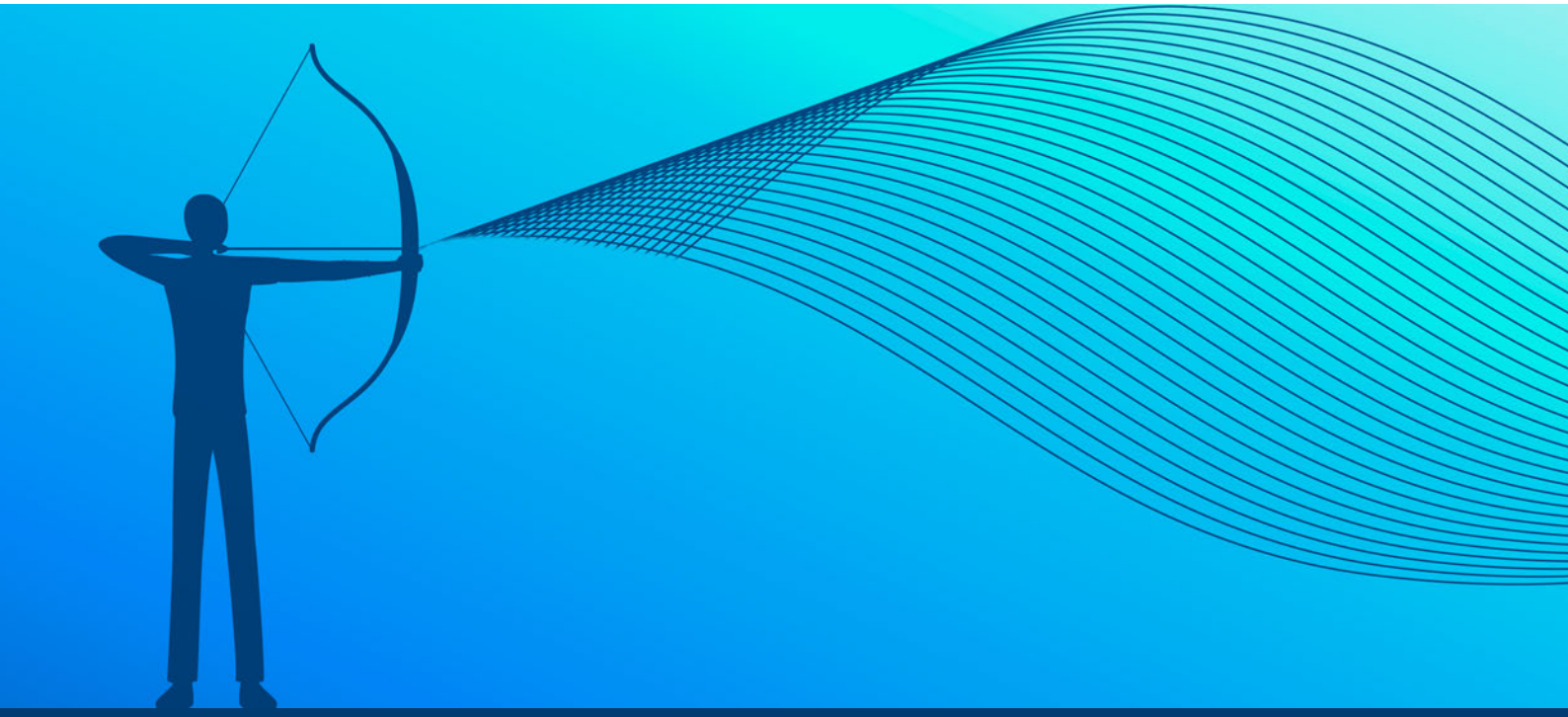


インテル® IPP を利用した画像識別の最適化

Tencent* の MD5 画像識別スピードが 2 倍に向上

Yueqiang Lu インテル APAC R&D Ltd. アプリケーション・エンジニア
Ying Hu インテル APAC R&D Ltd. テクニカル・コンサルティング・エンジニア

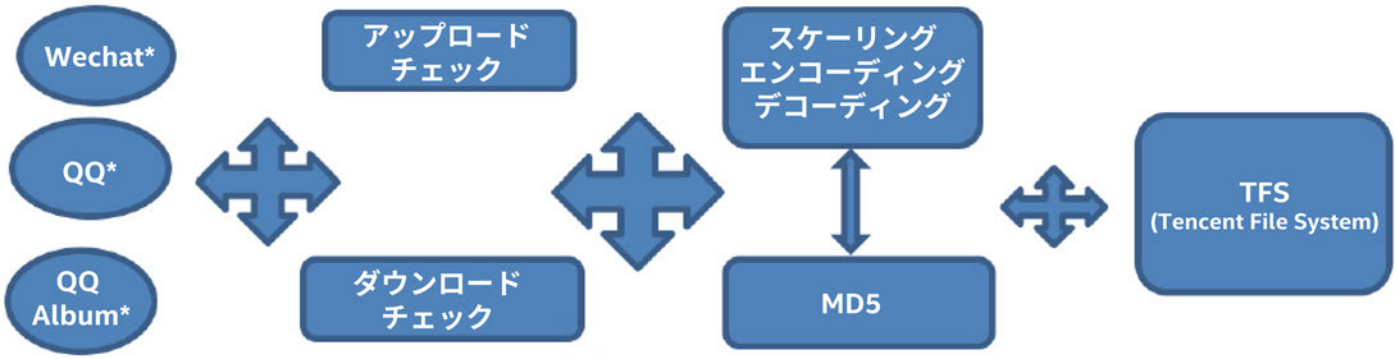
Tencent, Inc. は、中国におけるインターネット・サービス・ポータルの最大手です。中国最大のオンライン・ゲーム・コミュニティーと Web ポータル (**qq.com**) に加え、トップ 2 つのアプリケーション (WeChat*、QQ*) を所有しています。

毎日、Tencent* では WeChat*、QQ*、QQ Album* でユーザーにより生成された数十億もの画像を処理しています。人気の高いアプリケーションでは、1 日で数億もの画像がアップロード、格納、処理、そしてダウンロードされます—そのため、大量の計算リソースを必要とします。

これらの画像を管理、格納、処理するため、Tencent* は Tencent File System (TFS) を開発しました。しかし、画像サイズは圧縮しても数百ペタバイトに達します。また、爆発的な成長が続いており、クラスターは 20,000 を超えるサーバーにより構成されています。

技術背景

TFS ベースの画像処理システムは、アップロード、エンコード、ダウンロード・サービスを提供します。画像アップロードでは、異なる解像度にスケーリングし、メッセージ・ダイジェスト・アルゴリズム 5 (MD5) を用いて関連 ID を生成します。[1] そして、格納用に画像を WebP* 形式に変換します。画像のダウンロード中、システムは画像の正しい読み取り位置を特定し、ユーザーが要求する形式と解像度に変換する必要があります (**図 1**)。



1 Tencent File System* の画像処理

毎秒、膨大な数の訪問者が Web サイトを訪れるため、ダウンロード・コンポーネントが誤った画像を読み取る可能性がわずかですがあります。このようなエラーを回避するには、MD5 による計算とチェックが必要です。ただし、これは非常に大きな計算ワークロードであるため、Tencent* は MD5 の計算パフォーマンスを最大限に高める必要がありました。

元々、Tencent* は Operator* OS と md5sum* ユーティリティー・ツールを使用して、各画像ファイルの MD5 値を計算していました。インテルは Tencent* のエンジニアと協力して、インテル® インテグレーテッド・パフォーマンス・プリミティブ (**インテル® IPP**) を利用してパフォーマンスの最適化に取り組みました。その結果、Tencent* はインテル® アーキテクチャー・ベースのプラットフォームで 2 倍のパフォーマンス向上を達成しました。

インテル® ストリーミング SIMD 拡張とソフトウェアの最適化

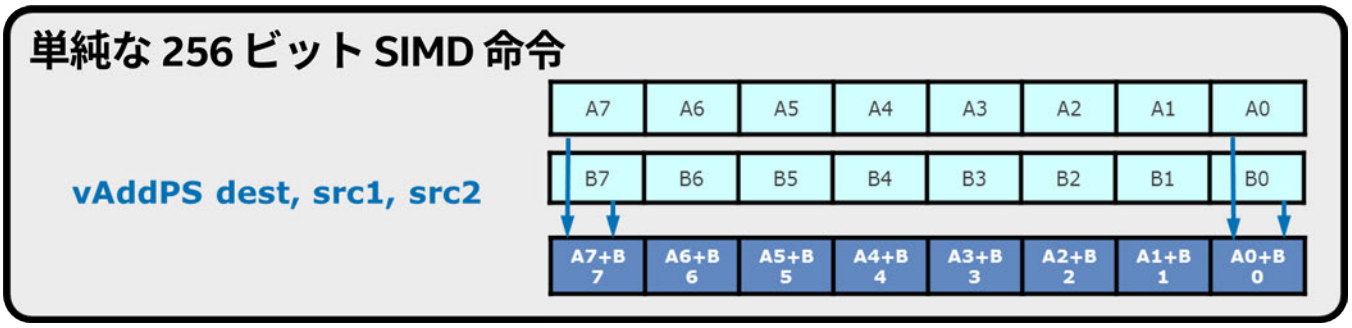
インテル® Pentium® III プロセッサーで、インテル® ストリーミング SIMD 拡張 (インテル® SSE) 命令セットが導入されました。これは、インテル® Pentium® プロセッサーで導入された MMX® 命令と呼ばれる以前の **SIMD** (Single Instruction Multiple Data) 命令セットを大幅に変更したものです。

その後、インテル® アーキテクチャーの進化とともにインテル® SSE 命令セットも進化し、ベクトル幅が広くなり、新しい拡張可能な構文と豊富な機能が追加されました。最新の SIMD 命令セットであるインテル® アドバンスト・ベクトル・エクステンション 2 (インテル® AVX2) は、第 4 世代インテル® Core™ プロセッサー以降で利用できます。

TFS システムで使用されているほとんどのインテル® Xeon® プロセッサーは、インテル® SSE2 をサポートしています。インテル® SSE2 は、インテル® SSE3、インテル® SSE4.x、およびインテル® AVX により補完されます。

インテル® AVX は、インテル® SSE の 256 ビット命令セット拡張で、計算負荷の高いアプリケーションでより優れたパフォーマンスを提供できるように設計されています。インテル® SSE ベースの SIMD 命令セットに浮動小数点演算と整数演算の新しい機能が追加されており、よりコンパクトな SIMD 命令セットが含まれます。

図 2 は、1 つの SIMD 操作で 8 つのデータ (32 ビットの整数型、浮動小数点型) 命令を処理できることを示しています。



2 8 つのデータ命令を処理する SIMD 操作

インテル® AVX は、浮動小数点と整数データのベクトル処理の幅を広げることで、パフォーマンスを向上します。その結果、画像とオーディオ/ビデオ処理、科学シミュレーション、金融解析、3D モデリング/解析のような広範なアプリケーションで、ハイパフォーマンスと効率良いデータ管理を実現します。

インテル® SSE により利点が得られるアルゴリズム

インテル® SSE により利点が得られるアルゴリズムには、[2] 単一の 32 ビット/64 ビット・ワードよりも大きなデータセットに対して論理演算や数値演算を行うアルゴリズムが含まれます。インテル® SSE はベクトル命令または SIMD アーキテクチャーを利用して、ビット単位の XOR (排他的論理和)、整数/浮動小数点の累乗算、単一クロックサイクルでの複数の 32 ビット/64 ビット・ワードのスケーリングなどの操作を完了します。スピードアップは、各算術演算子や論理演算子に適用される並列演算とベクトルサイズ (マルチワード・データ) によってもたらされます。

以下は、SIMD ベクトル命令により大きな利点が得られるアルゴリズムの例です。

- **画像処理とグラフィックス。**どちらも解像度 (単位面積当たりの画素数) とピクセル・エンコーディング (明るさと色を表す 1 画素当たりのビット数) がスケーリングし、処理フレームレートが向上します。
- **デジタル信号処理 (DSP)。**センサーや計器からデジタル化されたサンプルは、画像のような解像度とデータ取得レートになります。通常、1 次元のデジタル化されたデータの時系列は、大量の時系列サンプルを処理する DFT (離散フーリエ変換) などのアルゴリズムを使用して変換されます。
- **ダイジェスト、ハッシュ、エンコード。**単純なパリティー、巡回冗長検査 (CRC)、MD5、セキュア・ハッシュ・アルゴリズム (SHA)、ガロア演算、リードソロモン・エンコーディング、暗号ブロックチェーン (CBC) のようなセキュリティー、データ破損防止、データ損失防止に使用されるアルゴリズムはすべて、データブロック (通常、数キロバイト) に対して論理演算子または算術演算子を利用します。
- **データ変換とデータ圧縮。**ほとんどの場合、エンジニアリング・シミュレーションや科学計算には経時的なデータ変換がつきもので、変換されるデータグリッドが含まれます。例えば、物理熱力学、力学、流体力学、電界モデルでは、浮動小数点値のグリッドで有限要素の物理フィールドを表します。これらの有限要素グリッドは、物理プロセスをシミュレーションするため、経時的な数学的変換によって更新されます。

インテル® AVX 向けに最適化されたインテル® IPP

インテル® IPP は、あらゆる画像/信号処理、データ圧縮、暗号化のニーズに適したパフォーマンス・ビルディング・ブロックです。これらの関数はロイヤルティー・フリーで、インテル® SSE、インテル® AVX、インテル® AVX2、およびインテル® AVX-512 命令セットを使用して高度に最適化されているため、最適化コンパイラー単独で生成した場合よりも高速なアプリケーションを生成できます。[3]

表 1 にインテル® IPP の機能を要約します。

パフォーマンスと電力効率を最適化	インテルにより開発され、将来にわたって開発期間を短縮	広範なクロスプラットフォームと OS 機能
高度にチューニングされたルーチン	現在および以前のプロセッサー向けに最適化	高度に最適化された数千の信号、データ、メディア関数
インテル® SSE4、インテル® SSSE3、インテル® SSE、インテル® AVX、インテル® AVX2、インテル® AVX-512 命令セットを使用して高度に最適化	開発、デバッグ、保守期間を短縮	広範なドメインをサポート
最適化コンパイラー単独で生成したよりも高速	一度記述したコードで将来の最適化にも対応	インテル® Quark™ プロセッサー、インテル® Core™ プロセッサー、インテル® Xeon® プロセッサー、インテル® Xeon Phi™ コプロセッサー・プラットフォームをサポート

表 1. インテル® IPP の機能

AVX = アドバンスト・ベクトル・エクステンション命令
SSE = ストリーミング SIMD 拡張命令
SSSE = ストリーミング SIMD 拡張命令補足命令

インテル® IPP ライブラリーは、さまざまな SIMD 命令セット向けに最適化されています。さらに、インテル® IPP は、実行しているプロセッサーで利用可能な SIMD 命令を判定して、そのプロセッサーに最適な SIMD 命令を選択する自動「ディスパッチ」機能も提供します。

表 2 は、インテル® IPP で使用されるプロセッサー固有コードです。

プロセッサー固有ライブラリー		
IA-32 アーキテクチャー	**インテル® 64 アーキテクチャー**	**説明**
px	mx	インテル® ストリーミング SIMD 拡張 (インテル® SSE) 対応プロセッサー向けに最適化された汎用コード
w7		インテル® SSE2 対応プロセッサー向けに最適化されたコード
	m7	インテル® SSE3 対応プロセッサー向けに最適化されたコード
v8	u8	インテル® Atom™ プロセッサーを含む、インテル® ストリーミング SIMD 拡張命令 3 補足命令 (インテル® SSSE3) 対応プロセッサー向けに最適化されたコード
p8	y8	インテル® SSE4.1 対応プロセッサー向けに最適化されたコード
g9	e9	インテル® アドバンスト・ベクトル・エクステンション (インテル® AVX) とインテル® AES New Instructions (インテル® AES-NI) 対応プロセッサー向けに最適化されたコード
h9	l9	インテル® AVX2 対応プロセッサー向けに最適化されたコード
	n0	Knights Landing (開発コード名) 上のインテル® AVX2-512 (F、CD、ER、PF) 向けに最適化されたコード
	k0	インテル® Xeon® プロセッサー上のインテル® AVX2-512 (F、CD、BW、DQ、VL) 向けに最適化されたコード

表 2. プロセッサー固有コード

ディスパッチについては、「**インテル® IPP ライブラリーの CPU ディスパッチの理解**」(英語) を参照してください。インテル® AVX 向けに最適化されたインテル® IPP 関数については、「**インテル® AVX 向けに最適化されたインテル® IPP 関数**」(英語) を参照してください。

インテル® IPP の MD5

ハッシュ関数は、デジタル署名とともに暗号化に使用されます。また、データの整合性を確保するためにも使用されます。デジタル署名とともに使用する場合は、一般公開されている関数でメッセージをハッシュし、結果のハッシュ値に署名します。メッセージを受け取った側は、メッセージをハッシュし、ブロックサイズとハッシュ値を照合します。

ハッシュ関数は、「メッセージ・ダイジェスト」や「一方向暗号化関数」とも呼ばれます。データの整合性を確保する場合は、ハッシュ関数を使用して特定の入力に対応するハッシュ値を計算します。そして、必要に応じて、入力データが変更されないことをチェックします。また、利用可能な入力を使用して再度ハッシュ値を計算し、オリジナルのハッシュ値と比較します。インテル® IPP は、ストリーミング・メッセージで次のハッシュ・アルゴリズムを実装しています。

- MD5 [RFC 1321]
- SHA-1
- SHA-224
- SHA-256
- SHA-384
- SHA-512 [FIPS PUB 180-2]

これらのアルゴリズムは、エンタープライズ・アプリケーションで広範に利用されています。

MD5 について

MD5 は、一般に利用されている暗号化ハッシュ関数で、通常 32 桁の 16 進数をテキスト形式で表現した 128 ビット (16 バイト) のハッシュ値を生成します。

MD5 は、要件の厳しい環境では「暗号化に適していない」と見なされていましたが、ソフトウェアの世界では、転送したファイルが受信者に到達したことを確認する手段として広く利用されてきました。例えば、ファイルサーバーは、ユーザーがダウンロードしたファイルのチェックサムと比較できるように、ファイルの事前に計算された MD5 (md5sum) チェックサムを提供することがよくあります。ほとんどの Linux* ディストリビューションには、md5sum ユーティリティーが含まれています。

インテル® IPP の MD5 関数は、ハッシュ・アルゴリズムをストリーミング・メッセージのダイジェストに適用します。チェーン・ダイジェスト値の計算を管理するため、ステート・コンテキスト (ippsSHA1State など) を使用してすべての必要な変数を保持します。例えば、MD5 ハッシュ・アルゴリズムを実装するプリミティブは、ippsMD5State コンテキストを使用する必要があります。関数 Init (MD5Init) は、コンテキストを初期化し、指定された初期化ベクトルを設定します。初期化後、関数 Update (MD5Update) は、選択されたハッシュ・アルゴリズムを利用して、すべてのメッセージブロックがなくなるまで、入力メッセージストリームをダイジェストします。関数 Final (MD5Final) は、指定されたパディングスキームにより、不完全なメッセージブロックにパディングを追加して、最終メッセージブロックにします。そして、ハッシュ・アルゴリズムを利用して、最終ブロックをメッセージ・ダイジェスト値に変換します。

次の例は、アプリケーション・コードで MD5 関数を利用して、入力メッセージストリームをダイジェストする方法を示します。

1. 関数 MD5GetSize を呼び出し、ipps MD5State コンテキストの設定に必要なサイズを取得します。
2. 必要なメモリー空間を割り当てます。メモリーを割り当てたら、MD5Init 関数を呼び出し、MD5 で指定された初期化ベクトルを利用して、コンテキストの初期ステートを設定します。
3. キューにある入力メッセージストリームがなくなるまで、関数 MD5Update を呼び出してダイジェストします。現在のダイジェスト値を特定するため、2 つの MD5Update 呼び出しの間に MD5GetTag を呼び出します。
4. 関数 MD5Final を呼び出し、不完全なブロックにパディングを追加して最終 MD5-1 メッセージブロックにし、160 ビットのメッセージ・ダイジェスト値に変換します。
5. コンテキストに格納されている秘密データをクリーンアップします。
6. オペレーティング・システムのメモリー解放サービス関数を呼び出し、ippsMD5State コンテキストを解放します。

インテルのエンジニアは、主にベクトル化、インテル® SSE 命令、インテル® アーキテクチャーの機能 (キャッシュ効率やレジスター再利用など) によりインテル® IPP 関数を最適化しました。

インテル® IPP の MD5 実装では、次の最適化手法が利用されています。

- 小さなループの代わりに、完全にアンロールされたコードを使用
- メモリー操作の代わりに、レジスターによる巡回置換を使用
- 一般的なパラメーター化された 32 ビットの回転の代わりに、コードで直接回転

「インテルのエンジニアとの共同作業で、弊社のオンライン画像格納・処理アプリケーションの画像識別コンポーネントにインテル® IPP ライブラリーを採用しました。その結果、アプリケーションのパフォーマンスが大幅に向上し、運用コストも大幅に削減されました。インテルの協力に感謝しています。今後も、機会があれば共同作業を行っていきたいと思います。」

TFS ベースの画像格納・処理チームリーダー
Nicholas 氏

インテル® IPP コードで md5sum を置き換えました。**表 3** のコードを使用するだけで、手動による最適化は不要でした。

md5sum コード	インテル® IPP の MD5

`md5sum  performance.PNG > hash.md5`
`cat hash.md5`
`0e5e74555b68db366c85b1b194f258fe`
`performance.PNG.`
 	
`The md5sum is along with Cent OS`
`distribution.`
	`char* intel_md5sum(const char *p,`
	`unsigned uiLen)`
	`{`
	`static Ipp8u MD[32];`
	`int size;`
	`IppsMD5State *ctx;`
	`ippsMD5GetSize(&size);`
	`ctx = (IppsMD5State*)malloc(size);`
	`IppStatus st = ippsMD5Init(ctx);`
	`st = ippsMD5Update((const Ipp8u *)`
	`p, (int)uiLen, ctx);`
	`st = ippsMD5Final(MD, ctx);`
	`free(ctx);`
	`return (char*)MD;`
	`}`

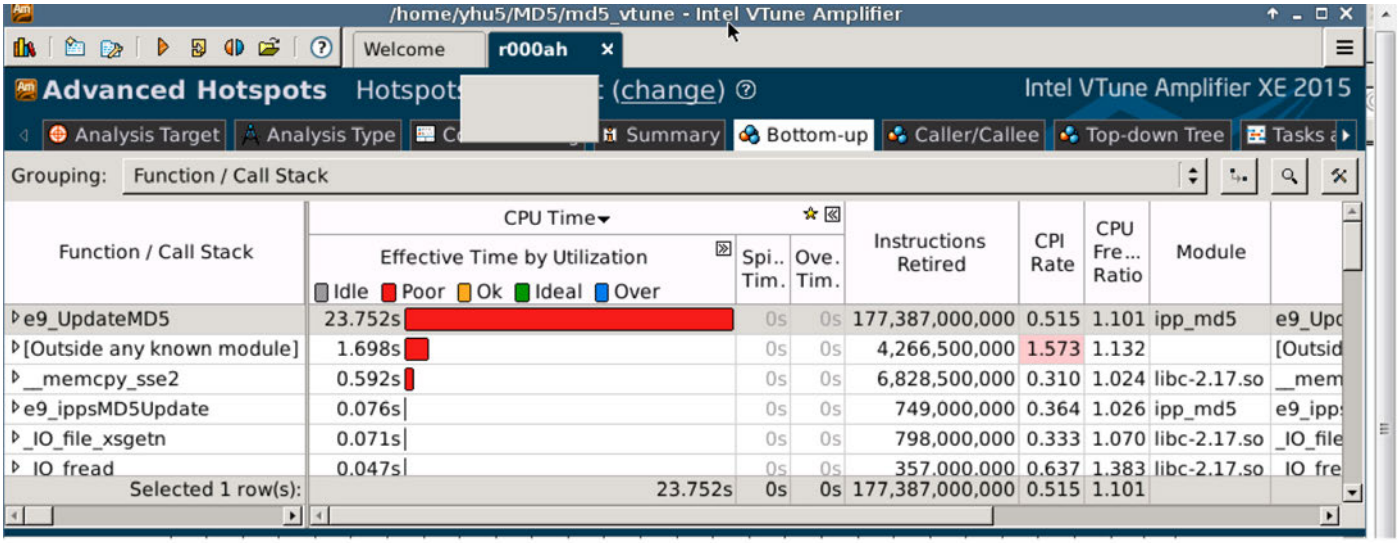
表 3. コード例

インテル® IPP による MD5 の高速化

インテル® IPP の MD5 コード **`md5test.cpp`** は、次のように gcc でコンパイルします。

```
[root@localhost code]# make -f Makefile.gcc
g++ -O2 ipp_md5.cpp -o ipp_md5
-I/opt/intel/compilers_and_libraries_2016.0.109/linux/ipp/include
/opt/intel/compilers_and_libraries_2016.0.109/linux/ipp/lib/intel64/libippcp.a
/opt/intel/compilers_and_libraries_2016.0.109/linux/ipp/lib/intel64/libippcore.a
```

これにより、インテル® IPP 暗号化ライブラリーがプログラムにリンクされ、自動的に計算リソースのパフォーマンスが引き出されます。**図 3** は、インテル® VTune™ Amplifier XE で ipp_md5 プログラムを実行した結果です。e9 (インテル® AVX 向けに最適化された) コードを実行するインテル® IPP 関数 e9_ippsMD5Update で、プログラムの CPU 時間のほとんどが費やされていることが分かります。



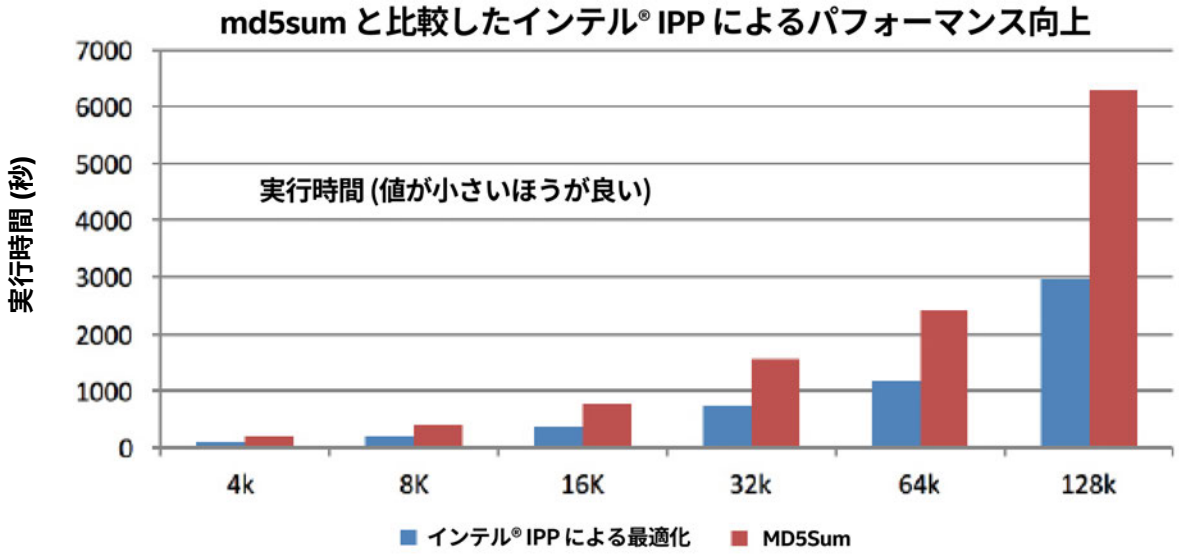
3 インテル® VTune™ Amplifier XE で ipp_md5 を実行した結果

パフォーマンス・データ

テストは、インテル® IPP と Linux* の md5sum を使用して、異なるサイズの画像ファイルで実行しました。10,000 反復のパフォーマンス結果が**表 4** と**図 4** です。

プロセッサー/OS	テストした画像サイズ	md5dum (10,000 反復の平均時間)	IPP_Md5 (10,000 反復の平均時間)
インテル® Xeon® プロセッサー E5-2620 (15M キャッシュ、2GHz、7.20GT/秒、インテル® QPI、インテル® AVX をサポートする CentOS* 6.5)	4K	206ms	95ms
	8K	406ms	189ms
	16K	789ms	369ms
	32K	1574ms	740ms
	64K	2420ms	1183ms
	128K	6273ms	2943ms

表 4. テスト実行のパフォーマンス結果



システム構成 - バージョン: インテル® IPP 9.0.0、ハードウェア: インテル® Xeon® プロセッサー E5-2620 (15M キャッシュ、2GHz、7.20GT/秒、インテル® QPI)、メモリー: 64G、オペレーティング・システム: CentOS* 8 6.5 x86_64、ベンチマークのソース: テスト用画像ファイル。
* 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。

最適化に関する注意事項: 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。

4 テスト実行の結果

インテル® Xeon® プロセッサー E5-2620 (15M キャッシュ、2GHz、7.20GT/秒、インテル® QuickPath インターコネクト [インテル® QPI]、インテル® AVX 対応) では、Linux* の md5sum と比較してパフォーマンスが 100% 向上しました。Tencent* エンジニアは、オンラインシステムにもインテル® IPP の MD5 を実装しました。その結果、オリジナルの MD5 と比較して、パフォーマンスが 60% 向上しました。

まとめ

Tencent* では、毎日、WeChat*、QQ*、QQ Album* でユーザーにより生成された数十億もの画像を処理しています。すべての画像は、TFS ベースの画像格納・処理システムによって処理されます。各イメージには、MD5 ハッシュにより一意の ID を付ける必要があります。インテルは Tencent* のエンジニアと協力して、インテル® IPP を利用することで、この関数コンポーネントを最適化し、2 倍のパフォーマンス向上を達成しました。

インテル® IPP を利用することで、画像の md5sum 計算速度を簡単に向上できました。この作業を通じて、インテル® IPP とパフォーマンス・チューニング手法を用いて最新のインテル® ハードウェア向けに最適化することで、これらの計算負荷の高い処理に対応できたことは大きな前進です。

参考文献

1. **Wikipedia* の md5sum の説明**
2. **インテル® ストリーミング SIMD 拡張命令とインテル® IPP によるアルゴリズムの高速化 (英語)**
3. **インテル® AVX 命令を使用した complex float データ型の IIR フィルターの実装**