ケーススタディー

ソフトウェア/サービス
ビッグデータ解析

ビッグデータ・プラットフォームの高速化

Software

MeritData Inc. はインテルのハイパフォーマンス・ライブラリーを使用してビッグデータ・アルゴリズムと視覚化のパフォーマンスを向上

「インテルのエンジニアとのコラボレーションにより、インテル® DAAL およびインテル® MKL を使用してビッグデータ解析プラットフォーム Tempo のアルゴリズムを最適化したところ、パフォーマンスと顧客のユーザー体験が大幅に向上しました。さらなるコラボレーションを楽しみにしています。」

—MeritData Inc.
データマイニング・アルゴリズム・アーキテクト
Jin Qiang 氏

MeritData, Inc. (英語) は、中国有数のビッグデータ解析テクノロジー/サービス・プロバイダーです。同社の Tempo ビッグデータ・プラットフォームは、有名企業およびクラウド・サービス・プロバイダーにより、電力、製造、金融サービスなどの業界で広く利用されています。ハイパフォーマンス・コンピューティング・テクノロジー、最先端のデータ解析アルゴリズム、高次元の視覚化、クリエイティブなデータ視覚化言語を統合することにより、MeritData は、データ処理、データマイニング、データ視覚化ソリューションを使用して、顧客がデータから有用な情報を抽出できるように (内部および外部のユーザーがデータを効率的に処理できるように) 支援します。

データを素早く正確に解析するには、アルゴリズムの原則、計算、プログラミングのすべての観点からアルゴリズムを最適化する必要があります。MeritData のアルゴリズム・エンジニアは、インテルと協力して、エクストリーム・ラーニング・マシン (ELM)、L1/2 スパース反復、線形回帰 (LR)、QR 行列因数分解、主成分分析 (PCA) を含む、複数のデータマイニング・アルゴリズムの最適化に取り組みました。インテル® データ・アナリティクス・アクセラレーション・ライブラリー (インテル® DAAL) およびインテル® マス・カーネル・ライブラリー (インテル® MKL) でアルゴリズムを最適化したところ、パフォーマンスは平均で 3 倍、ピークで 14 倍に向上しました。

ビッグデータ解析プラットフォームにおけるスピードの重要性

世界的に、データの量は指数的に増加しており、ビッグデータ解析市場も活気を帯びています。ほぼすべての大企業は、ビッグデータ解析にリソースを投資し、数年分の過去に生成されたデータと新しく生成されるデータの両方を統合して、データから有用な情報を素早く正確に抽出することを目指しています。

MeritData の Tempo ビッグデータ解析プラットフォームは、さまざまな顧客向けにさまざまなデータタイプを格納して処理します。すべてのデータを効率良く解析し、増加する大量のデータセットを高速に処理するには、アルゴリズムのパフォーマンスを大幅に向上する必要があります。

アルゴリズムのモデリングは、入力データに対して繰り返し計算を行う必要がある負荷の高い計算に役立ちます。データの量が少ない場合、実行時間は許容範囲内に収まります。しかし、データの量が急増するとともに、一部のアルゴリズムの実行時間は指数的に増加し、顧客の要件を満たせなくなります。

MeritData は、インテルの協力の下、インテル® DAAL およびインテル® MKL を使用して Tempo のコア・アルゴリズム・ライブラリーを高速化し、顧客に強力なデータ解析ソリューションを提供することができました。オリジナルのハードウェア依存の実装と比較して、新しいスキームでは大量のデータ処理およびモデリングを素早く正確に解析し、顧客が素早く有用な情報を抽出できるようになりました。

ソリューション: Tempo プラットフォーム

MeritData は、インテル® アーキテクチャー上でコア・アルゴリズム・ライブラリーを高速化するため、インテルとともに、インテル® MKL およびインテル® DAAL を使用した Tempo ビッグデータ解析プラットフォームの構築に取り組みました。チームは、クラウド・コンピューティング・アーキテクチャーを利用して、高速なモデリングおよび解析向けのビッグデータ解析ソリューションを実装しました。同時に、データ解析の分野やレベルの異なる顧客の要望を満たすために、データの調査、データの視覚化、詳細な解析などを統合したサービスを提供しました。

Tempo プラットフォームのシステム・アーキテクチャーは、データアクセス層、解析およびモデリング、結果表示、アクセス層を含み、クラウドサービスのアクセス、クラウドリソースのスケジュール、クラウド・プラットフォームの管理などの機能を提供します。

データアクセス層は、SQL および NoSQL データベースへの適応、Kafka*、flume ストリーミング・データ・ソース、非構造化テキスト・データ・ソースとの結合を含む、異なるデータソースへの対応を行います。

解析およびモデリング層

MeritData は、インテル® MKL およびインテル® DAAL を使用して Tempo のコア解析アルゴリズムを高速化しました。このインテリジェントなデータ解析プラットフォームは、次の処理を行います。

- ノード・スケジュールの管理
- 割り当てられた計算ノードの解析
- 関連するジョブのログの書き込み

結果表示およびアクセス層

解析モデリングが完了すると、モデリングの結果が統計として表示され、リソースの使用状況の改善と迅速な意思決定に役立つ結果レポートが生成されます。結果に基づいて、新しいデータソースを処理して正確な予測を行う新しいモデルを構築することもできます。将来の開発に対応するため、この層は API 呼び出し形式のアクセス・インターフェイスを提供しています。

技術的な利点

インテル® MKL は、インテル® アーキテクチャー上の複数レベルの並列処理向けに高度に最適化されている幅広い行列、ベクトル、演算処理ルーチンを提供します。これらのルーチンは、マルチコア、メニーコア、クラスター・アーキテクチャーの利用可能なリソースを効率的に利用して、インテル® Xeon® プロセッサー・ベースのノード全体にわたってワークロードのバランスをとります。高度に最適化された関数 (図 2) は、高い計算パフォーマンスが求められる科学、工学、金融サービス向けアプリケーションで広く利用されており、インテル® プロセッサーの性能を最大限に引き出して、アプリケーションのパフォーマンスを向上し、開発時間を短縮できます。

インテル® DAAL は、インテル® MKL と同様のハイパフォーマンス・ライブラリーですが、インテル® MKL よりも多くのビッグデータ向けの機能を備えています。オフライン、ストリーミング、分散解析を使用するすべてのデータ解析段階 (前処理、変換、解析、モデリング、検証、意思決定) 向けに高度に最適化されたアルゴリズム・ビルディング・ブロックを提供することにより、ビッグデータ解析の高速化を支援します。

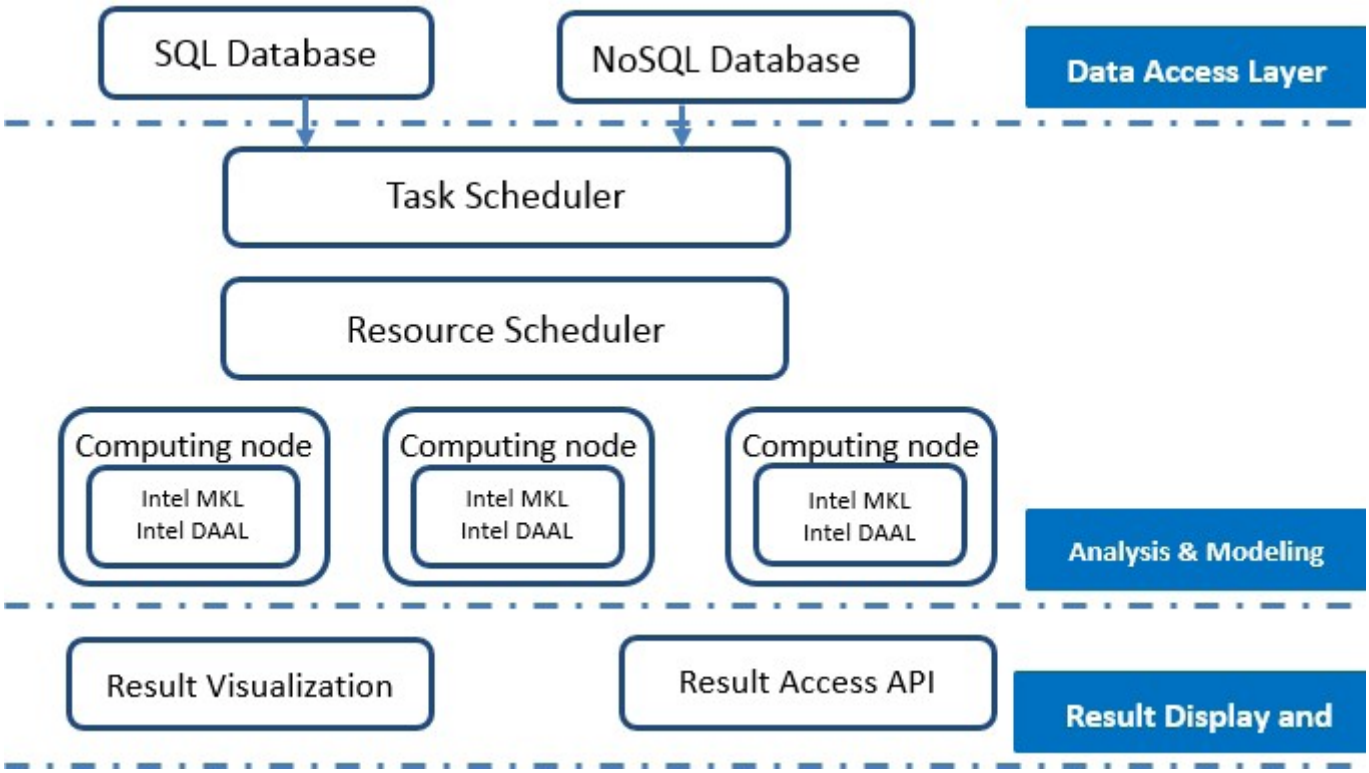


図 1. データマイニング・システム・アーキテクチャー向け Tempo ビッグデータ解析プラットフォーム

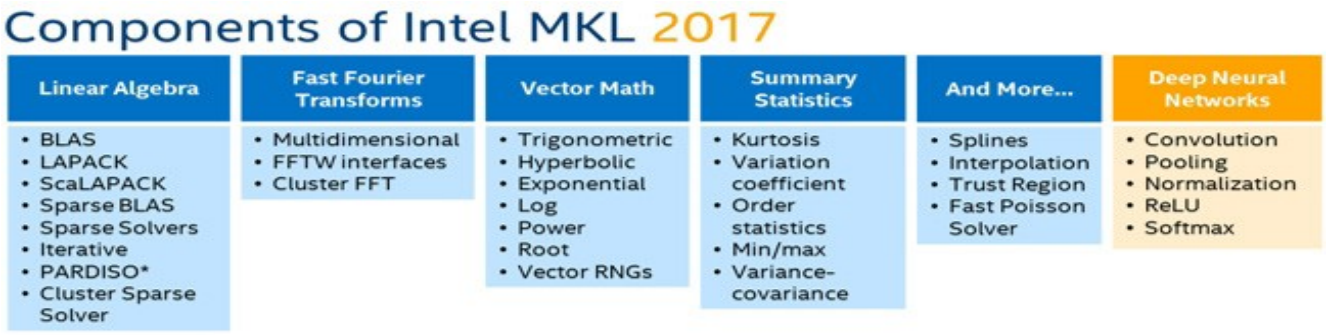


図 2. インテル® MKL のコンポーネント

また、Hadoop*、Spark*、R、MATLAB* を含む一般的なデータ・プラットフォームで効率良いデータアクセスを行えるように設計されています。

インテル® DAAL (図 3) は、データセットの基本的な記述統計から高度なデータマイニングおよびマシンラーニングにわたる、豊富なアルゴリズムを提供します。ビッグデータ開発者が、比較的少ない労力で、多くのビッグデータ・アルゴリズム向けに高度に最適化されたコードを作成できるように支援します。

Tempo のパフォーマンスを向上

MeritData は、インテル® MKL およびインテル® DAAL を使用して、Tempo ビッグデータ解析プラットフォームのコア解析アルゴリズムを高速化しました。オリジナルの手法よりもパフォーマンスは大幅に向上し、顧客のデータの価値の最大化、ビジネスニーズへの対応、大量の新しいデータの迅速な解析に大いに貢献しました。

具体的には、チームは、Tempo のエクストリーム・ラーニング・マシン (ELM) アルゴリズムと L1/2 スパース反復アルゴリズムを最適化しました。

ELM アルゴリズム

ELM は MeritData のデータマイニング・アルゴリズムの 1 つです。チームは、ELM アルゴリズムのコードをプロファイルし、行列計算 (行列-行列乗算などの逆関数を含む) がパフォーマンス・ボトルネックになっていることを特定しました。元々、MeritData は複数の行列乗算に Eigen ライブラリーを使用していましたが、関連する関数をインテル® MKL の関数に置換することにしました。コードの置換は非常に単純でした (表 1)。

ELM アルゴリズムで異なるデータ量を使用してテストした結果を図 4 に示します。最適化により、アルゴリズムのパフォーマンスは平均で約 12 倍に向上しました。

L1/2 スパース反復アルゴリズム

チームは、テストおよび解析の後、このアルゴリズムではカーネル関数と基本行列演算がパフォーマンス・ボトルネックになっていることを特定しました。そこで、基本行列演算にインテル® MKL BLAS 関数を使用し、カーネル関数にインテル® DAAL カーネル関数を使用しました。表 2 は、インテル® DAAL による最適化前と最適化後のコードを示しています。

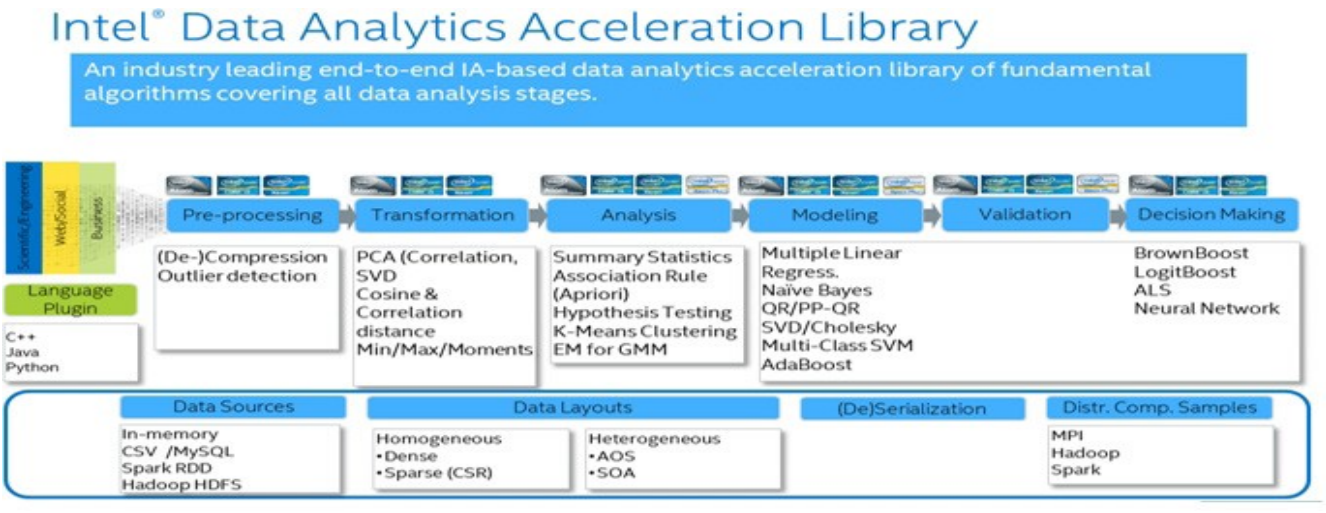


図 3. インテル® DAAL のコンポーネント

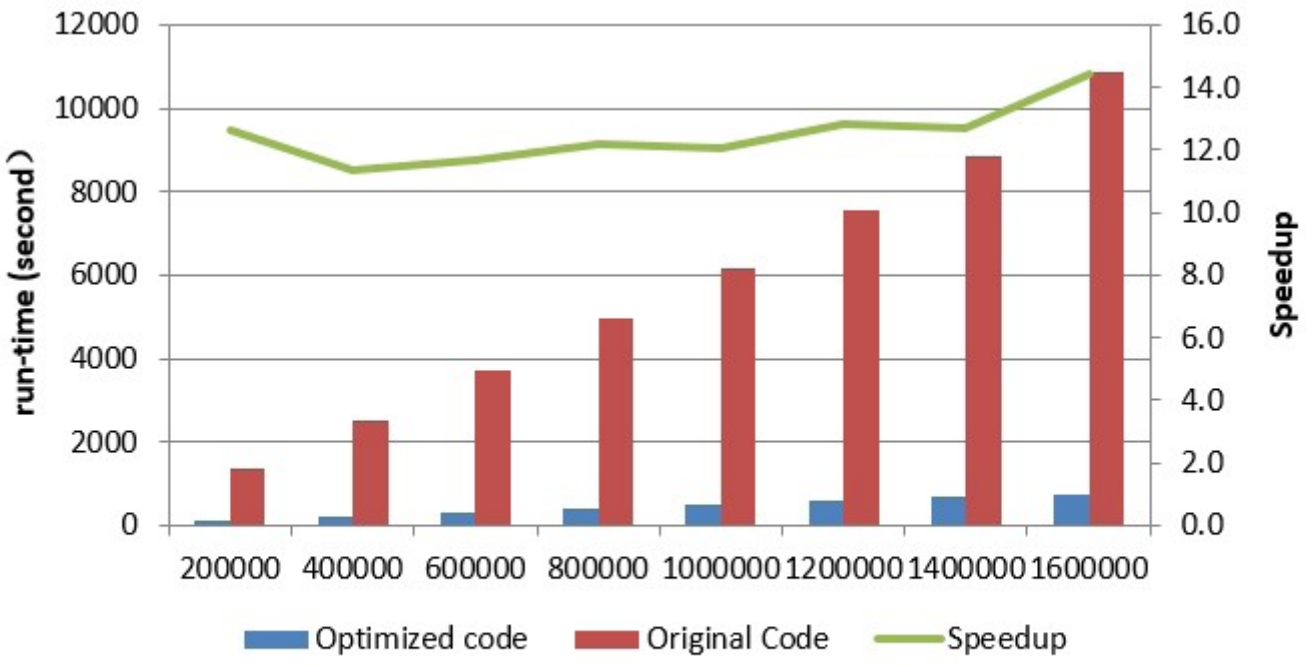
Original Version

```
Eigen::MatrixXd Pk;
Eigen::MatrixXd Bk;
jobjectArray     update(JNIEnv*
env,jobject,jobjectArray
HM,jobjectArray TM){
// get row num of data.
int height = env->GetArrayLength(HM);
// get column num of data.
…
int width = env->GetArrayLength(tmp);
// internal matrix
Eigen::MatrixXd mat(height,width);
…
// TM matrix
Eigen::MatrixXd Tm(row2,col2);
…
// do caculations
Eigen::MatrixXd     matTran     =
mat.transpose();
Eigen::MatrixXd matTmp = matTran*mat;
...
// inverse matrix
Pk = matTmp.inverse();
Bk = Pk*matTran*Tm;
…
}
```

Optimized by Intel® MKL

```
double* Pk;
double* Bk;
jobjectArray update(JNIEnv*
env,jobject,jobjectArray HM,jobjectArray TM){
// get row num of data.
int rowNum = env->GetArrayLength(HM);
// get column num of data.
…
int colNum = env->GetArrayLength(tmp);
// internal matrix
double* mat = (double
*)mkl_calloc(rowNum*colNum, sizeof( double ),
64);…
// TM matrix
double* Tm = (double
*)mkl_calloc(rowNumT*colNumT,
sizeof( double ), 64); …
// do caculations
double* matTmp = (double
*)mkl_calloc(colNum*colNum, sizeof( double ),
64);
MKL_INT lda = colNum,ldc = colNum;
cblas_dgemm(CblasRowMajor,CblasTrans,CblasNoT
rans,colNum,colNum,rowNum,1.0,mat,lda,mat,co
Num,0.0,matTmp,ldc);
...
// inverse matrix
MKL_INT* ipiv = (int *)mkl_calloc(colNum,
sizeof( int ), 32);
MKL_INT info =
LAPACKE_dgetrf(LAPACK_ROW_MAJOR,colNum,colNum
,matTmp,lda,ipiv);
info =
LAPACKE_dgetri(LAPACK_ROW_MAJOR,colNum,matTmp
,lda,ipiv);
…
}
```

表 1. インテル® MKL による最適化前と最適化後の ELM アルゴリズムのコードの比較



Configuration Info - Versions: Intel® Math Kernel Library (Intel MKL) 11.3.2; Hardware: Intel® Xeon CPU E5-2699 V3 2.30GHz, 2 sockets x 18 cores, AVX 2.0Supported. 45MB Cache, 128 GB Memory; Operating System: Centos6.7; 4 node cluster with spark-1.5.1, hadoop-2.6.0, jdk-7u79-linux-x64, scala-2.10.4. Benchmark Source: MeritData test code and test data set

図 4. インテル® MKL による最適化前と最適化後の ELM アルゴリズムの実行時間の比較

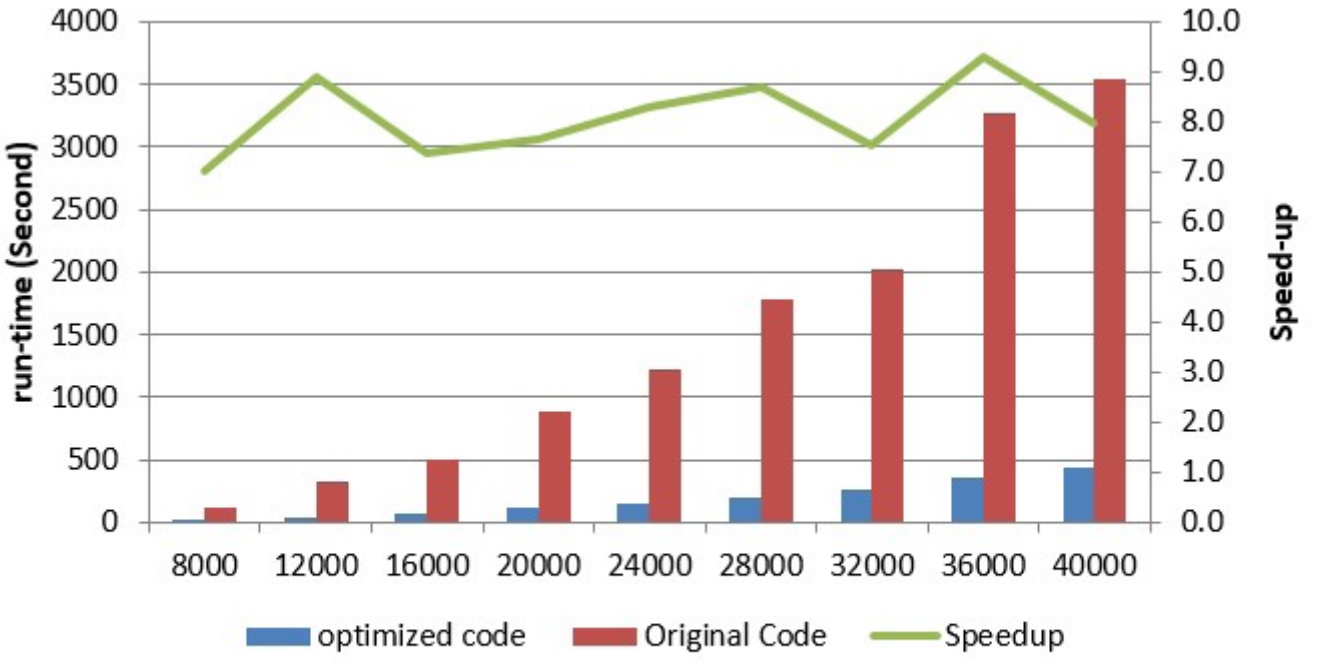
Original Code

```
public Matrix getKernelMatrix()
throws Exception {
Matrix result = new
Matrix(m_data.numInstances(),
m_data.numInstances(), 0);
for (int i = 0; i <
m_data.numInstances() - 1; i++) {
for (int j = i + 1; j <
m_data.numInstances(); j++) {
result.set(i, j, evaluate(i, j,
m_data.instance(0)));
}
}
result = result.plus(
result.transpose().plus(
Matrix.identity(m_data.numInstances()
,
m_data.numInstances()))).copy();
return result;
}
```

Optimized by Intel® DAAL

```
jobject getKernelMatrix(JNIEnv*
env,jobject,jdouble param,jint rows,jint
cols,jobject byteBuffer,jobject dstBuffer){
…
kernel_function::linear::Batch<>
linearKernel;
/* Set the kernel algorithm parameter */
linearKernel.parameter.k = 1.0;
linearKernel.parameter.b = 1.0;
linearKernel.parameter.computationMode =
kernel_function::matrixMatrix;
/* Set an input data table for the algorithm
*/
linearKernel.input.set(kernel_function::X,
data);
linearKernel.input.set(kernel_function::Y,
data);
/* Compute the linear kernel function */
linearKernel.compute();
/* Get the computed results */
services::SharedPtr<kernel_function::Result>
lkResult = linearKernel.getResult();
/* Get the results */
services::SharedPtr<NumericTable> lkMat =
lkResult->get(kernel_function::values);
BlockDescriptor<double> block;
lkMat->getBlockOfRows(0, rows, readOnly,
block);
...
}
```

表 2. インテル® DAAL による最適化前と最適化後の L1/2 スパース反復アルゴリズムのコードの比較



Configuration Info - Versions: Intel Data Analysis Acceleration library from Parallel_studio_xe_2017_beta; Hardware: Intel® Xeon CPU E5-2699 V3 2.30GHz, 2 sockets x 18 cores, AVX 2.0Supported. 45MB Cache, 128 GB Memory; Operating System: Centos6.7; Benchmark Source: MeritData test code and test data set

図 5. インテル® DAAL による最適化前と最適化後の L1/2 スパース反復アルゴリズムの実行時間の比較

L1/2 スパース反復アルゴリズムで異なるデータ量を使用してテストした結果を図 4 に示します。最適化により、アルゴリズムのパフォーマンスは約 8 倍に向上しました。

まとめ

データを深く理解する必要性の高まりとともに、MeritData の顧客は、数年分の過去に生成されたデータと新しく生成されるデータの両方を統合すること、および限られたリソースでデータから有益な情報を抽出することを望んでいます。MeritData の Tempo プラットフォームは、データマイニング・タスク向けの高速かつ効率良いソリューションを提供します。

インテルのエンジニアとのコラボレーションにより、MeritData は Tempo のアルゴリズムの最適化にインテル® DAAL および インテル® MKL を採用しました。その結果、MeritData の顧客はビッグデータを高速かつ正確に解析して、異なる種類とレベルのデータ解析をモデル化できるようになりました。

このソリューションにより、データ解析処理のキャパシティー、パフォーマンス、ユーザー体験は大きく向上しました。

MeritData は、インテルと協力して、インテル® アーキテクチャーの計算能力を最大限に引き出すように、Tempo ビッグデータ解析プラットフォームの最適化を続ける予定です。MeritData の開発者は、今後も、少ない労力で高いパフォーマンスを実現し、顧客に最高のユーザー体験を提供することができるでしょう。

関連資料

インテル® DAAL

インテル® MKL

インテル® テクノロジーの機能と利点はシステム構成によって異なり、対応するハードウェアやソフトウェア、またはサービスの有効化が必要となる場合があります。

実際の性能はシステム構成によって異なります。絶対的なセキュリティーを提供できるコンピューター・システムはありません。詳細については、各システムメーカーまたは販売店にお問い合わせいただくか、https://www.intel.co.jp/ を参照してください。

インテル® コンパイラーでは、インテル® マイクロプロセッサーに限定されない最適化に関して、他社製マイクロプロセッサー用に同等の最適化を行えないことがあります。これには、インテル® ストリーミング SIMD 拡張命令 2、インテル® ストリーミング SIMD 拡張命令 3、インテル® ストリーミング SIMD 拡張命令 3 補足命令などの最適化が該当します。インテルは、他社製マイクロプロセッサーに関して、いかなる最適化の利用、機能、または効果も保証いたしません。本製品のマイクロプロセッサー依存の最適化は、インテル® マイクロプロセッサーでの使用を前提としています。インテル® マイクロアーキテクチャーに限定されない最適化のなかにも、インテル® マイクロプロセッサー用のものがあります。この注意事項で言及した命令セットの詳細については、該当する製品のユーザー・リファレンス・ガイドを参照してください。注意事項の改訂 #20110804

性能に関するテストに使用されるソフトウェアとワークロードは、性能がインテル® マイクロプロセッサー用に最適化されていることがあります。SYSmark* や MobileMark* などの性能テストは、特定のコンピューター・システム、コンポーネント、ソフトウェア、操作、機能に基づいて行ったものです。結果はこれらの要因によって異なります。製品の購入を検討される場合は、他の製品と組み合わせた場合の本製品の性能など、ほかの情報や性能テストも参考にして、パフォーマンスを総合的に評価することをお勧めします。詳細については、https://www.intel.com/performance (英語) を参照してください。

インテルは、本資料で参照しているサードパーティーのベンチマーク・データまたは Web サイトの設計や実装について管理や監査を行っていません。本資料で参照している Web サイトまたは類似の性能ベンチマーク・データが報告されているほかの Web サイトも参照して、本資料で参照しているベンチマーク・データが購入可能なシステムの性能を正確に表しているかを確認されるようお勧めします。

この文書および情報は、インテルのお客様向けの参考情報として記載されているものであり、現状のまま提供され、明示されているか否かにかかわらず、いかなる保証もいたしません。ここにいう保証には、商品適格性、特定目的への適合性、知的財産権の非侵害性への保証を含みますが、これらに限定されるものではありません。本資料は、本資料に記述、表示、または記載されたいかなる知的財産権のライセンスも許諾するものではありません。インテル製品は、医療、救命、延命措置、重要な制御または安全システム、核施設などの目的に使用することを前提としたものではありません。

© 2018 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴ、Xeon は、アメリカ合衆国および / またはその他の国における Intel Corporation の商標です。

* その他の社名、製品名などは、一般に各社の表示、商標または登録商標です。

JPN/1803/PDF/XL/SSG/SS