします。[13]

Hadoop* エコシステムで MPI を実行する

前述のように、MPI は新しい HPC の課題に対応することが可能で、特定の分野では Apache* Spark* の代わりに利用できます。しかし、HPC と Hadoop* は、それぞれの分野で一般的なアプリケーションのニーズに応じて並行して進化してきたため、リソース・マネージャーを含め異なるエコシステムを利用しており、MPI と Hadoop* のツールやフレームワークを組み合わせて使用することは容易ではありません。HPC と Hadoop* を融合させるため現在行われている取り組みは、主に Hadoop* のツールを HPC 環境に移行し (または、MR-MPI[14] のように MPI で新たに MapReduce フレームワークを実装し)、ハイエンドシステムによってもたらされるパフォーマンスの利点を利用しようとするものです。データ解析や機械学習アプリケーションを HPC クラスター上で実行して処理を高速化することは 1 つの選択肢ですが (そして、これは最も確実であると考えられますが)、[15] すでに Hadoop* インフラストラクチャーを導入し、運用に成功している企業にとってその利点はないでしょう。

別の選択肢は、クラスター上に異なる 2 つのインフラストラクチャーを保持せずに MPI アプリケーションを Hadoop* 環境で実行するか、2 つの異なるクラスターを構築することです。[16] このセクションでは、MPI を Cloudera Distribution Including Apache Hadoop* (CDH) 上の Hadoop* エコシステムに統合するためのメカニズムを説明します。[17]

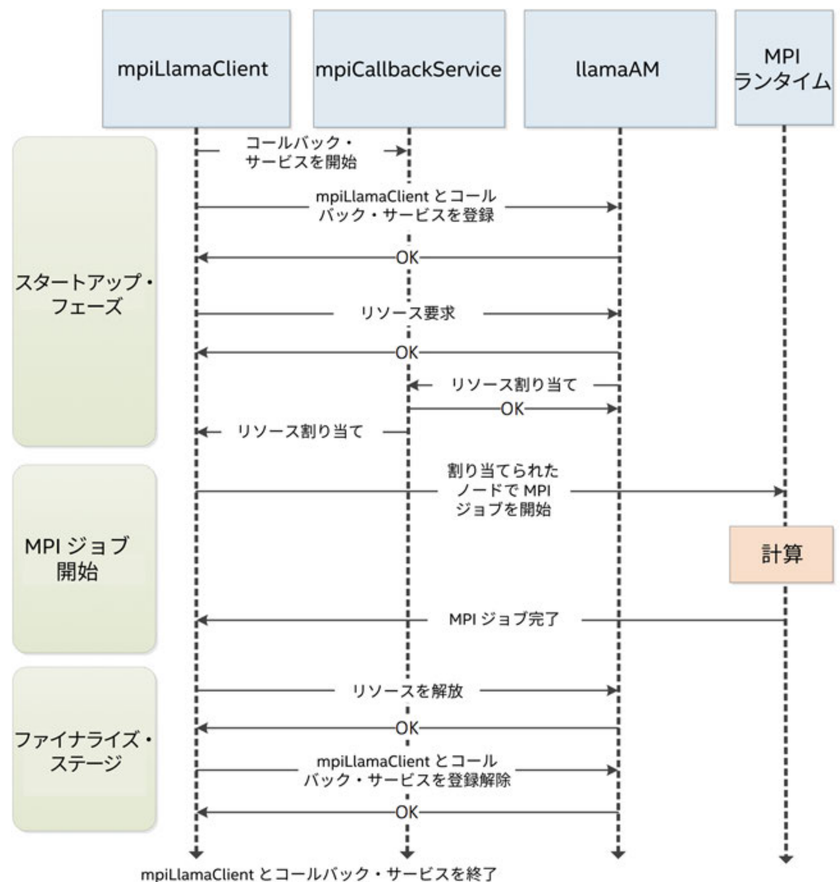
CDH パッケージには、Llama* が含まれています。[18] これは、YARN* のアプリケーション・マスターで、元々 Impala* 用に設計されたものです。[19] Llama* は、リソースの要求/解放とその他の機能 (MPI の実行に不可欠なギャング・スケジュールなど) に関連したクロス言語の Apache* Thrift* API を提供します。この機能セットにより、プログラムで Hadoop* クラスターノードに関する情報を取得し、必要に応じて MPI ジョブ用のリソースを要求して、MPI と Hadoop* ジョブが同一のインフラストラクチャーを動的に共有できるようにします。

Llama* の機能を利用して Hadoop* クラスター上で MPI を実行するには、2 つの補完サービスを実装する必要があります。

- **MPI Llama* クライアント:** 必要な情報 (クラスターノード名など) の取得とリソース要求/解放を Llama* に照会するエンティティー。
- **MPI Llama* コールバック・サービス:** 特定のイベント (ノード割り当てなど) に関する Llama* からの通知を待機するデーモン。

実際のワークフローは、3 つの独立したフェーズで構成されます (**図 1**)。

- **スタートアップ:** 補完サービス (クライアントとコールバック) を起動し、Llama* に登録して、MPI ジョブに必要なリソースを要求します。このフェーズは、コールバック・サービスがリソースの割り当て完了通知を受け取ると終了します。
- **MPI ジョブの開始:** Llama* によって提供されたリソースリストを基に、Hadoop* クラスターノード上で MPI ジョブをネイティブ実行します。
- **ファイナライズ:** MPI ジョブが完了したらリソースを解放し、補完サービスを終了します。



1 Hadoop* クラスターでの MPI の実行

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

このメカニズムの利点の 1 つは、Llama* の通信時間は MPI アプリケーションの複雑さに依存しないため、純粋な MPI 通信と比べてオーバーヘッドが一定であることです。実際のアプリケーションでは、合計ウォールクロック時間に占める Llama* の通信オーバーヘッドの割合はわずかであるため、MPI と Hadoop*/Spark* アプリケーションのパフォーマンス比較に関連するすべての結果に影響しません。

ほかのアプローチ (mpich2-yarn[20] プロジェクトなど) と比較したこのアプローチのもう 1 つの利点は、リソース要求と MPI ジョブの開始をワークフローの別々のフェーズで行っているため、特定の MPI 実装に縛られないことです。つまり、MPI の起動コマンドを変更するだけで、MPI 実装を切り替えることができます。例えば、MPI 実装として BDMPI[21] を利用し、MPI でアウトオブコア・アルゴリズムを効率良く実行して、ネイティブ Hadoop* フレームワークの代わりに、ビッグデータ問題に柔軟に対応できます。

前述の機能は、インテル® MPI ライブラリー 5.1 Update 2 で実装されています (使用モデルについては、『**インテル® MPI ライブラリー・リファレンス・ガイド**』を参照してください)。

Hadoop* クラスター上で MPI を実行する場合、一般にいくつかの制限があります。

- MPI 実装は HDFS をサポートしていないため、MPI-IO は共有フォルダーでのみ、またはローカルで利用できます。そのため、クラスターの管理者は、HDFS に加えて、NFS フォルダーもセットアップしたほうが良いでしょう。
- YARN* は、コンテナーの CPU アフィニティー情報を提供しないため、同じノードで複数のアプリケーションを実行する場合、ピニング機能の使用には細心の注意を払うべきです。

パフォーマンスの評価

データ解析でよく使用される 2 つの分散処理アルゴリズムを使用します。

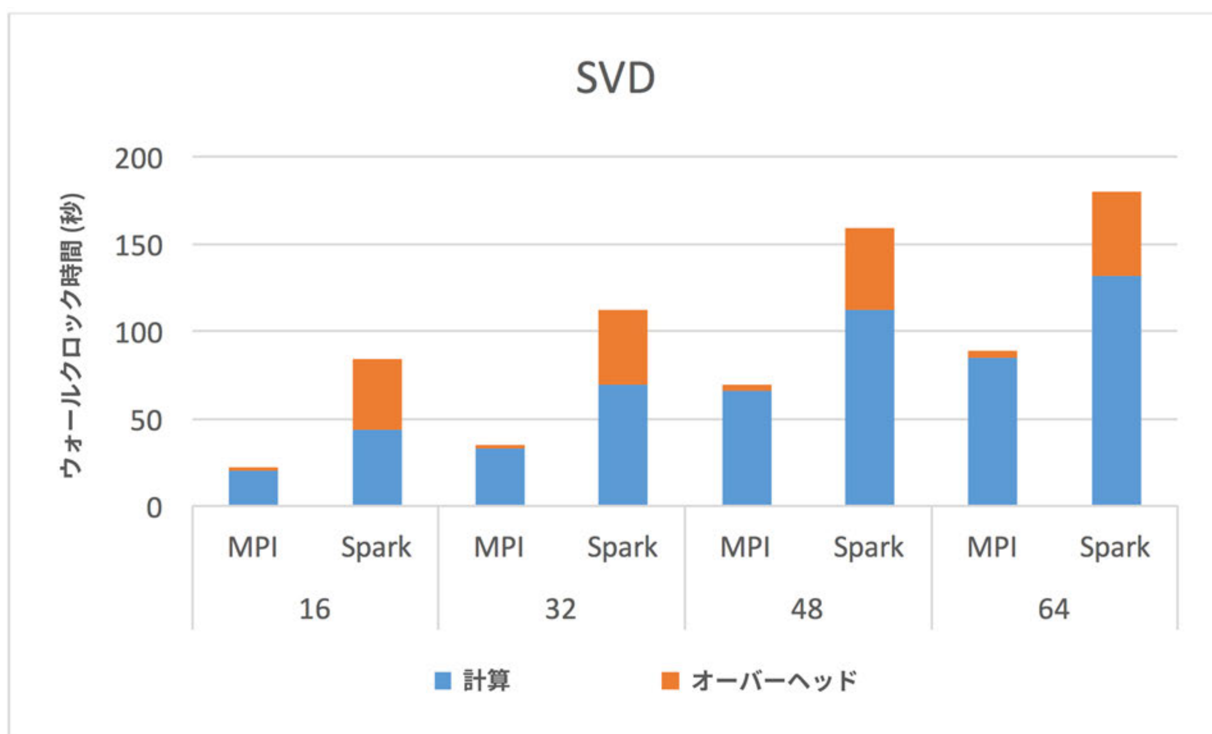
1. 特異値分解 (SVD)
2. 主成分分析 (PCA、相関メソッドを使用)

インテル® DAAL に含まれる各アルゴリズムの Spark* および MPI 向けのサンプルを使用します。これらのサンプルは、同じビルディング・ブロックで構成されているため、実装の詳細ではなく、フレームワーク (Spark* と MPI) および言語 (Java* と C) によるパフォーマンスの違いを説明することができます。Spark* 用の SVD サンプルは、ドライバーノードで左直交行列を収集しないように変更しました。この収集は一般に過剰にデータを収集し、Spark* RDD の利点が得られないためです (そのため、Spark* 用アルゴリズムは、MPI 用アルゴリズムよりも複雑さが緩和されています)。

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。

パフォーマンスの評価には、8 ノードのクラスターを使用しました。各ノードには、1 基のインテル® Xeon® プロセッサー X5570、1 つの NetEffect* NE020 10GB イーサネット・アダプター、12GB RAM が搭載されています。また、動作環境として、SLES* 11.0 Linux* オペレーティング・システム、CDH の Cloudera Express* 5.4.6 バージョン (Spark* 1.3.0)、インテル® MPI ライブラリー 5.1.2、インテル® DAAL 2016 Update 1、インテル® C++ コンパイラー 15.0.4、JDK 1.7.0_67、Scala* 2.10.4 を使用しました。

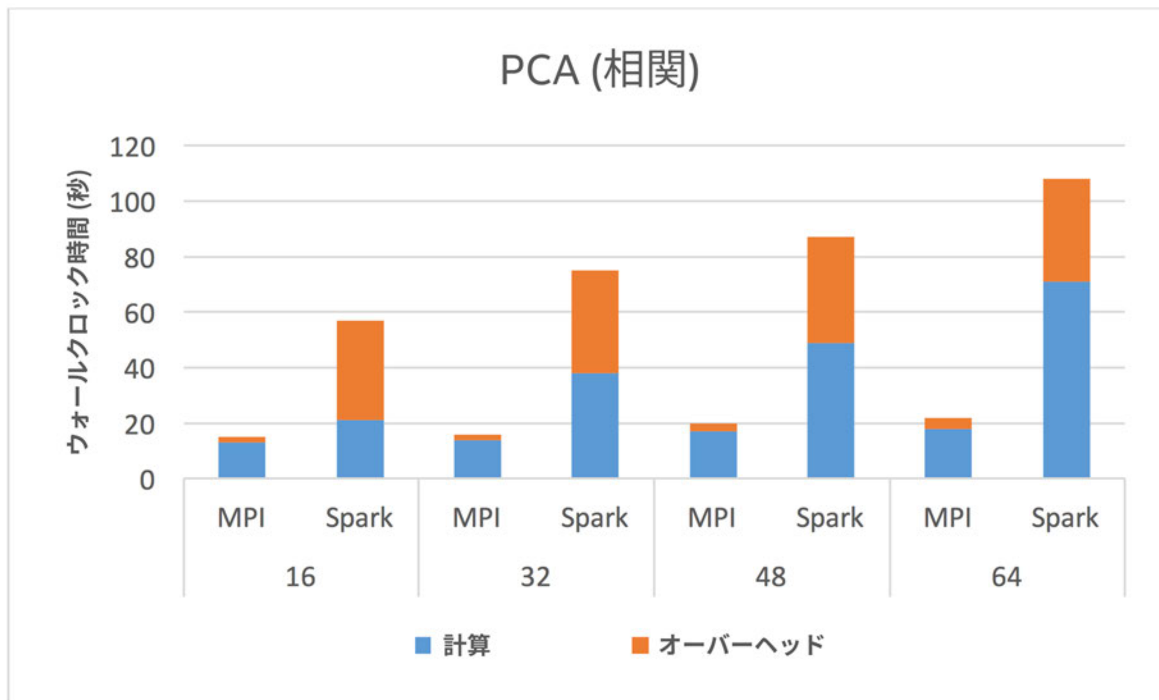
それぞれ 10,000 x 1,000 要素の入力行列を含む、16、32、48、および 64 データブロックでパフォーマンスを測定しました。64 ブロックのデータセットのサイズは 4.2GB です。MPI サンプルの実行には、1 ブロックにつき 1 MPI プロセスを開始しました。Spark* では、spark.executor.instances プロパティーでエグゼキューターの数をブロック数に設定しました。Spark* は独自のヒューリスティックに従ってエグゼキューターの数を定義するため、常にこの設定が尊重されるわけではありません。また、Spark* は JVM と独自の設定に影響されやすいため、経験に基づいて、パフォーマンスを最適化するようにメモリー設定を選択しました (**図 2** と **3** を参照)。



2 特異値分解

spark.executor.memory = 1300m
spark.yarn.executor.memoryOverhead = 1024m
spark.kryoserializer.buffer.max.mb = 512m

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。



3 主成分分析

spark.executor.memory = 1024m
spark.yarn.executor.memoryOverhead = 768m

上記の合計ウォールクロック時間は、実際の計算処理とオーバーヘッドから成ります。MPI サンプルのオーバーヘッドは、インテル® MPI ライブラリーが Llama* とリソースについてネゴシエートし、MPI ジョブを開始するのに必要な時間で、Spark* サンプルのオーバーヘッドは、ウォールクロック時間からすべての計算処理ステージの実行時間の合計を引いたものです。

MPI は、オーバーサブスクリプション・モードでの実行の影響を受けやすいため、MPI ランクの数が物理コア数よりも大きい場合、パフォーマンスがやや低下することがあります。それでも、ウォールクロック時間が Spark* よりも短いことが分かります。

まとめ

パフォーマンスの評価結果と参考文献の結果は、[3、4] MPI がデータ解析で使用される一部のアルゴリズムに適していることを示しています。また、この記事で紹介したように MPI を Hadoop* エコシステムに統合することで、この分野での MPI の利用を拡大できます。

ここで重要なことは、MPI を Hadoop* ベースのフレームワークの代わりに使用し、優れたパフォーマンスが得られる問題もありますが、そうでない問題もあるということです。そのため、2 つを組み合わせて利用することは自然であり、それにより新たな相乗効果が得られます。実行時間の長いサービスは Hadoop* ツールで実装し、計算負荷が高く、複雑な通信を含む、比較的実行時間の短いタスクには MPI を利用します。

MPI は、優れたパフォーマンスをもたらすだけでなく、Hadoop* 開発者は、長年にわたって記述された数百万行の MPI コードを活用することができます。さらに、データ解析が HPC 分野に拡大し、新たなハイパフォーマンス・データ解析分野が出現しつつある中、MPI とビッグデータ・フレームワークを同じエコシステムで利用すべきかどうかという問題は、もっと注目されるべきです。MPI と Hadoop* エコシステムの統合は、その 1 つの選択肢と言えます。

BLOG HIGHLIGHTS

コードの現代化を成功させるための 3 つのアドバイス

CLAY BRESHEARS >

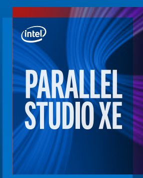
プログラムを高速化するための開発者への 3 つのアドバイスについてブログの執筆依頼を受け取ったとき、最初に思い浮かんだのは「Location! Location! Location!」という言葉でした。そこから不動産を連想し、映画「Glengarry Glen Ross (邦題: 摩天楼を夢見て)」のブレイク (Alec Baldwin) の台詞「A-B-C. A-Always, B-Be, C-Concurrent. Always be concurrent.」を思い出しました。

この単純な台詞には、さまざまなものが凝縮されています。私はこれまで、粒度、ロードバランス、タスク分割、ループの並列化など、個々のトピックについて多数の IDZ ブログを執筆してきましたが、正にそれらがアドバイスと言えます。(それらのブログを見つけ、そのうち 3 つを読めば、この記事を最後まで読む必要はないかもしれません。あるいは、この記事を読んだ後に、個別のブログで詳細を確認するほうが良いでしょう。) また、コンカレントおよび並列プログラミングに関する本も執筆しました。お手元にある方は、後半の任意の 3 つのページをお読みになれば、コードを高速化するための 3 つのヒントが得られるでしょう。

この記事の続きはこちらでご覧になれます。 >

参考文献

1. J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters." Comm. ACM, 51(1): 107–113, January 2008.
2. Apache Hadoop Wiki, **hadoop.apache.org/#What+Is+Apache+Hadoop%3F**.
3. S. Jha, J. Qiu, A. Luckow, P. Mantha, and G. C. Fox. "A Tale of Two Data-Intensive Paradigms: Applications, Abstractions, and Architectures," eprint arXiv:1403.1528, March 2014.
4. F. Liang, C. Feng, X. Lu, and Z. Xu. "Performance Benefits of DataMPI: A Case Study with BigDataBench." In The 4th Workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware, BPOE-4, Salt Lake City, Utah, 2014.
5. T. Hoefler, A. Lumsdaine, and J. Dongarra. "Towards Efficient MapReduce Using MPI." Recent Advances in Parallel Virtual Machine and Message Passing Interface, pp. 240–249, 2009.
6. "Intel® Enterprise Edition for Lustre*," **intel.com/content/www/us/en/software/intel-enterprise-edition-for-lustre-software.html**.
7. A. Woodie. "What Can GPFS on Hadoop Do For You?" Datanami, February 2014, **datanami.com/2014/02/18/what_can_gpfs_on_hadoop_do_for_you_/**.
8. N. Rutman. "Map/Reduce on Lustre: Hadoop Performance in HPC Environments" (technical white paper). Xyratex, **xyratex.com/sites/default/files/Xyratex_white_paper_MapReduce_1-4.pdf**.
9. W. Gropp and E. Lusk. "Fault Tolerance in MPI Programs." The International Journal of High Performance Computing Applications, Volume 18, No. 3, Fall 2004, pp. 363–372.
10. J. Hursey, J. M. Squyres, and A. Lumsdaine. "A Checkpoint and Restart Service Specification for Open MPI" (technical report), **open-mpi.org/papers/iu-cs-tr635/iu-cs-tr635.pdf**.
11. ベータ版 インテル® Data Analytics Acceleration Library 2016 のリリース, **isus.jp/products/daal/announcing-intel-daal-2016-beta**.
12. High-Performance Big Data Project, Network-Based Computing Laboratory, Ohio State University. "RDMA-Based Apache Hadoop," **hibd.cse.ohio-state.edu**.
13. インテル® MPI ライブラリー, **isus.jp/intel-mpi-library**.
14. S. Plimpton. "MapReduce and MPI." SOS 17 - Intersection of HPC & Big Data, March 2013.
15. J. Dursi. "HPC Is Dying, and MPI Is Killing It." **dursi.ca/hpc-is-dying-and-mpi-is-killing-it**.
16. The Nielsen Company. "Bridging the Worlds of High Performance Computing and Big Data," **sites.nielsen.com/newscenter/bridging-the-worlds-of-high-performance-computing-and-big-data**.
17. "CDH Components," **cloudera.com/content/cloudera/en/products-and-services/cdh.html**.
18. Cloudera, Inc. "Llama," **cloudera.github.io/llama**.
19. Cloudera, Inc. "Apache Impala," **impala.io**.
20. GitHub, Inc. "mpich2-yarn," **github.com/alibaba/mpich2-yarn**.
21. Karypis Lab. "BDMPI - Big Data Message Passing Interface," **glaros.dtc.umn.edu/gkhome/bdmpi/overview**.



インテル® MPI ライブラリーを評価する

インテル® Parallel Studio XE Cluster Edition に含まれます >