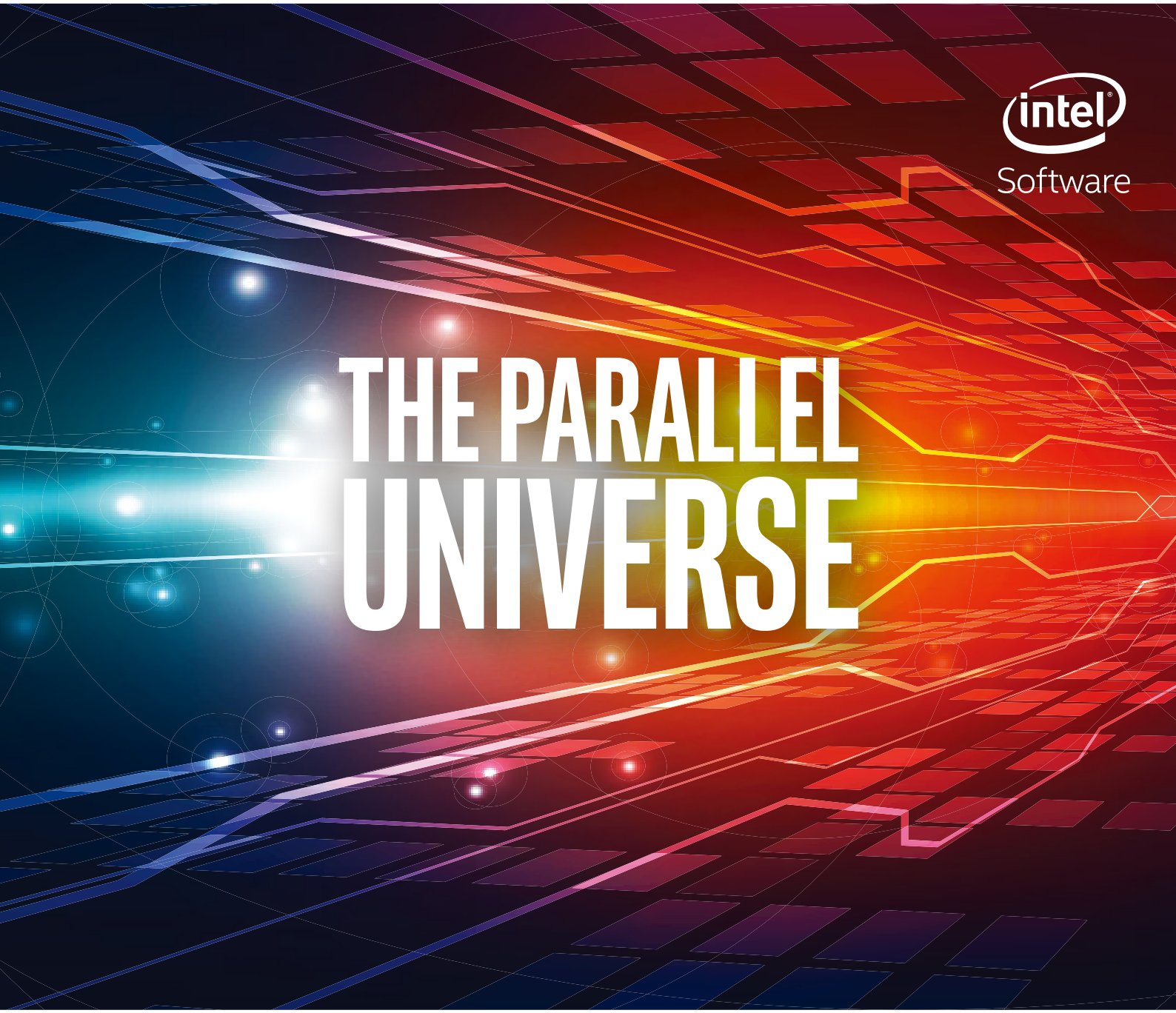


第 2 世代インテル® Xeon® スケーラブル・プロセッサーで最先端のパフォーマンスを実現

最新のパフォーマンス解析ツールを使用して
インテル® Optane™ DC パーシステント・メモリーに対応

NUMA マイグレーションのパフォーマンスへの影響を測定

Issue
37
2019

目次

編集者からのメッセージ 3
ブラックホールとハイパフォーマンス・コンピューティング
Henry A. Gabb インテル コーポレーション シニア主席エンジニア

注目記事

第 2 世代インテル® Xeon® スケーラブル・プロセッサーで最先端のパフォーマンスを実現 5
HPC、AI、分析アプリケーションのパフォーマンスを最大限に引き出す新しい機能とツール

最新のパフォーマンス解析ツールを使用してインテル® Optane™ DC パーシステント・メモリーに対応 19
ボトルネックとストレージの問題を解決

NUMA マイグレーションのパフォーマンスへの影響を測定 27
トレードオフを考慮してパフォーマンスを最大化

Python* の並列化 : numexpr によるベクトル化の指示 35
配列と数式の計算パフォーマンスを大幅に向上

インテル® Xeon® プロセッサーでインテル® AVX-512 を利用してオープン・シェーディング言語を加速 39
フル・レンダリングのデジタルコンテンツ制作を最大 2 倍高速化

パフォーマンスの最適化と生産性 (PoP) プロジェクト 53
終わりのないパフォーマンスの追求

インテル® ソフトウェアへの投資が HPC ソフトウェア開発者にもたらす 7 つのメリット 59
インテルおよび HPC ソフトウェアを見直す

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。

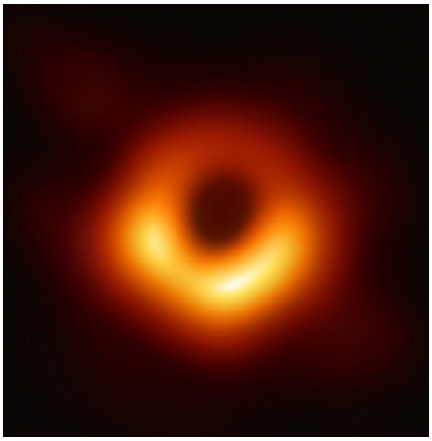
編集者からのメッセージ

Henry A. Gabb インテル コーポレーション シニア主席エンジニア
HPC と並列コンピューティング分野で長年の経験があり、並列プログラミングに関する記事を多数出版しています。『Developing Multithreaded Applications: A Platform Consistent Approach』の編集者 / 共著者で、インテルと Microsoft* による Universal Parallel Computing Research Centers のプログラム・マネージャーを務めました。

ブラックホールとハイパフォーマンス・コンピューティング

もう随分前のことのように思われますが、イベント・ホライゾン・テレスコープがブラックホールの画像を公開したのはわずか 3 ヶ月程前のことです。

これは明らかに素晴らしい科学的偉業でした。しかし、1 つの画像だけでは、この偉業を成し遂げた膨大な量の専門知識、データ、計算を伝えることはできません。ECHO を含む、この偉業の達成に使用されたいくつかのコードの詳細は、**イベントホライゾン一般相対論的電磁流体コードの比較プロジェクト** (英語) で提供されています。「**ECHO-3DHPC による天体物理シミュレーションのパフォーマンスを向上**」(2018 年に公開された **The Parallel Universe 34 号**の記事) では、ライプニッツ研究センター (LRZ) の研究者とインテルによるこのコードの最適化を紹介しています。



出典 : Event Horizon Telescope Collaboration

この号の注目記事「**第 2 世代インテル® Xeon® スケーラブル・プロセッサーで最先端のパフォーマンスを実現**」では、最新の**インテル® Xeon® プロセッサー・ファミリー**について説明します。この新しいプロセッサーには、**インテル® ディープラーニング・ブースト** (英語) と**インテル® Optane™ DC パーシステント・メモリー**のサポートが追加されており、CPU ソケットごとに最大 56 コア、ソケットごとに 12 の DDR4 メモリーチャネルを搭載しています。この記事を読めば、この新しいプロセッサーがパフォーマンスの新記録を達成した理由が分かります。「**最新のパフォーマンス解析ツールを使用してインテル® Optane™ DC パーシステント・メモリーに対応**」では、この新しいメモリー・テクノロジーによってアプリケーションが利点を得られるかどうか判断する方法、およびこのテクノロジーを使用するアプリケーションの解析方法を紹介します。

NUMA (Non-Uniform Memory Access) アーキテクチャーは、長年にわたって利用されています。メモリーアクセスを高速化するため、スレッドをデータの近くに配置すべきことはほとんどの人が知っていますが、スレッドが実際に実行されている場所やオペレーティング・システムがスレッドを移動しているかどうかに注意を払う人はほとんどいません。「**NUMA マイグレーションのパフォーマンスへの影響を測定**」は、NUMA システムにおけるスレッドの振る舞いを理解するのに役立ちます。

この号でも引き続き Python* コードの最適化と並列化に関する記事を取り上げています。「**Python* の並列化 : numexpr によるベクトル化の指示**」は、簡単なコード変更により、複雑な数式のパフォーマンスを劇的に向上する方法を示します。

「**インテル® Xeon® プロセッサーでインテル® AVX-512 を利用してオープン・シェーディング言語を加速**」では、オスカー受賞実績のあるオープン・シェーディング言語をベクトル化するインテルの取り組みを紹介します。オープン・シェーディング言語は、100 本を超える映画で採用されているデジタルコンテンツ制作の事実上のオープンソース標準です。

最後に、**Numerical Algorithms Group** (英語) の Mike Croucher 氏と、本誌の名誉編集長である James Reinders によるゲスト論説で締めくくります。Mike Croucher 氏は、ソフトウェア・パフォーマンス向上のため、欧州連合が資金援助する「**パフォーマンスの最適化と生産性 (PoP) プロジェクト**」について述べます。James Reinders は、「**インテル® ソフトウェアへの投資が HPC ソフトウェア開発者にもたらす 7 つのメリット**」で、インテル® ソフトウェアを利用して最小限の労力でパフォーマンスを最大限に引き出す方法を示します。これらの論説は、パフォーマンスへの道は何が利用可能かを知っているかどうかであることを示しています。

また、コードの現代化、ビジュアル・コンピューティング、データセンターとクラウド・コンピューティング、データサイエンス、システムと IoT 開発向けのインテルのソリューションの詳細は、**Tech.Decoded** (英語) を参照してください。

Henry A. Gabb
2019 年 7 月

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。

第 2 世代インテル® Xeon® スケーラブル・プロセッサーで最先端のパフォーマンスを実現

HPC、AI、分析アプリケーションのパフォーマンスを最大限に引き出す新しい機能とツール

Amarpal S. Kapoor インテル コーポレーション テクニカル・コンサルティング・エンジニア
Rama Kishan V. Malladi 同パフォーマンス・モデリング・エンジニア
Avinash Karani および Nitya Hariharan 同アプリケーション・エンジニア

サーバー向けのプロセッサーである、第 2 世代**インテル® Xeon® スケーラブル・プロセッサー** (開発コード名 Cascade Lake) が 2019 年 4 月に発表されました。この新しいプロセッサー・ファミリーは、すでに 95 のパフォーマンス世界新記録を達成し、確固たる地位を築いています[1]。新しい機能として、AI のディープラーニング推論を高速化するインテル® ディープラーニング・ブースト (インテル® DL ブースト) と**インテル® Optane™ DC** パーシステント・メモリーのサポートが追加されています。これらのプロセッサーは、CPU ソケットごとに最大 56 コア、ソケットごとに 12 の DDR4 メモリーチャネルを搭載しており、優れたパフォーマンスを発揮することから、高密度インフラストラクチャーのさまざまな HPC、AI、分析アプリケーションに最適です。

インテル® Xeon® スケーラブル・プロセッサーは、50 を超えるワークロード最適化ソリューションとさまざまなカスタム・プロセッサーを含む、広範な計算ニーズと需要に対応するように設計されています。8200 シリーズでは最大 28 コア (56 スレッド)、9200 シリーズでは最大 56 コア (112 スレッド) を利用できます。各プロセッサー・コアには、1MB の専用 L2 キャッシュとコア当たり 1.275MB (28 コアで 38.5MB) のノンインクルーシブ共有 L3 キャッシュがあります。各ソケットには、クロスダイ (マルチソケット) 通信用に 10.4GT/ 秒で動作する最大 3 つのインテル® ウルトラ・パス・インターコネクト (インテル® UPI) リンクがあります。プロセッサー・メモリー・インターフェイスは、2,933MT/ 秒で動作する最大 6 チャネル (8200 シリーズ) または 12 チャネル (9200 シリーズ) の DDR4 メモリーをサポートします。また、インテル® Optane™ DC パーシステント・メモリー・モジュールを使用して、ソケットごとに最大 4.5TB のメモリーをサポートします。DL アプリケーションのパフォーマンスを向上するため、インテル® Xeon® スケーラブル・プロセッサーには 512 ビットの VNNI (Vector Neural Network Instructions) が用意されており、各コアでサイクルあたり最大 16 DP/32 SP/128 INT8 の MAC (乗累算) 命令を処理できます。サイドチャネル攻撃に対応するため、インテル® Xeon® スケーラブル・プロセッサーは、ソフトウェアベースの手法と比較してオーバーヘッドの小さいハードウェアの緩和策を実装しています[2]。これらのプロセッサー機能は、複数の計算ドメインに適用されますが、ここではそのいくつかを説明します。

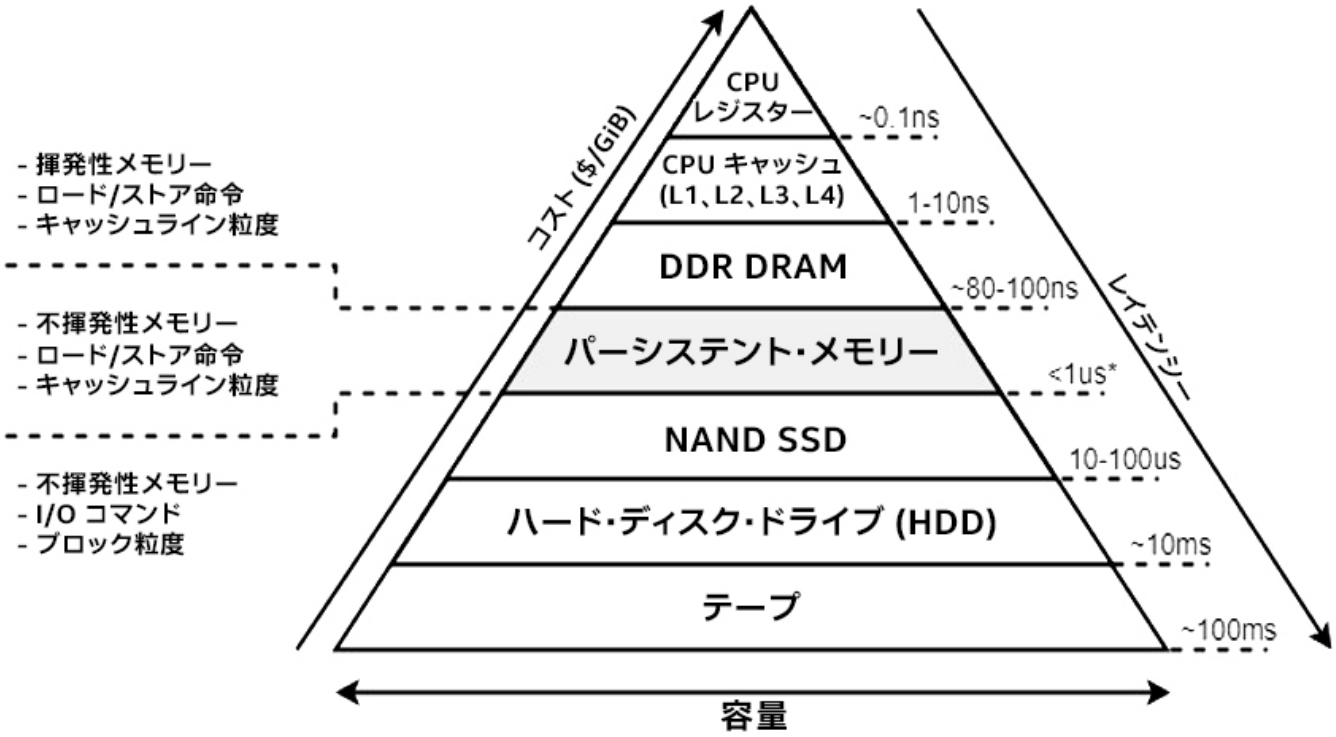
さらに、インテル® Optane™ DC パーシステント・メモリーの使用、DL 推論を高速化するインテル® アドバンスト・ベクトル・エクステンション 512 (インテル® AVX-512) VNNI、インテル® Xeon® スケーラブル・プロセッサー上での HPC アプリケーションの相対パフォーマンス・ゲインについても説明します。

[編集者注 : アプリケーションでこの新しいメモリーを最大限に利用する方法は、この号の「**最新のパフォーマンス解析ツールを使用してインテル® Optane™ DC パーシステント・メモリーに対応**」で紹介します。]

インテル® Optane™ DC パーシステント・メモリー

インテル® Optane™ DC パーシステント・メモリーは、手頃な価格で大容量のデータ永続性を提供する、DRAM に近いレイテンシーを備えた新しいタイプの不揮発性大容量メモリーです。**図 1** は、さまざまなメモリーとストレージデバイスの予測レイテンシーです。SSD と従来の DRAM の間に位置する新しい層がインテル® Optane™ DC パーシステント・メモリーです。

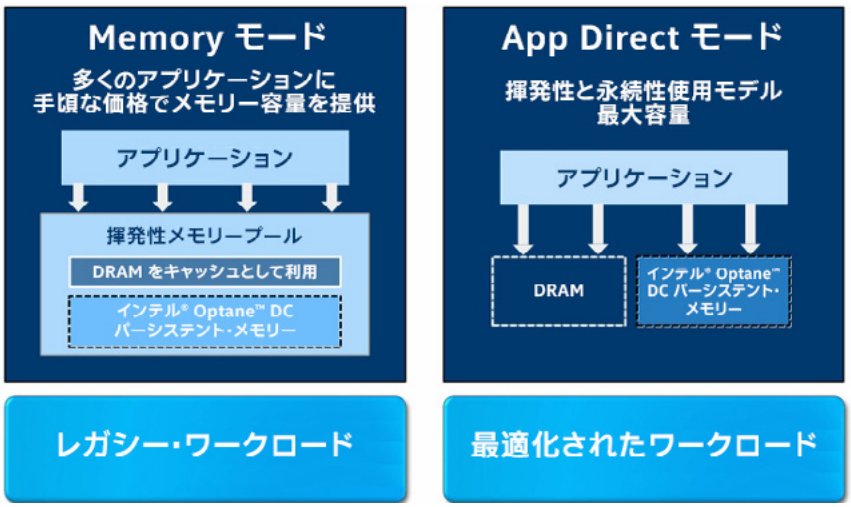
インテル® Optane™ DC パーシステント・メモリーは、DRAM と同じフォームファクターで、物理的および電気的に DDR4 のインターフェイスおよびスロットと互換性があります。インテル® Xeon® スケーラブル・プロセッサー・ベースのマシンは、DRAM とインテル® Optane™ DC パーシステント・メモリーの両方を搭載しています。(システム・アクティビティーの処理には DRAM が必要なため、インテル® Optane™ DC パーシステント・メモリーのみで構成することはできません。)



1 **さまざまなメモリーとストレージデバイスの予測レイテンシー**

インテル® Optane™ DC パーシステント・メモリーは、2 つのモードで使用できます (**図 2**)。

- Memory モード
- App Direct モード



2 **インテル® Optane™ DC パーシステント・メモリーの動作モード**

Memory モード

既存のアプリケーションのソースを変更しなくても利点が得られる、インテル® Optane™ DC パーシステント・メモリーを使用する最も単純なモードです。このモードでは、オペレーティング・システムとユーザー・アプリケーションが揮発性メモリーの新しいプールにアクセスできます。DRAM はホットな (頻繁にアクセスされる) データのキャッシュとして機能し、インテル® Optane™ DC パーシステント・メモリーは大容量の揮発性メモリーを提供します。メモリー管理は、インテル® Xeon® スケーラブル・プロセッサーのメモリー・コントローラーによって行われます。データがメモリーから要求されると、メモリー・コントローラーは最初に DRAM キャッシュをチェックします。データが見つかれば、応答レイテンシーは DRAM レイテンシーと同じです。データが DRAM キャッシュにない場合、インテル® Optane™ DC パーシステント・メモリーから読み取るため、レイテンシーは高くなります。メモリー・コントローラーの予測メカニズムは、要求されたデータを事前にフェッチすることでキャッシュヒット率を向上するのに役立ちます。しかし、広範なアドレス範囲へのランダムなアクセスパターンを持つワークロードでは、予測メカニズムの利点が得られず、レイテンシーが DRAM レイテンシーよりもわずかに高くなります[3]。

App Direct モード

データをメモリーに永続的に保持するには、インテル® Optane™ DC パーシステント・メモリーを App Direct モードで使用します。このモードでは、オペレーティング・システムとユーザー・アプリケーションは、DRAM とインテル® Optane™ DC パーシステント・メモリーの両方をディスクリート・メモリー・プールとして認識します。プログラマーは、どちらのメモリープールにもオブジェクトを割り当てることができます。最短レイテンシーでフェッチする必要があるデータは、DRAM に割り当てる必要があります (このデータは本質的に揮発性です)。DRAM に収まらない大きなデータや永続性が求められるデータは、インテル® Optane™ DC パーシステント・メモリーに割り当てる必要があります。これらの新しいメモリー割り当ての可能性により、App Direct モードではソースコードの変更が必要になります。App Direct モードでは、従来の HDD/NVMe ストレージデバイスに代わる高速ストレージとしてインテル® Optane™ DC パーシステント・メモリーを使用することもできます。

設定

Memory モードと App Direct モードを切り替えるには、BIOS 設定の変更が必要です。**`ipmctl`** は、インテル® Optane™ DC パーシステント・メモリー・モジュール (PMM) の設定と管理に利用できるオープンソースのユーティリティーです[4]。いくつかの管理コマンドを次に示します。

```
$ ipmctl show -topology
$ ipmctl show -memoryresources
```

PMM のプロビジョニングは 2 段階のプロセスです。最初に goal を指定して、PMM にストアします。そして、次回起動時に BIOS がそれを読み取ります。

```
Memory mode
$ ipmctl create -goal MemoryMode=100

App Direct mode
$ ipmctl create -goal PersistentMemoryType=AppDirect

Mixed mode
$ ipmctl create -goal MemoryMode=60
```

混在モードでは、指定された割合のインテル® Optane™ DC パーシステント・メモリーを Memory モードで使用し、残りを App Direct モードで使用します。上記のコマンドは、利用可能なパーシステント・メモリーの 60% を Memory モードで使用し、40% を App Direct モードで使用します。(詳細は「参考資料」の 4 と 5 を参照。)

パーシステント・メモリー開発キット (PMDK)

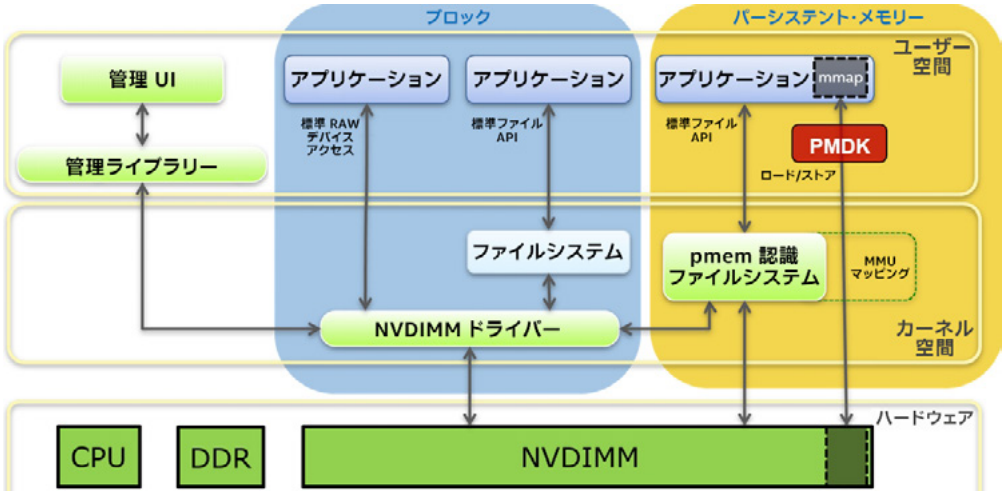
アプリケーションは、従来のメモリーと同様に、永続的なメモリー常駐のデータ構造にアクセスできるため、メモリーとストレージの間でデータのページブロックをやり取りする必要がなくなります。この低レイテンシーの直接アクセスを実現するには、アプリケーションがパーシステント・メモリー領域にアクセスできるようにする新しいソフトウェア・アーキテクチャーが必要です[6]。**図 3** に示す SNIA (Storage Network Industry Association) プログラミング・モデルを利用できます。

PMDK は、システム管理者とアプリケーション開発者がパーシステント・メモリー・デバイスの管理とアクセスを容易に行えるようにするライブラリーとツールのコレクションです。これらのライブラリーを使用することで、メモリーマップ・ファイルとしてパーシステント・メモリーにアクセスできます。**図 3** の SNIA モデルは、データがアプリケーションにメモリーマップされている場合、アプリケーションが **`read`**、**`write`**、**`pread`**、**`pwrite`** などの従来の POSIX* 標準 API や **`memcpy`** などのロード / ストア操作を使用してパーシステント・メモリー・デバイスにアクセスできることを示しています。アプリケーション I/O は既存のファイルシステムのページキャッシュをバイパスし、パーシステント・メモリー・メディアに直接アクセスするため、パーシステント・メモリー領域は最も高速にアクセスできます[6]。

PMDK には、パーシステント・メモリー・システムのプログラミング要件に対応するため、次のライブラリーとユーティリティーが含まれています。

PMDK libraries
Libpmem
Libpmemobj
Libpmemblk
Libpmemlog
Libvmem
Libvmmalloc
Libpmempool
Librmem
Libvmemcache

PMDK libraries
pmempool
pmemcheck



3 SNIA プログラミング・モデル[6]

PMDK の使用例

このセクションでは、トランザクション・オブジェクト・ストアを提供する **`libpmemobj`** ライブラリーを使用して、メモリー割り当て、トランザクション、およびパーシステント・メモリー・プログラミング機能を含む PMDK の使用例を示します。この例では、次の 2 つのアプリケーションを使用します。

- **`writer.c`:** パーシステント・メモリーに文字列を書き込みます。
- **`reader.c`:** パーシステント・メモリーから文字列を読み取ります。

表 1 にコードの抜粋とコメントを示します。完全なソースコードは、「参考資料」の 7 を参照してください。

表 1. パーシステント・メモリーでの文字列操作

writer.c	reader.c

`int main(int argc, char *argv[1]`
`{`
`PMEMobjpool *pop = pmemobj_create(argv[1],`
`LAYOUT_NAME,`
`PMEMOBJ_MIN_POOL, 0666),`
`if (pop == NULL) {`
`Perror("pmemobj_create");`
`return 1;`
`}`
`...`
`...`
`pmemobj_close(pop);`
`return 0;`
`}`
	`int main(int argc, char *argv[1]`
	`{`
	`PMEMobjpool *pop = pmemobj_open(argv[1],`
	`LAYOUT_NAME);`
	`if (pop == NULL)`
	`{`
	`perror("pmemobj_open");`
	`return 1;`
	`}`
	`...`
	`...`
	`pmemobj_close(pop);`
	`return 0;`
	`}`

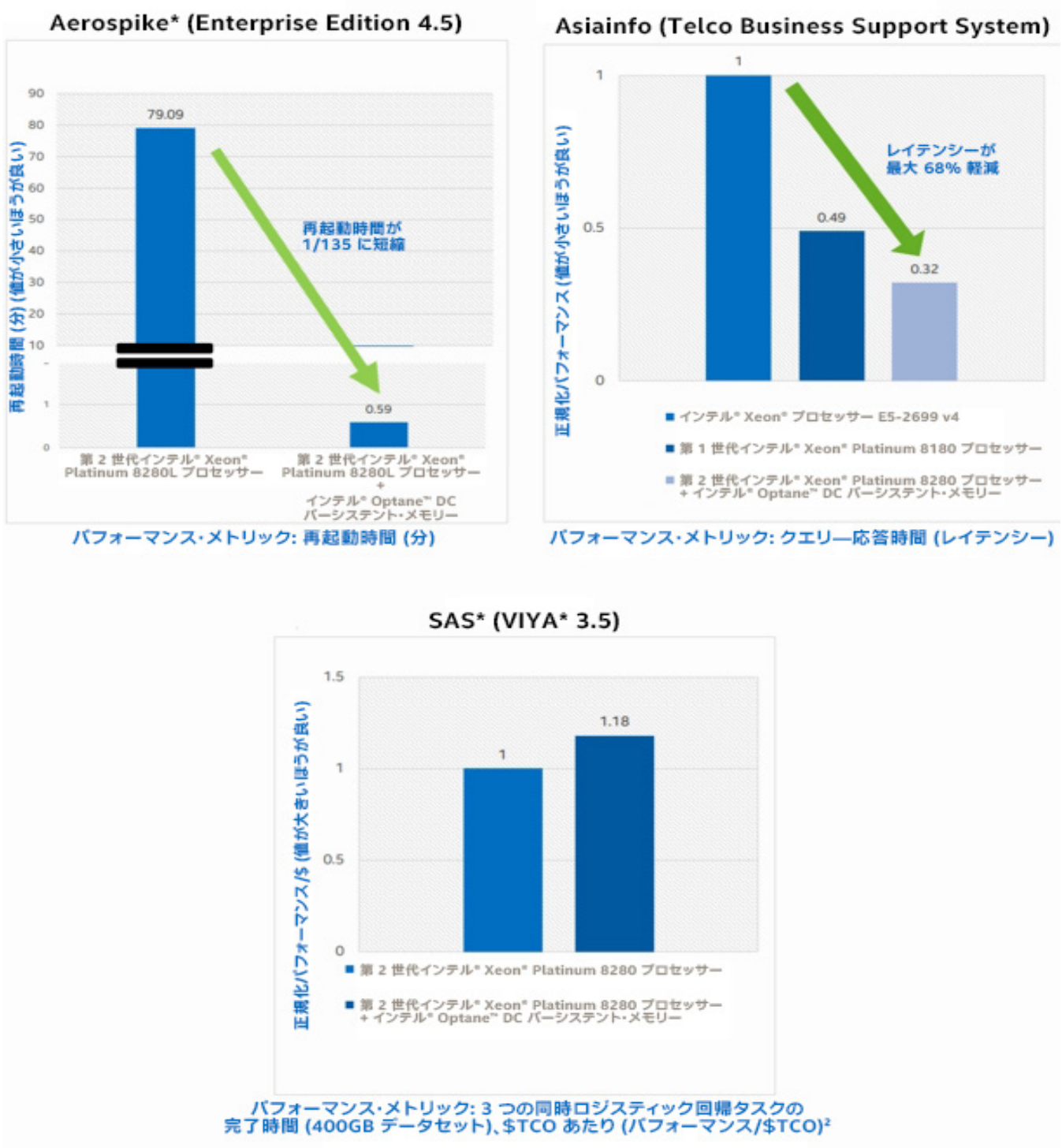
pmemobj_create API は、ファイル作成関数で想定される通常のパラメーターと、プールを識別する任意の文字列であるレイアウトを受け取ります。	**reader.c** では、新しいプールを作成する代わりに **writer.c** で作成したプールを同じレイアウトを使用して開きます。
`PMEMoid root = pmemobj_root(pop, sizeof (struct my_root));` `struct my_root *rootp = pmemobj_direct(root);`	`PMEMoid root = pmemobj_root(pop, sizeof (structr my_root));` `struct my_root *rootp = pmemobj_direct(root);`
ルート・オブジェクトと呼ばれる、アプリケーションの既知の場所をメモリープールに保持する必要があります。これがアンカーとなり、すべてのメモリー構造が関連付けられます。上記のコードでは、**pmemobj_root** を使用して **pop** メモリープールでルート・オブジェクトを作成しています。また、**pmemobj_direct** を使用してルート・オブジェクトを利用可能な直接ポインターに変換しています。	プール内にルート・オブジェクトがすでに作成されているため、**pmemobj_root** はルート・オブジェクトをゼロで初期化せずにそのまま返します。ルート・オブジェクトには、writer.c でストアされた文字列が含まれています。
`char buf[MAX_BUF_LEN] = {0};` `if (scanf("%9s", buf) == EOF) {`	`if (rootp->len == strlen(rootp->buf))` `printf("%s\n", rootp->buf);`
最大 9 バイトが一時バッファーに読み取られます。	上記のコードはパーシステント・メモリーから文字列を読み取ります。

`root->len = strlen(buf);`
`pmemobj_persist(pop, &rootp->len, sizeof(rootp->len));`
`pmemobj_memcpy_persist(pop, rootp->buf, buf, rootp->len);`	

上記のコードは、**pmemobj_persist** を使用して **(&rootp->len &rootp->len+sizeof(rootp->len))** 範囲内の変更をパーシステント・メモリーに永続的にストアするように強制し、**pmemobj_memcpy_persist** を使用してローカルバッファーからパーシステント・メモリーへ文字列をコピーします。	

インテル® Optane™ DC パーシステント・メモリーによるパフォーマンス・ゲイン

このセクションでは、インテル® Optane™ DC パーシステント・メモリーを使用して、Aerospike*、Asiainfo ベンチマーク、SAS* VIYA* などのエンタープライズ・アプリケーションで達成されたパフォーマンス・ゲインを紹介します (**図 4**)。

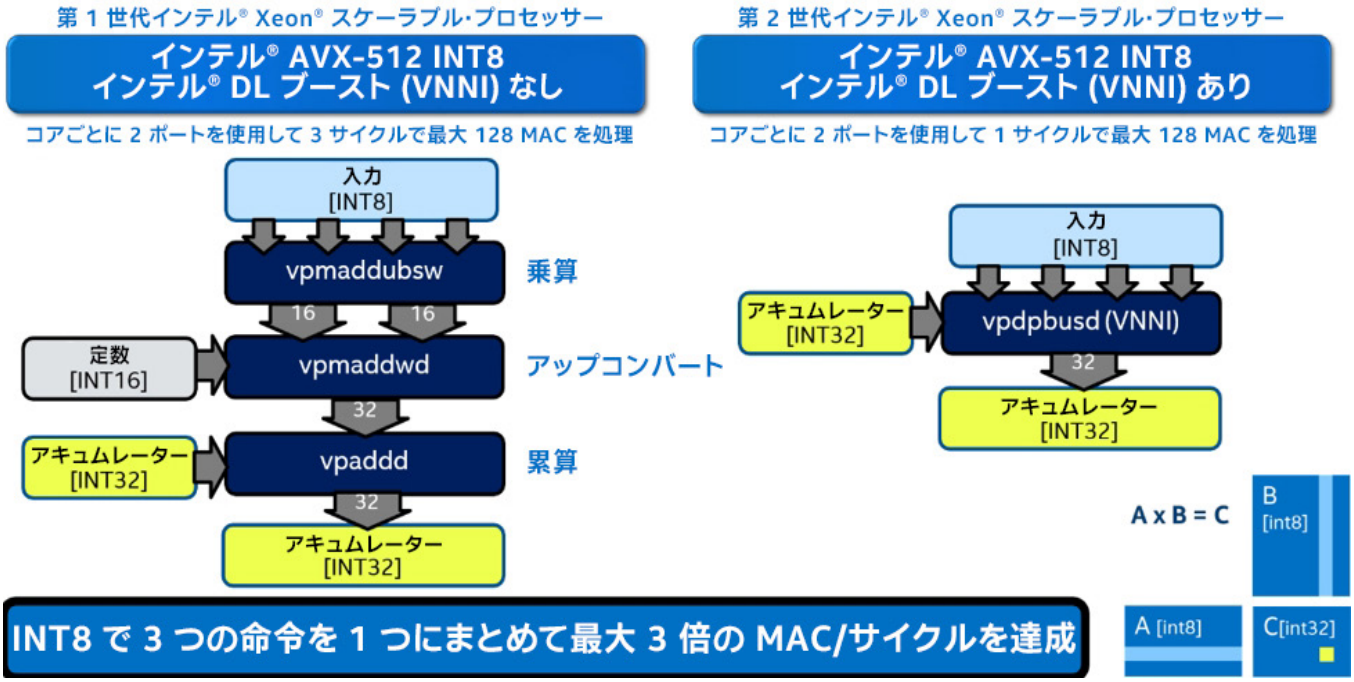


4 **インテル® Optane™ DC パーシステント・メモリーを使用することでエンタープライズ・アプリケーションのパフォーマンスが向上**

Aerospike* は、NoSQL* Key-Value データベース・アプリケーションです。インテル® Optane™ DC パーシステント・メモリーを App Direct モードで使用することで、再起動時間が 1/135 に短縮されました。再起動が数時間から数秒になったことで、中断を減らし、ソフトウェア・アップデートとセキュリティー・アップデートをより頻繁に行えるようになりました。Asiainfo ベンチマークの 1 つは、インテル® Xeon® スケーラブル・プロセッサーとインテル® Optane™ DC パーシステント・メモリーを App Direct モードで使用することで、レイテンシーが 68% 軽減されました。より多くのデータをメモリーに格納できるようになり、低速な SSD へのスピルオーバーが減ったことで、パフォーマンスが向上しました。SAS* VIYA* は、クラウド上にデプロイされた AI 機能を利用するオープンな統合分析プラットフォームです。インテル® Optane™ DC パーシステント・メモリーを Memory モードで使用することで、勾配ブースティング・モデルに必要な大規模データセットを、ほとんどまたは全くパフォーマンスを低下させることなく、低コストでメモリーに配置することができました。これにより、18% のパフォーマンス・ゲインを実現しました。

インテル® AVX-512 VNNI による AI 推論の高速化

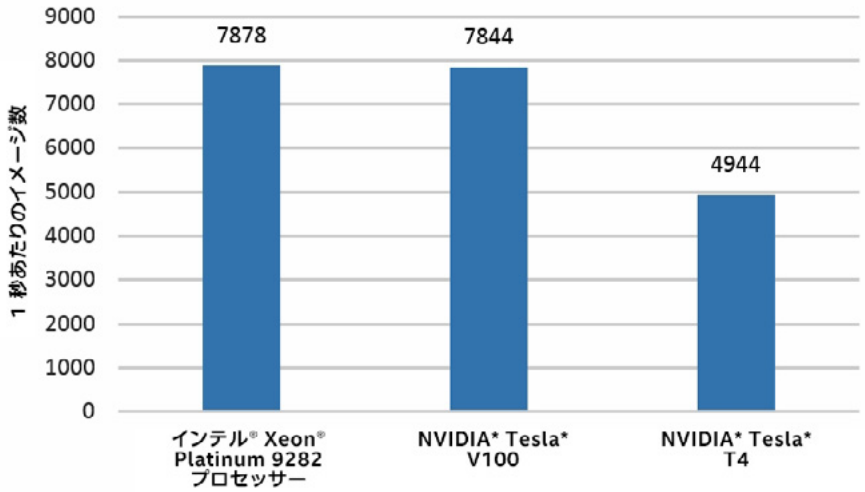
ニューラル・ネットワークはいくつかの行列操作を必要とし、それらは MAC 命令を使用して実現できます。第 1 世代インテル® Xeon® スケーラブル・プロセッサーでは、2 つの 8 ビット (INT8) 値を乗算し、その結果を 32 ビット値に集積するには 3 つの命令が必要でした。第 2 世代インテル® Xeon® スケーラブル・プロセッサーでは、この操作を 1 つの命令で行うことができます[8]。この命令数の減少はパフォーマンス・ゲインを示しており、これは実行パイプラインのポート 0 と 5 で MAC 命令を同時実行することで達成されます (**図 5**)。



5 インテル® DL ブースト・テクノロジー

現在、インテル® コンパイラーは、組込み関数とインライン・アセンブリーを介してのみ VNNI 命令をサポートしています。組込み関数やアセンブリーを使用せずに VNNI 機能を利用する場合、**ディープ・ニューラル・ネットワーク向けインテル® マス・カーネル・ライブラリー (インテル® MKL-DNN)**[9] (英語) と **BigDL**[10] (英語) の使用を推奨します。インテル® MKL-DNN は、従来の HPC 環境向けに高度に最適化された DL プリミティブのコレクションです。一方、BigDL (インテル® MKL-DNN を利用) は、Apache Spark* でビッグデータ・ユーザー向けに同様の最適化された DL 機能を提供します。

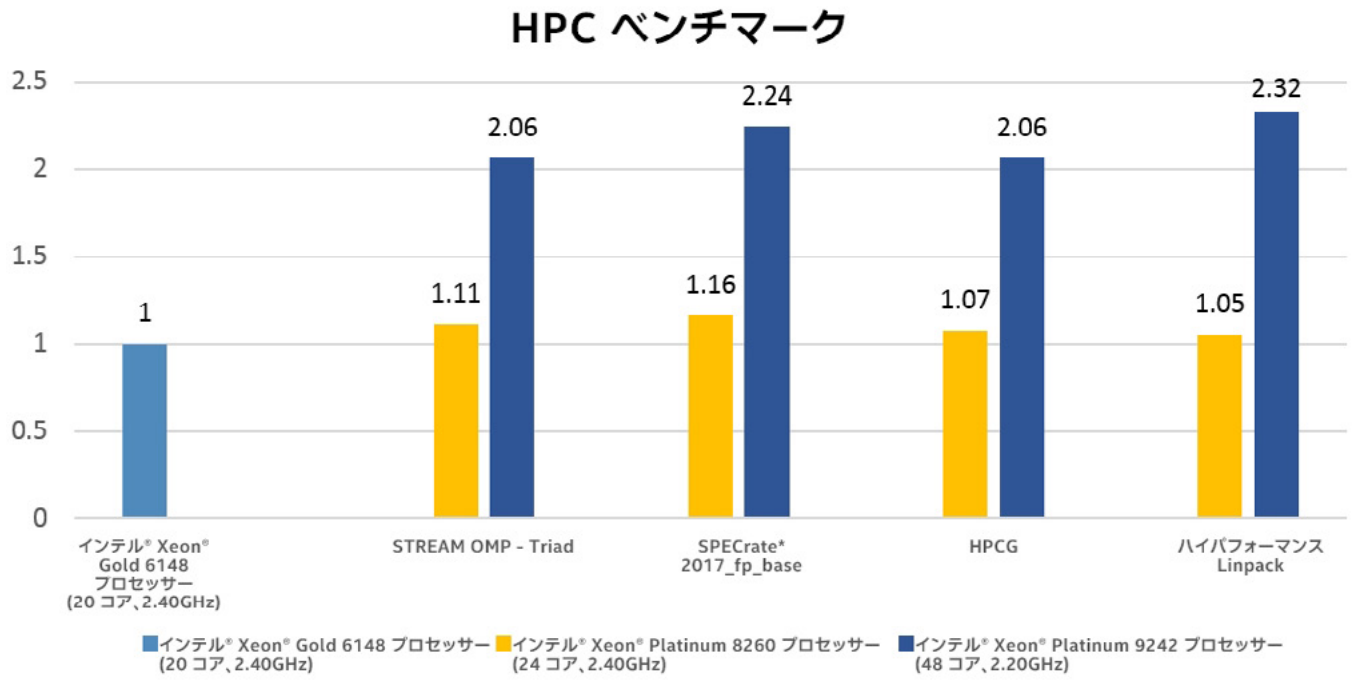
インテル® MKL-DNN で最適化された Caffe 1.1.3 は、第 1 世代インテル® Xeon® スケーラブル・プロセッサーと比較して、デュアルソケット インテル® Xeon® Platinum 8280 プロセッサー上では 14 倍の推論スループットを達成し、デュアルソケット インテル® Xeon® Platinum 9282 プロセッサー上では 30 倍の推論スループットを達成します[11]。インテル® Xeon® Platinum 9282 プロセッサー上で Caffe ResNet-50 を使用した同様の調査では、NVIDIA* GPU よりも優れた推論スループットが実証されています (**図 6**)[12]。Chainer*、DeepBench、PaddlePaddle*、PyTorch* など、ほかの一般的なフレームワークもパフォーマンス向上のためインテル® MKL-DNN を使用しています。



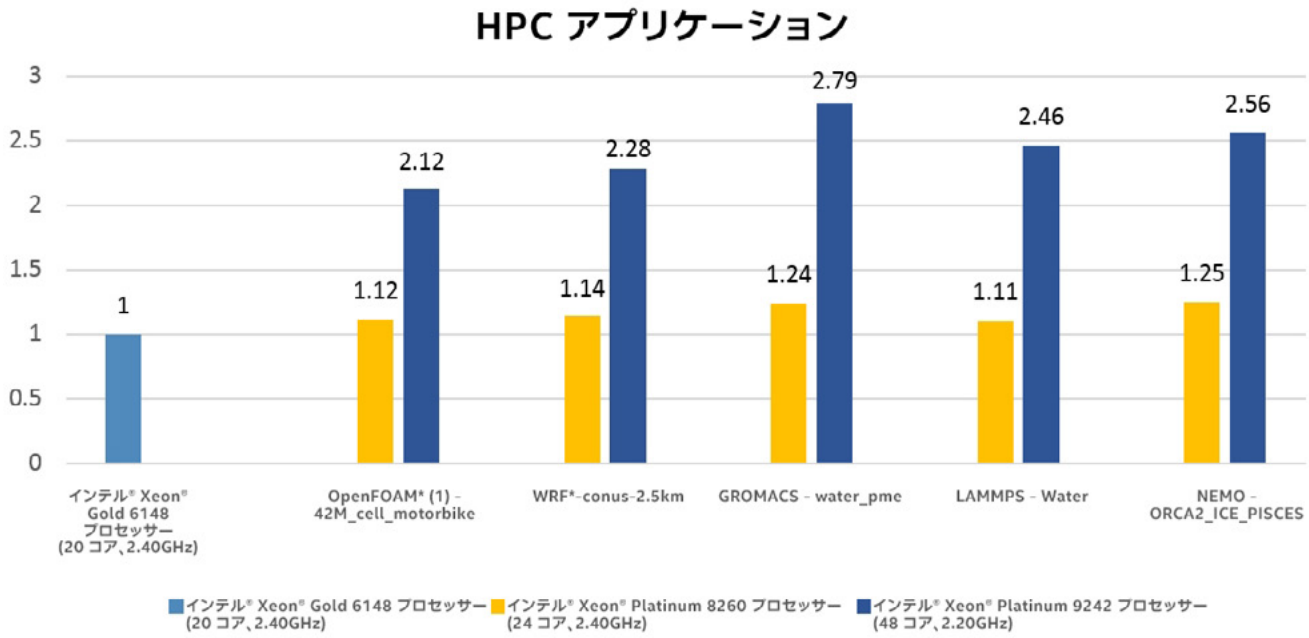
6 **Caffe ResNet-50 の CPU-GPU 推論スループットの比較 (値が大きいほうが良い)**

インテル® Xeon® スケーラブル・プロセッサー上での HPC アプリケーション・パフォーマンス

第 2 世代インテル® Xeon® Platinum プロセッサーでは、コア数と帯域幅の増加により、HPC アプリケーションのパフォーマンスが大幅に向上します。**図 7** は業界標準ベンチマーク、**図 8** は実際のアプリケーションでのパフォーマンス・ゲインをそれぞれ示しています。LAMMPS* と GROMACS* は、インテル® AVX-512、コア数の増加、ハイパースレッディングの恩恵を受けています。OpenFOAM*、WRF*、NEMO など、メモリー帯域幅依存のコードは、第 2 世代インテル® Xeon® スケーラブル・プロセッサーの高帯域幅によりパフォーマンスが向上しています。



7 **HPC 業界標準ベンチマークの相対パフォーマンス**



8 **HPC アプリケーションの相対パフォーマンス**

OpenFOAM* に関する法務上の注意書き : この結果は、OpenFOAM* ソフトウェアの開発・配布元である OpenCFD Limited (**www.openfoam.com** (英語)) により承認または是認されたものではありません。OPENFOAM および OpenCFD は、OpenCFD Limited の登録商標です。

HPC、AI、分析アプリケーションのパフォーマンスを向上

最新のインテル® Xeon® スケーラブル・プロセッサーの新しいハードウェア機能は、さまざまな HPC、AI、および分析アプリケーションにパフォーマンスの向上をもたらします。インテルはプロセッサー・テクノロジーの革新を続けており、現在開発中の開発コード名 Cooper Lake アーキテクチャーでは、AI 訓練サポートを強化する bfloat[16] が追加される予定です。また、最近リリースした 10nm テクノロジー・ベースの第 10 世代インテル® Core™ プロセッサーでは、パフォーマンスと密度が向上しています。

参考資料

1. **データ中心の現状に関する報道発表 (英語)**
2. **ハードウェアの脆弱性に対する対応 (英語)**
3. **インテル® Optane™ DC パーシステント・メモリーの動作モードに関する説明 (英語)**
4. **インテル® Optane™ DC パーシステント・メモリー用の ipmctl (英語)**
5. **Linux* 上でのインテル® Optane™ DC パーシステント・メモリーの設定 (英語)**
6. **パーシステント・メモリー開発キット (PMDK) (英語)**
7. **PMDK GitHub* (英語)**
8. **インテル® アーキテクチャーの命令セット (英語)**
9. **ディープ・ニューラル・ネットワーク向けインテル® マス・カーネル・ライブラリー (インテル® MKL-DNN) (英語)**
10. **BigDL: Apache Spark* 向け分散ディープラーニング・ライブラリー (英語)**
11. **第 2 世代インテル® Xeon® スケーラブル・プロセッサー : ソリューション (英語)**
12. **インテル® CPU が ResNet-50 ディープラーニング推論で NVIDIA* GPU を凌駕**
13. **インテル® Parallel Studio XE ツールスイート**

NEWS HIGHLIGHTS

インテルの「oneAPI」プロジェクトは多様なアーキテクチャーに統合プログラミング・モデルを提供

INTEL CORPORATION

ロンドンで開催されたインテルの Software Technology Day において、インテルのエンジニアリング・リーダーは、さまざまなコンピューティング・アーキテクチャーにわたるアプリケーション開発を容易にする統合プログラミング・モデルを提供する、インテルのソフトウェア・プロジェクト「oneAPI」の最新情報を発表しました。

「oneAPI」は、多様なアーキテクチャーにわたるワークロードの開発を簡素化できるように統合プログラミング・モデルを提供する開発ツールのセットです。インテルのコンピュート・パフォーマンス開発製品グループ本部長の Bill Savage は次のように述べています。

「コンピューティングの幅が広がり、専用アクセラレーターが含まれるようになりました。インテルは、開発者がそれらのハードウェアのパフォーマンスを最大限に引き出すことができるソフトウェア・ソリューションを提供します。」

この記事の続きはこちらでご覧になれます。>

付録

システム構成 : シングルノード インテル® Xeon® スケーラブル・プロセッサー上での HPC パフォーマンス

インテル® Xeon® Gold 6148 プロセッサー : インテルのリファレンス・プラットフォーム : 2S インテル® Xeon® Gold 6148 プロセッサー (2.40GHz、20 コア)、12x16GB DDR4-2666、1 SSD、クラスター・ファイル・システム : Panasas* (124TB ストレージ) ファームウェア v6.3.3.a & OPA ベースの IEEL Lustre*、BIOS: SE5C620.86B.00.01.0015.110720180833、マイクロコード : 0x200004d、セキュリティー修正のため ksplice を使用した 7.5 カーネル上の Oracle* Linux* Server 7.6 (RHEL 7.6 互換)、カーネル : 3.10.0-862.14.4.el7.crt1.x86_64、OFED スタック : RH7.5 上の Lustre* v2.10.4 を使用する OFED OPA 10.8。

インテル® Xeon® Platinum 8260 プロセッサー : インテルのリファレンス・プラットフォーム : 2S インテル® Xeon® Platinum 8260 プロセッサー (2.40GHz、24 コア)、12x16GB DDR4-2933、1 SSD、クラスター・ファイル・システム : Panasas* (124TB ストレージ) ファームウェア v6.3.3.a & OPA ベースの IEEL Lustre*、BIOS: SE5C620.86B.0D.01.0286.011120190816、マイクロコード : 0x4000013、セキュリティー修正のため ksplice を使用した 7.5 カーネル上の Oracle* Linux* Server 7.6 (RHEL 7.6 互換)、カーネル : 3.10.0-957.5.1.el7.crt1.x86_64、OFED スタック : Oracle* Linux* 7.6 (RHEL 7.6 互換) 上の Lustre* v2.10.6 を使用する OFED OPA 10.9。

インテル® Xeon® Platinum 9242 プロセッサー : インテルのリファレンス・プラットフォーム : 2S インテル® Xeon® Platinum 9242 プロセッサー (2.20GHz、48 コア)、24x16GB DDR4-2933、1 SSD、クラスター・ファイル・システム : 2.12.0-1 (サーバー) 2.11.0-14.1 (クライアント)、BIOS: PLYXCRB1.86B.0572.D02.1901180818、マイクロコード : 0x4000017、CentOS* 7.6、カーネル : 3.10.0-957.5.1.el7.x86_64。

TOOL HIGHLIGHTS

インテル® Advisor パフォーマンス解析クックブック
INTEL CORPORATION

インテル® Advisor を使用して、ループ交換とキャッシュ・ブロッキングによりメモリー・アクセス・パターンを最適化する方法を紹介します。

これは、メモリーのボトルネックを特定してパフォーマンスを向上するのに役立ちます。

次のワークフローに従って、ステップごとに説明します。

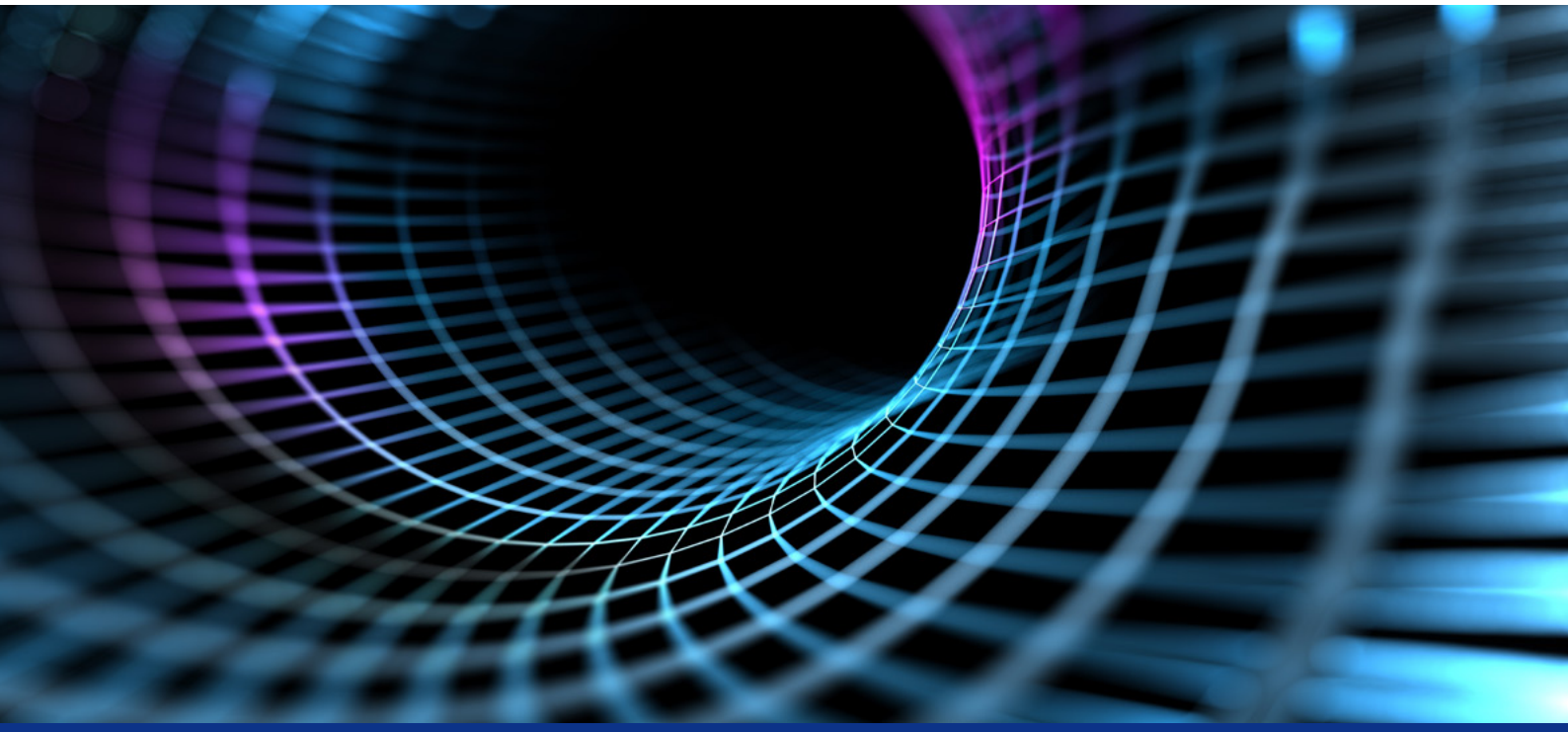
1. ベースラインの確立
2. ループ交換の実行
3. メモリー階層のレベルごとにメモリートレースを検証
4. キャッシュ・ブロッキング手法の実装

この記事の続きはこちらでご覧になれます。 >

インテル® Xeon® Gold 6148 プロセッサーおよび インテル® Xeon® Platinum 8260/9242 プロセッサー

アプリケーション	ワークロード	インテル® コンパイラー	インテル® ソフトウェア	BIOS 設定
STREAM OMP 5.10	Triad	2019u2		HT 有効、 ターボ無効、 コアごとに 1 スレッド
HPCG 2018u3	インテル® MKL を含むバイナリー	2019u1	MPI 2019u1、 MKL 2019u1	HT 有効、 ターボ無効、 コアごとに 1 スレッド
SPECrate*2017_fp_base	2019 年 6 月 20 日現在の公開されている最良の結果 : • **6148 (英語)** • **8260 (英語)** • **9242 (英語)**			
HPL 2.1	インテル® MKL を含むバイナリー	2019u1	MKL 2019、 MPI 2019u1	HT 有効、 ターボ無効、 コアごとに 2 スレッド
WRF 3.9.1.1	conus-2.5km	2018u3	MPI 2018u3	HT 有効、ターボ有効、 コアごとに , 1 スレッド
GROMACS 2018.2	すべてのワークロード	2019u2	MKL 2019u2、 MPI 2019u2	HT 有効、 ターボ無効、 コアごとに 2 スレッド
LAMMPS 12 Dec 2018	すべてのワークロード	2019u2	MPI 2019u2	HT 有効、 ターボ有効、 コアごとに 2 スレッド
OpenFOAM* 6.0	42M_cell_motorbike	2019u1	MPI 2018u3	HT 有効、 ターボ無効、 コアごとに 1 スレッド
NEMO v4.0	ORCA2_ICE_PISCES	2018u3	MPI 2018u3	HT 無効、 ターボ有効、 純粋な MPI 実行

インテル® MKL
ディープ・ニューラル・ネットワーク向け
無料
ダウンロード
(英語)



最新のパフォーマンス解析ツールを使用してインテル® Optane™ DC パーシステント・メモリーに対応

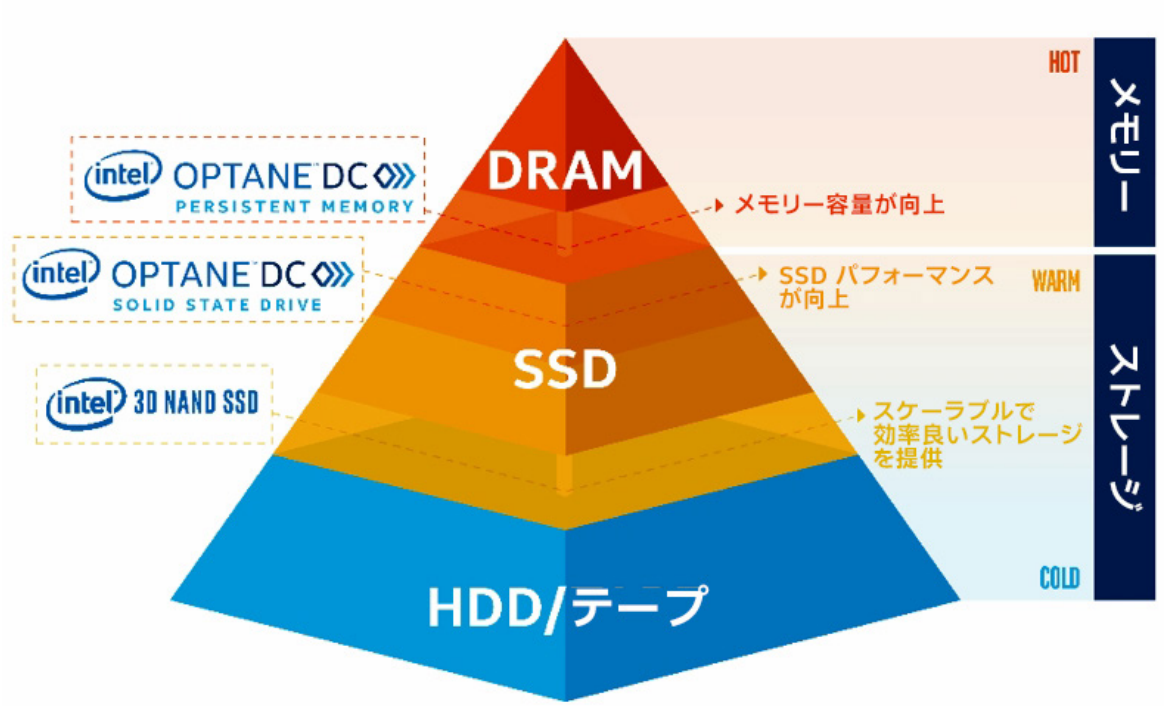
ボトルネックとストレージの問題を解決

Jackson Marusarz インテル コーポレーション テクニカル・コンサルティング・エンジニア
Kevin O' Leary インテル コーポレーション シニア・テクニカル・コンサルティング・エンジニア

良い知らせと悪い知らせがあります。最初に悪い知らせからお伝えしましょう。年々急増するデータとデータ分析や人工知能 (AI) などの分野の進歩により、多くのアプリケーションではプラットフォーム上で利用可能なシステムメモリーや高速ストレージがボトルネックになっています。良い知らせは、**インテル® Optane™ DC パーシステント・メモリー**がリリースされたことです。

この新しいテクノロジーは、SSD やハードドライブよりも高速で DRAM に近いレイテンシーの大容量不揮発性メモリー / ストレージを提供します。これは、現在メモリー容量や低速なストレージデバイスによって制限されているすべてのワークロードに影響します。

図 1 は、現在のプラットフォームのメモリー階層におけるインテル® Optane™ DC パーシステント・メモリーの位置付けを示します。この記事は、インテル® ツールを使用して既存のワークロードをプロファイルし、この新しいハードウェアの利点が得られるかどうか評価する方法を理解するのに役立ちます。



1 **新しいメモリー階層**

インテル® Optane™ DC パーシステント・メモリーは、2 つのモードで使用できます。

1. **Memory モード**
2. **App Direct モード**

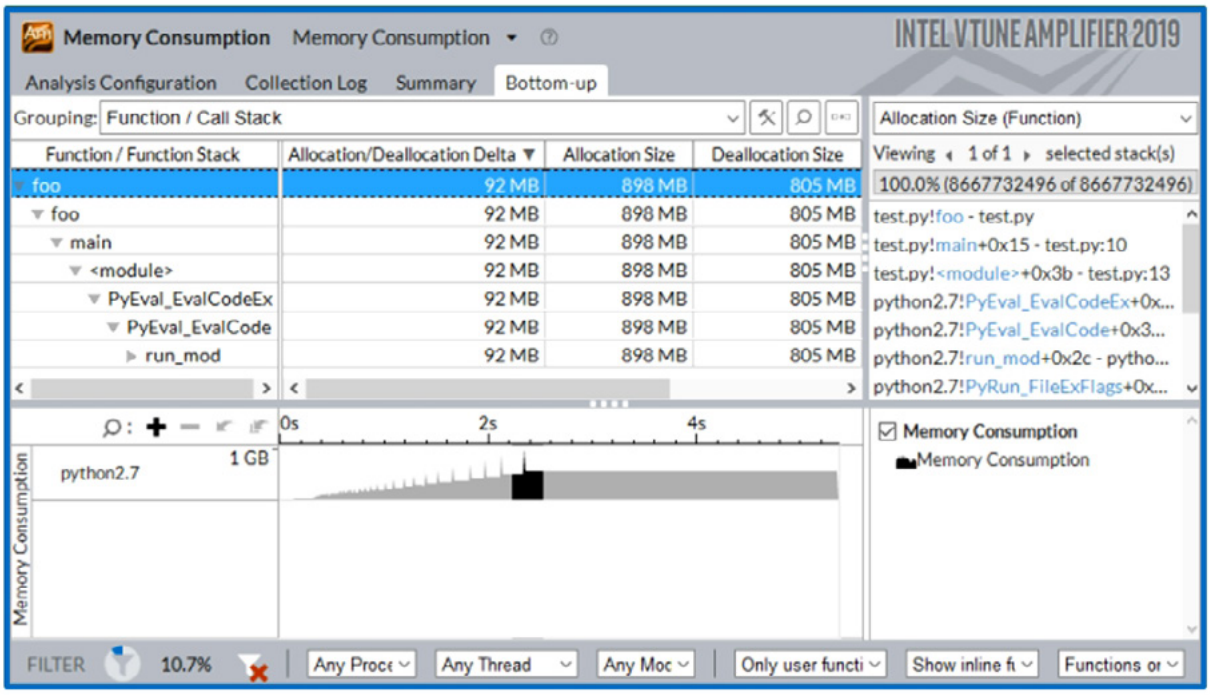
Memory モードでは、インテル® Optane™ DC パーシステント・メモリーはオペレーティング・システムで利用可能なシステムメモリーを拡張します。DRAM はインテル® Optane™ DC パーシステント・メモリーのキャッシュとして使用され、すべてのメモリー管理はユーザーに対して透過的です。コード変更は不要です。

App Direct モードでは、ユーザーは API を利用してインテル® Optane™ DC パーシステント・メモリー上にオブジェクトを手動で割り当てます。インテル® Optane™ DC パーシステント・メモリーは、従来のストレージとしても利用できます。このモードでは、インテル® Optane™ テクノロジーの不揮発性 (永続性) 機能が有効になります。

ワークロードがインテル® Optane™ DC パーシステント・メモリーの利点を得られるかどうか、およびどのモードを選択すべきかを判断するには、動作を特徴付け、特定のパフォーマンス・メトリックを理解することが重要です。インテルは、このプロセスを支援するツールを提供しています。

アプリケーションのメモリー使用量を測定する

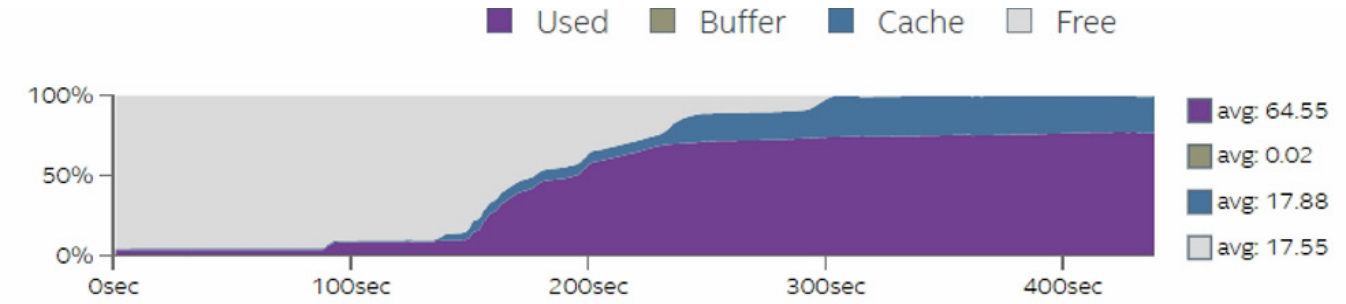
どちらのモードを使用するかにかかわらず、インテル® Optane™ DC パーシステント・メモリーを追加のシステムメモリーとして使用する場合、最初に理解すべきメトリックはワークロードのメモリー使用量です。メモリー使用量の測定には、**インテル® VTune™ Amplifier** を含む、多くのツールを利用できます。インテル® VTune™ Amplifier の**メモリー消費解析**は、アプリケーションによるメモリーの割り当てと割り当て解除を監視し、時間の経過とともに変化するメモリー消費量を追跡します (**図 2**)。



2 **メモリー消費レポート**

メモリー消費レポートの [Bottom-Up (ボトムアップ)] ビューを使用して、ワークロードの最大メモリー使用量を特定できます。また、インテル® VTune™ Amplifier の**プラットフォーム・プロファイラー**機能は、OS の統計情報を使用してメモリー消費を追跡し、利用可能なメモリーの割合のタイムラインを提供します (**図 3**)。

インテル® Optane™ DC パーシステント・メモリーを利用してパフォーマンスを向上するには、物理メモリーの増加によりアプリケーションが利点を得られなければなりません。つまり、メモリー消費がシステムで利用可能な DRAM の合計容量に近くなければなりません。物理メモリーは有限リソースであるため、オペレーティング・システムとほかのプロセスもメモリーを消費することを考慮する必要があります。アプリケーションのメモリー使用量と予想されるほかのメモリー消費量の合計が利用可能な DRAM の容量に近い場合、すべてのデータを DRAM に収めることはできないため、アプリケーションはインテル® Optane™ DC パーシステント・メモリーを使用できます。ワークロードが利用可能なメモリーによって制限されていない場合、メモリーを追加してもパフォーマンスの向上は見込めません。



3 **プラットフォーム・プロファイラーのメモリー使用量解析**

ホットなワーキングセットのサイズを特定する

ワークロードが利用可能なメモリーのほとんどを消費している場合、インテル® Optane™ DC パーシステント・メモリーにより利点が得られる可能性が高くなります。

次のステップでは、Memory モードと App Direct モードでアプリケーションがどのように動作するか特定します。このステップで重要なメトリックは、ホットなワーキングセットのサイズです。ホットなワーキングセットは、アプリケーションによって頻繁にアクセスされるオブジェクトのセットで構成されます。ホットなワーキングセットのサイズは、これらのオブジェクトのサイズの合計です。アクセスが頻繁かどうかの定義はあいまいであるため、このメトリックはメモリー使用量のように簡単に計算できません。しかし、動的メモリー・オブジェクトの解析を有効にするオプションを指定してインテル® VTune™ Amplifier のメモリーアクセス解析を使用することで計算できます。

メモリーアクセス解析を実行すると、GUI の [Bottom-Up (ボトムアップ)] ビューにアプリケーションによって割り当てられた各メモリー・オブジェクトとそのサイズ (括弧内に表示)、そのオブジェクトへのロードアクセス数とストアアクセス数が表示されます (**図 4**)。ロード数とストア数が多いオブジェクトを特定して、それらのオブジェクトのサイズ (括弧内の値) を合計した値がホットなワーキングセットのサイズになります。

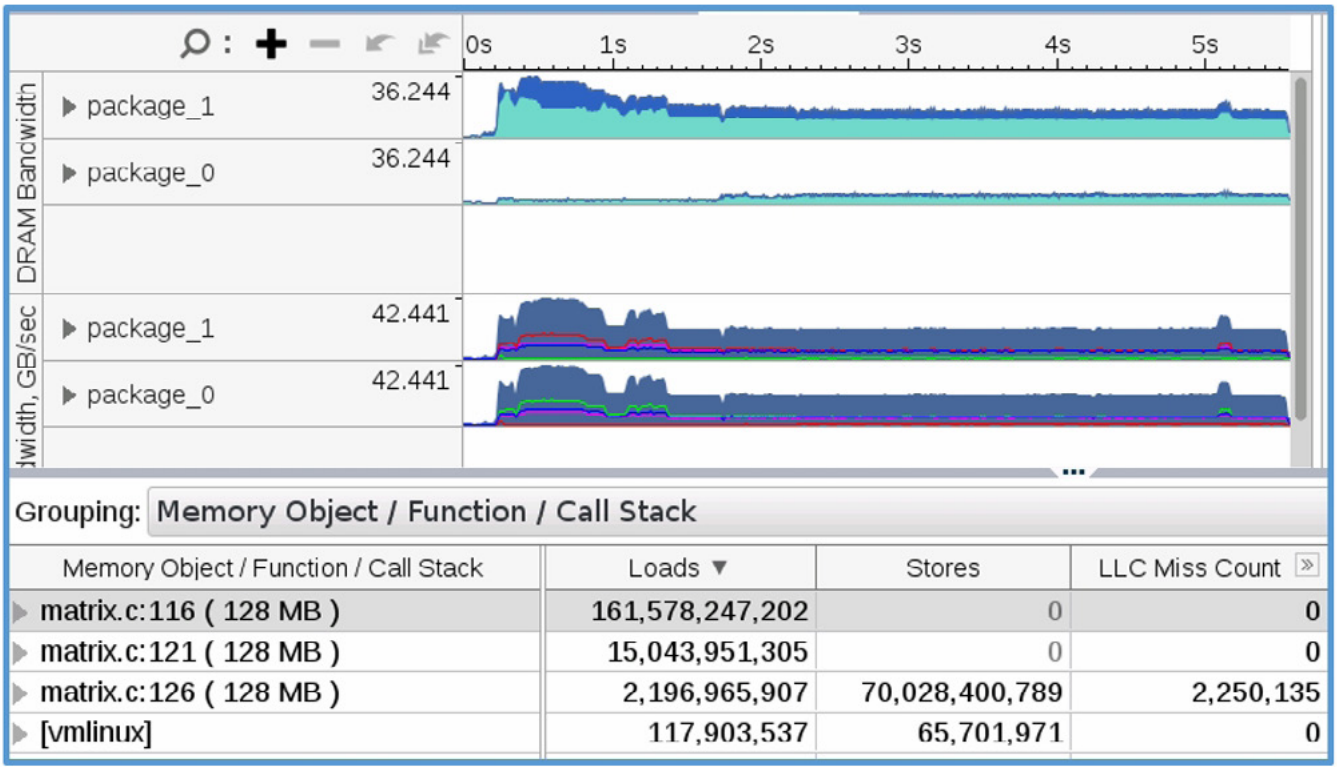
ホットなワーキングセットのサイズは、各メモリーモードでアプリケーションがどのように動作するか判断する上で重要です。

メモリー構成とメモリーモードの選択に関する注意事項

パーシステント・メモリーのパフォーマンスを考慮する際に重要なことは、メモリーアクセスの大部分は DRAM にすべきだということです。パーシステント・メモリーは、DRAM が利用できない場合に使用可能な追加のメモリーとして機能します。

この概念に基づいて、Memory モードは、ホットなワーキングセットが DRAM に収まる (つまり、前のステップで計算したホットなワーキングセットのサイズがシステムで利用可能な DRAM よりも小さい) アプリケーション

に適しています。これにより、ワーキングセットは定期的に DRAM にキャッシュされ、メモリー使用量が利用可能なパーシステント・メモリー容量よりも小さい限り、残りのデータはディスクではなくインテル® Optane™ DC パーシステント・メモリーに格納されます。



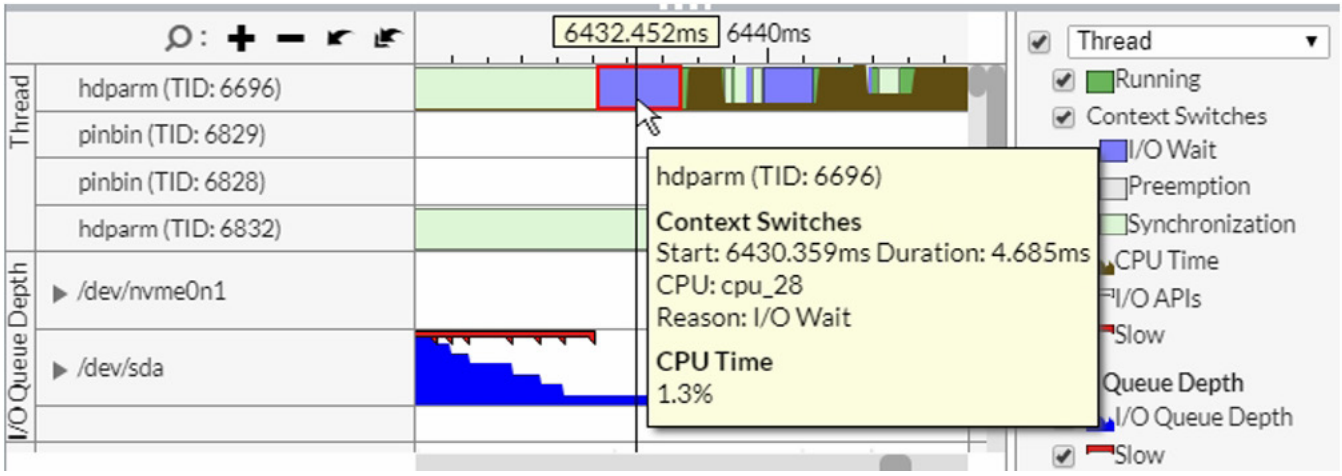
4 **動的メモリー・オブジェクト解析を有効にしたメモリーアクセス解析レポート**

ホットなワーキングセットのサイズが利用可能な DRAM を大きく上回る場合、Memory モードよりも App Direct モードのほうが適しています。App Direct モードでは、オブジェクトを DRAM に割り当てるか、インテル® Optane™ DC パーシステント・メモリーに割り当てるかを、ユーザーが明示的に定義する必要があります。不適切な割り当てはアプリケーション・パフォーマンスの低下につながるため、どちらに割り当てるかうまく選択することが重要です。その開始点として、ラストレベル・キャッシュ (LLC) ミスが多いオブジェクトを特定し、それらをできるだけ多く DRAM に割り当てます。インテル® VTune™ Amplifier のメモリーアクセス解析 (**図 4**) に必要な情報があります。DRAM に割り当てたオブジェクトは、インテル® Optane™ DC パーシステント・メモリーよりも低レイテンシーになることが保証されます。LLC ミスが少ないまたは DRAM に収まらない残りのオブジェクトは、インテル® Optane™ DC パーシステント・メモリーに割り当てます。

割り当てに関してもう 1 つ考慮すべきことは、オブジェクト・アクセスのロード / ストア比率です。インテル® Optane™ DC パーシステント・メモリーのロードは、一般にストアよりもはるかに高速です。ロード / ストア比率の高いオブジェクト (ロードが多いオブジェクト) を特定して、それらはパーシステント・メモリーに割り当て、ストアが多いオブジェクトは DRAM に割り当てます。ロード数とストア数も、メモリーアクセス解析で分かります。

不揮発性ストレージとしてインテル® Optane™ DC パーシステント・メモリーを使用する

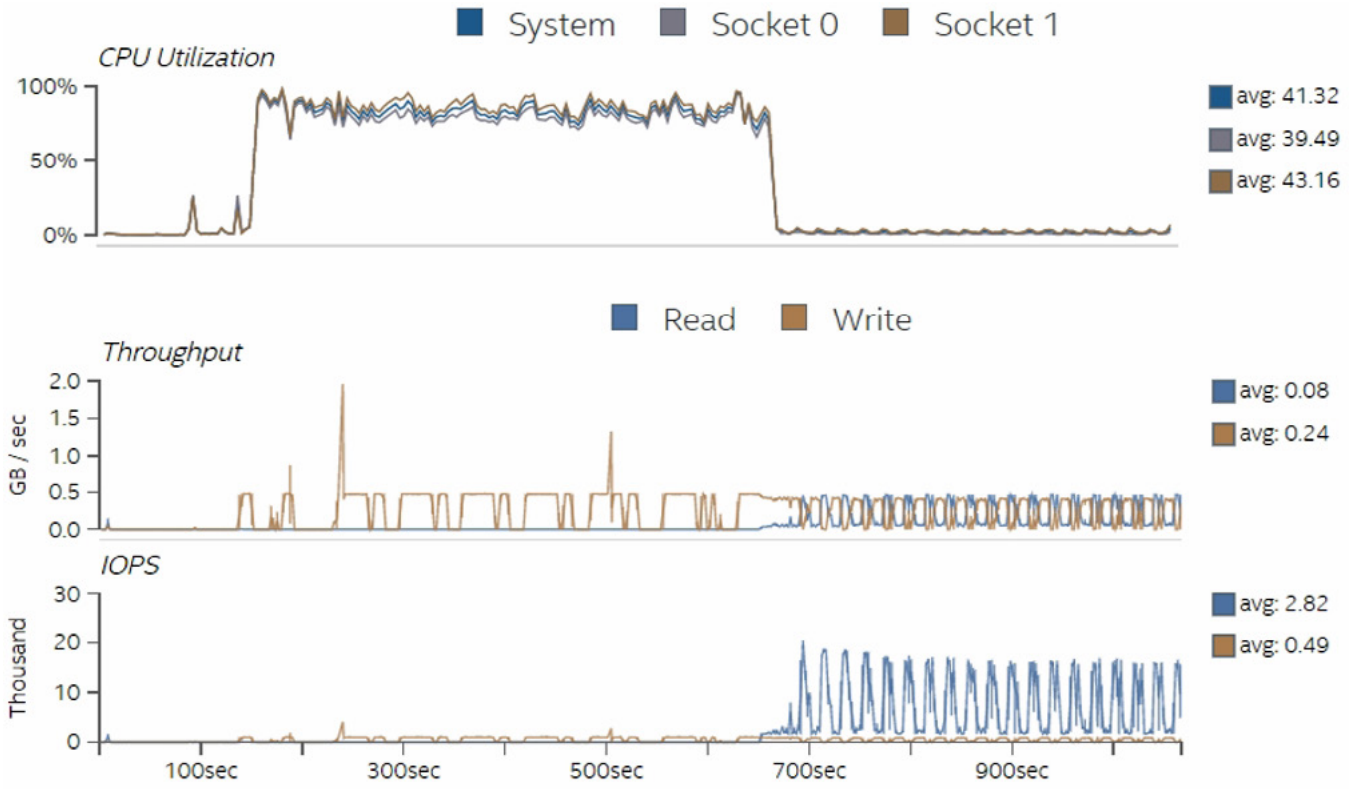
インテル® Optane™ DC パーシステント・メモリーを不揮発性ストレージとして使用する場合は比較的簡単です。アプリケーションにディスクの読み書きに関連するパフォーマンスの問題がある場合、この新しいテクノロジーはパフォーマンスを大幅に向上できる可能性があります。多くの開発者はディスクがボトルネックであることをすでに認識しています。そうでない場合は、インテル® ツールを使用してパフォーマンスの問題の原因がストレージがどうかを調査できます。例えば、インテル® VTune™ Amplifier の**入力と出力解析**は、ディスクアクセスに関連する CPU ストールの診断に役立ちます (**図 5**)。



5 **インテル® VTune™ Amplifier の入力と出力解析**

また、インテル® VTune™ Amplifier のプラットフォーム・プロファイラー解析は、CPU パフォーマンスに関連付けることができるディスクの統計情報を提供します (**図 6**)。

これらのメトリックを使用して、ストレージアクセスに関連するパフォーマンスのボトルネックを特定します。ストレージアクセスが重大なパフォーマンスの問題を引き起こしている場合は、インテル® Optane™ DC パーシステント・メモリーを高速な永続ストレージとして使用することで、パフォーマンスを向上できます。パーシステント・メモリーはファイルシステムの一部として設定でき、頻繁にアクセスするファイルをメモリーモジュールに直接配置できます。



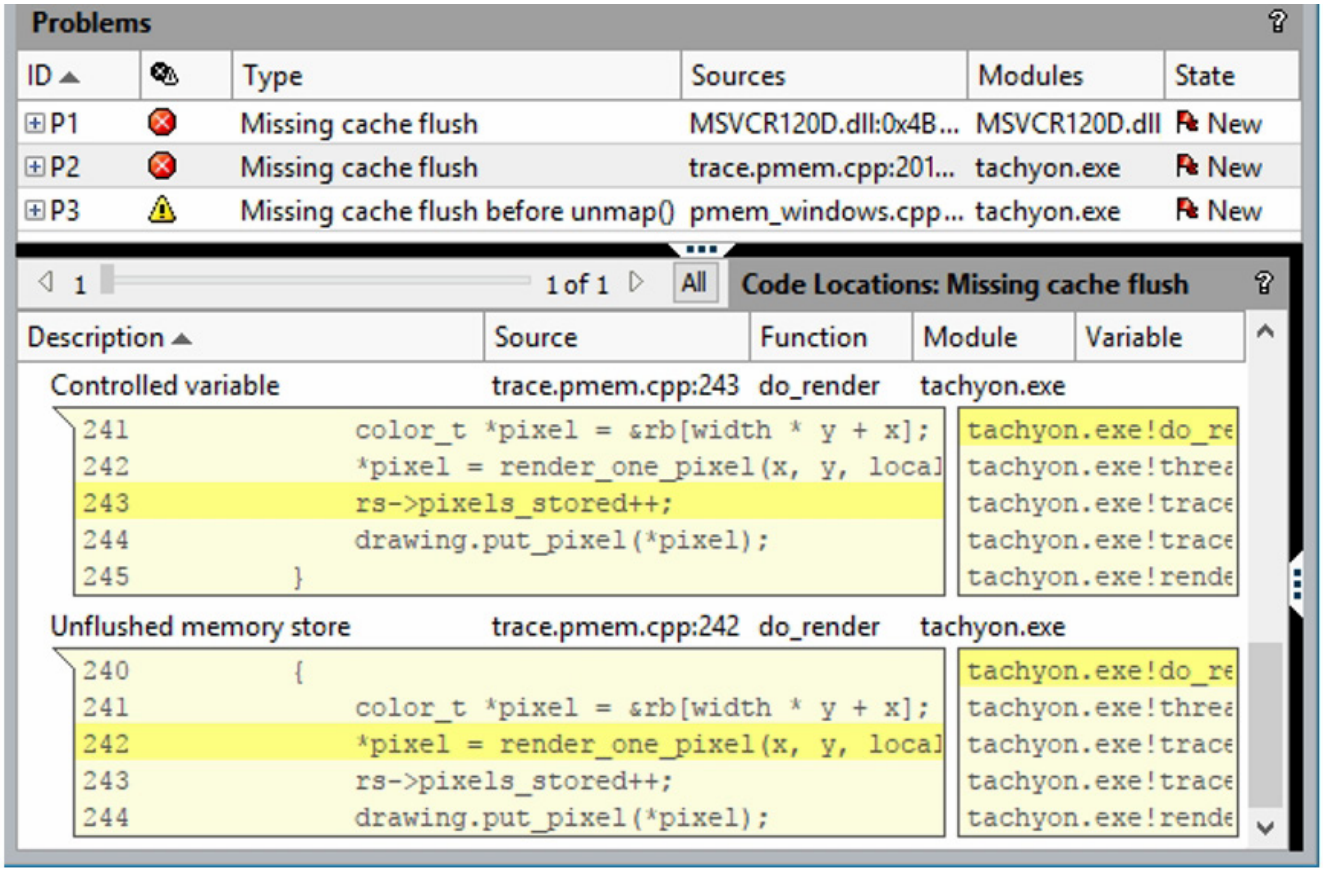
6 **プラットフォーム・プロファイラーの CPU 利用率とディスク使用メトリック**

パーシステント・メモリー・アプリケーションの正当性を検証する

パフォーマンスの問題の特定に加えて、パーシステント・メモリー・アプリケーションのプログラミングにはいくつかのソフトウェアの課題があります。その 1 つは、パーシステント・メモリーへのストアは、キャッシュ階層から排出されメモリー・サブシステムに認識されるまで、実際には永続化されないことです。**インテル® Inspector** の永続性インスペクターは、潜在的なエラーを検出するため開発者が使用できる新しいランタイムツールです (**図 7**)。キャッシュ・フラッシュ・ミスに加えて、以下を検出します。

- **冗長な**キャッシュフラッシュとメモリーフェンス
- **アウトオブオーダー**のパーシステント・メモリー・ストア
- パーシステント・メモリー開発キット (PMDK) の**不正な**取り消しログ

インテル® VTune™ Amplifier
パフォーマンスのボトルネックを素早く特定
無料ダウンロード



7 **インテル® Inspector の永続性インスペクター**

ボトルネックとストレージの問題を解決する

この記事では、新しいインテル® Optane™ テクノロジーの可能性のほんの一部を紹介しました。ビッグデータへの対応に取り組み、システムメモリーや高速ストレージの制限に関連するパフォーマンスの問題に直面している場合は、インテル® Optane™ DC パーシステント・メモリーが役立ちます。インテルはまた、これらの問題によってワークロードがどのように制限され、パーシステント・メモリーをどのように活用できるか理解するのを支援するため、インテル® VTune™ Amplifier やインテル® Inspector などのツールを提供しています。

詳細は、インテル® Optane™ DC パーシステント・メモリーの **Web サイト**とインテル® ソフトウェア・ツールの**ページ** (英語) を参照してください。

インテル® Advisor
現代のハードウェア向けにコードを最適化
無料
ダウンロード

NUMA マイグレーションのパフォーマンスへの影響を測定

トレードオフを考慮してパフォーマンスを最大化

Gurbinder Gill テキサス大学オースティン校 大学院研究助手
Ramesh V. Peri インテル コーポレーション シニア主席エンジニア

最近のメモリーシステムは、コアと DRAM 全体がソケット間で分割される NUMA (Non-Uniform Memory Access) アーキテクチャーを採用しています。各コアは単一のメモリー空間としてメモリー全体にアクセスできます。しかし、ローカルソケット上のメモリーのほうが、リモートソケット上のメモリーよりも高速にアクセスできます。そのため、不均等メモリーアクセス (NUMA、Non-Uniform Memory Access) と呼ばれます。このようにアクセス・レイテンシーが異なるため、ローカル・ソケット・メモリーへのアクセスを常に優先すべきです。

このため、Linux* カーネルは、メモリーページをデータがアクセスされるソケットに移動する NUMA マイグレーションを行います。Linux* は、特定のソケットからページへのメモリーアクセス数やアクセスのレイテンシーなど、

ページ・マイグレーションの決定に必要な情報を保持しています。Linux* の NUMA マイグレーションは、**`numactl`** などのユーティリティーを使用して OS レベルの NUMA 割り当てポリシーが指定されていない限り、デフォルトで有効です。

NUMA ページ・マイグレーションは、単一のマシン上で独自のメモリー割り当てを持つ複数のアプリケーションを実行する場合に非常に役立ちます。システムが共有されるこのようなシナリオでは、特定のアプリケーションに関連するメモリーページをそのアプリケーションが割り当てられているコアに移動することは合理的です。

この記事では、単一のアプリケーションがマシン全体を使用する場合 (ハイパフォーマンス・コンピューティングでは一般的なシナリオです)、NUMA マイグレーションはパフォーマンスの低下につながることを示します。また、多くの場合、**`numactl`** などの OS レベルのユーティリティーよりも、異なるデータ構造の割り当てや割り当てポリシーの設計を細かく制御できるアプリケーションレベルの NUMA 割り当てポリシーの使用が推奨されます。

この記事では、次の 2 つのアプリケーションレベルの NUMA 割り当てポリシーについて説明します (**図 1**)。

- **NUMA インターリーブ :** メモリーページは、NUMA ソケット間でラウンドロビン形式で均等に分割されます (numactl -interleave all コマンドに似ています)。
- **NUMA ブロック :** 割り当てメモリーの均等なチャンクが NUMA ソケット間で分割されます。

1 **NUMA 割り当てポリシー (色分けは 2 つのプロセッサーを示す)**

インテル® Xeon® Gold プロセッサー上での評価

(NUMA インターリーブと NUMA ブロックの両方のポリシーを使用して) サイズ **`m`** のメモリーを割り当て、**`t`** スレッドを使用して各スレッドが連続する書き込みブロックをシーケンシャルに取得して一度だけ書き込む単純なマイクロベンチマークを実行して、NUMA マイグレーションの効果を評価します。

図 2 と**図 3** にメモリー割り当てポリシーの疑似コードと単純なマイクロベンチマーク計算の疑似コードをそれぞれ示します。評価には、4 ソケットのインテル® Xeon® Gold 5120 プロセッサー (56 コア、2.20GHz、187GB DDR4 DRAM) で構成されたシステムを使用しました。評価中、インテル® ハイパースレッディング・テクノロジーは無効に設定されていました。

```
1. /*** Memory allocation Policies ***/
2. /*
3.  * Let m be the amount of memory to allocate.
4.  * Default page size (PS) is 4KB.
5.  * Use MAP_HUGETLB for huge page size (2MB).
6.  * Touching pages bring them into the memory.
7.  */
8.  _PROT = PROT_READ | PROT_WRITE;
9.  _MAP_FLAGS   = MAP_PRIVATE | MAP_ANONYMOUS;
10. mmap_ptr = mmap(0, m, _PROT, _MAP_FLAGS, -1, 0);
11.
12./** NUMA Interleave **/
13./*
14. * Round robin page distribution among threads (e.g. thread 0 gets
15. * a page, then thread 1, then thread n, then back to thread 0 and
16. * so on until the end of the region).
17. */
18.
19./* Parallel loop */
20.for_each_thread{
21.    for(x = PS*ThreadID; x < m; x += PS*numThreads)
22.    {
23.      touch(mmap_ptr[x]); /* bring page in-memory */
24.    }
25.}
26.
27./** NUMA Blocked **/
28./*
29. * Distribute equal chunks of memory among threads (e.g. thread 0 gets
30. * first chunk, thread 1 gets next chunk, ... last thread gets last
31. * chunk).
32. */
33.CS = m/numThreads; /* CS is Chunk Size */
34.
35./* Parallel loop */
36.for_each_thread{
37.    for(x = CS*ThreadID; x < m && x < (ThreadID + 1) * CS; x += PS)
38.    {
39.      touch(mmap_ptr[x]); /* bring page in-memory */
40.    }
41. }
```

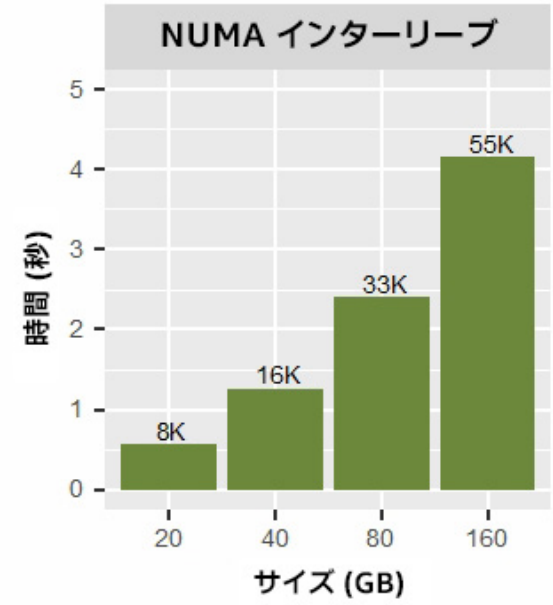
2 **NUMA インターリーブと NUMA ブロックメモリー割り当てポリシーの疑似コード**

```
1. /*** Microbenchmark Computation ***/
2. /*
3.  * Write to every 32nd location in m (memory allocated).
4.  * 32 is randomly chosen.
5.  */
6. NE = m/32; /* Total number of entries. */
7. ECS = NE/numThreads; /* Entries Chunk Size */
8.
9. /* Parallel loop */
10.for_each_thread{
11.   for(x = ECS*ThreadID; x < NE && x < (ThreadID + 1) * ECS; x += 1)
12.     mmap_ptr[x*32] = 42;
13.}
```

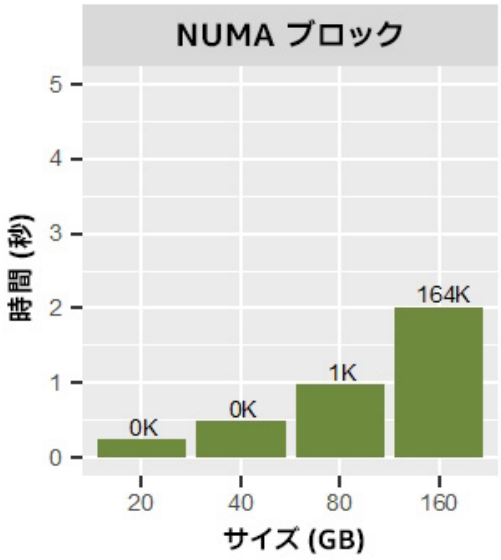
3 **評価に使用した単純なマイクロベンチマーク計算の疑似コード**

異なる NUMA 割り当てポリシーに対する NUMA マイグレーションの効果

図 4 は、t = 56 スレッドとインターリーブ割り当てを使用して、異なるメモリー割り当てサイズ (m) でマイクロベンチマークを実行した時間です。ワークロードが倍になると、実行時間も倍になります。これは想定どおりです。しかし、実行中のページ・マイグレーションの数も大幅に増加しています。NUMA ブロック割り当てでも同様のパターンが見られます (**図 5**)。しかし、ブロック割り当てでは、40GB 以下のワークロード・サイズではページ・マイグレーションが発生しないため、インターリーブ割り当てよりも優れたパフォーマンスを達成しています。メモリーページは、計算中にローカルに割り当てられ、アクセスされます。



4 **56 スレッドと NUMA インターリーブ割り当てを使用して異なるワークロード・サイズでマイクロベンチマークを実行した結果。各バーの上の数値はメモリー・ページ・マイグレーションの数 (千単位) です。**

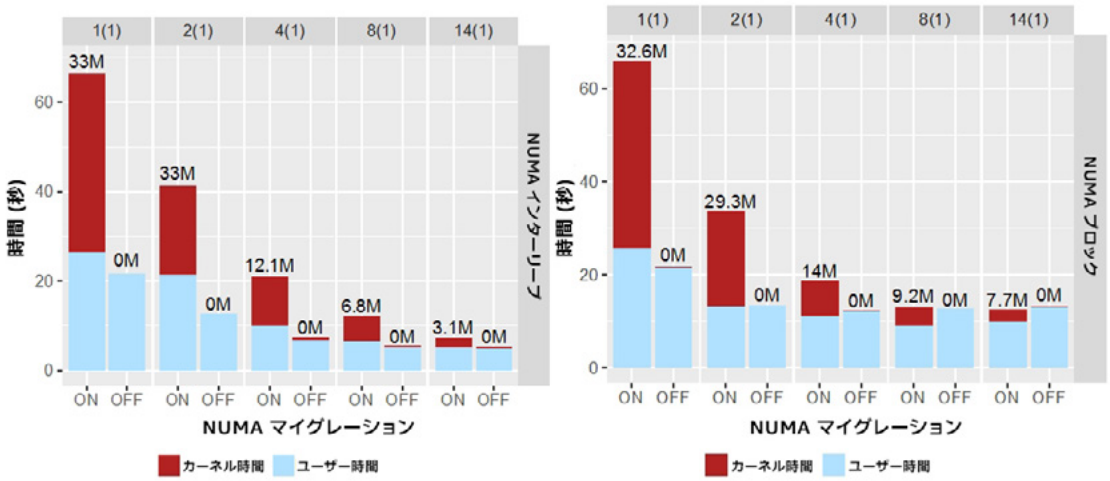


5 **56 スレッドと NUMA ブロック割り当てを使用して異なるワークロード・サイズでマイクロベンチマークを実行した結果。各バーの上の数値はメモリー・ページ・マイグレーションの数 (千単位) です。**

単一ソケット上での NUMA マイグレーションの効果

図 6 は、単一ソケット上で 160GB のワークロードを異なるスレッド数で実行した場合の合計実行時間、およびユーザーコードとカーネルコードで費やされた時間です。メモリー全体がソケット間で均等に分割されるため、各ソケットには約 47GB のメモリーがあります (187GB を 4 ソケットで分割)。ここでは、4 ソケットすべてにわたって 160GB を割り当てました。どちらの割り当てポリシーでも、マイクロベンチマークはスレッド数に応じてスケーリングしています。スレッド数が増加すると実行時間は減少し、ページ・マイグレーションの数も減少します。これは、アプリケーションの実行時間が長くなるほど、OS カーネルによるページ・マイグレーションが増えるためです。

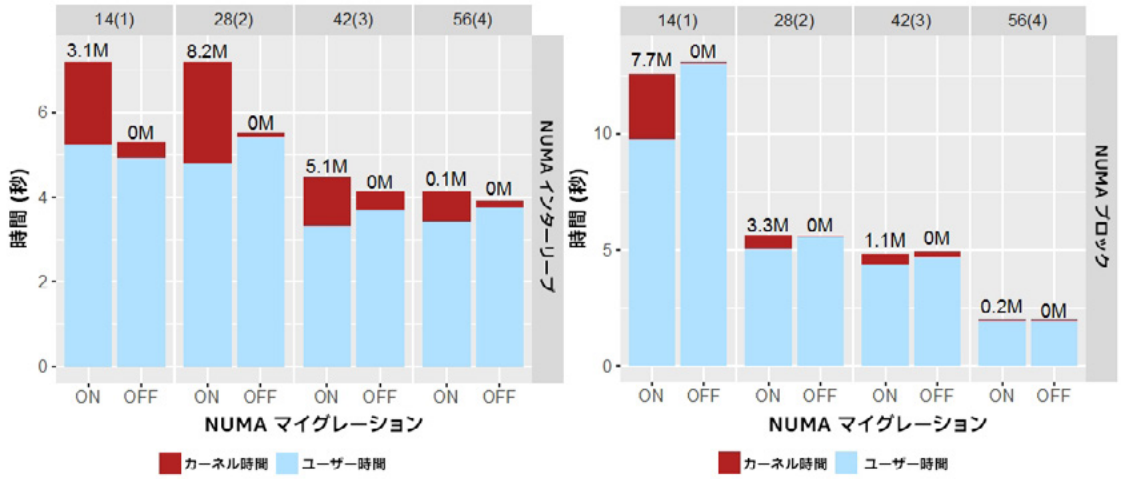
各バーの赤色の部分は、カーネルコードでページ・マイグレーションに費やされた時間です。NUMA マイグレーションが無効な場合、これはほぼゼロになります。NUMA マイグレーションを無効にした場合、インターリーブでは 2.4 倍、ブロックでは 1.6 倍の相乗平均スピードアップが得られます。このことから、NUMA マイグレーションがパフォーマンスに大きく影響することが分かります。



6 **単一ソケット上で固定ワークロード・サイズ (160GB) と異なるスレッド数を使用してマイクロベンチマークを実行した結果。スレッド数は上部に示します。括弧内はソケット数です。各バーの上の数値はメモリー・ページ・マイグレーションの数 (百万単位) です。**

複数のソケット間の NUMA マイグレーションの効果

単一ソケットと同様のパターン (**図 6**) が複数のソケットでも見られました (**図 7**)。各グラフは、異なるソケット数で NUMA マイグレーションを有効にした場合と無効にした場合の結果を示しています。ソケット上のすべてのコアが使用されています。NUMA マイグレーションが無効な場合は常に、カーネルで費やされる時間が減少することが分かります。もう 1 つ興味深い点は、NUMA マイグレーションが無効な場合、ユーザーコードで費やされる時間がわずかに増加していることです。これは、NUMA マイグレーションによりメモリー・アクセス・レイテンシーが減少することを示しています。ただし、NUMA マイグレーションのオーバーヘッドがメリットを上回り、全体のパフォーマンスを低下させる可能性があります。



7 **異なる数のソケット上で固定ワークロード・サイズ (160GB) と異なるスレッド数を使用してマイクロベンチマークを実行した結果。スレッド数は上部に示します。括弧内はソケット数です。各バーの上の数値はメモリー・ページ・マイグレーションの数 (百万単位) です。**

パフォーマンスの最大化

前述の結果から、NUMA マイグレーションなどの OS レベルの機能は、大きなパフォーマンス・オーバーヘッドが発生する可能性があるため、特にマシン全体で単一のアプリケーションを実行する場合 (ハイパフォーマンス・コンピューティングでは一般的なシナリオです)、慎重に使用する必要があります。

アプリケーションの実行時間に対する NUMA マイグレーションの影響は、次のようなさまざまな要因によって異なります。

- 使用される NUMA 割り当てポリシーの**種類**
- プロセッサー上で使用されるソケット**数**

NUMA ページ・マイグレーションによって生じるパフォーマンス・ノイズを回避するには、**図 8** に示すように、該当する OS レベルの機能を無効にします (NUMA マイグレーションはデフォルトで有効です)。

```
# echo 0 > /proc/sys/kernel/numa_balancing
```

8 **OS レベルの機能の無効化**

参考資料

1. **Linux* numactl ユーティリティー (英語)**
2. David Ott, "**Optimizing Applications for NUMA**," Intel Corporation, 2011.

NEWS HIGHLIGHTS

Embedded Vision Alliance が「2019 年最優秀ビジョン製品賞」の受賞者を発表

最優秀開発ツール : OpenVINO™ ツールキット

Embedded Vision Alliance は、2019 年の Embedded Vision Summit において「年間最優秀ビジョン製品賞」の受賞者を発表しました。この賞は、ビジュアル AI やコンピューター・ビジョンなど、急激に成長しているこの分野で業界をリードするテクノロジー企業の革新性と卓越性を表彰するものです。

Embedded Vision Alliance の創設者である Jeff Bier 氏は次のように述べています。

「今日、ビジュアル AI を実現するテクノロジーは、多様な成長市場において需要が高まっています。その結果、この分野での革新が加速しています。「年間最優秀ビジョン製品賞」は、インパクトのある革新的なテクノロジーを提供している企業を表彰するものです。」

この記事の続きはこちら (英語) でご覧になれます。>

ビジョンを
コード化しよう

エッジからクラウドまで AI を高速化しましょう。
OpenVINO™ ツールキットは、コンピューター・
ビジョンのワークロードをスピードアップし、
ディープラーニングの展開を合理化し、
インテル® プラットフォーム間での
ヘテロジニアス実行を容易にします。

無料のダウンロード (英語) >

intel Software

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 (software.intel.com/articles/optimization-notice#opt-jp) を参照してください。
Intel、インテル、Intel ロゴ、OpenVINO 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 またはその子会社の商標です。
* 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。
© Intel Corporation

Python* での並列処理：numexpr によるベクトル化の指示

配列と数式の計算パフォーマンスを大幅に向上

Fabio Baruffa 博士 インテル コーポレーション テクニカル・コンサルティング・エンジニア

Python* では、Numba[1] による JIT (ジャストインタイム) コンパイルから Cython による C スタイルのコードまで、さまざまなベクトル化 (命令レベルの並列処理) の手法を利用できます。Python* で並列処理を行う 1 つの方法は、Python* の数式をハイパフォーマンスなベクトル化されたコードに変換するシンボリック・エバリュエーター、numexpr を使用することです。numexpr は、すべてを一度にコンパイルする代わりに、要素のチャンクをベクトル化して (Python* コードから利用可能な高速化されたオブジェクト・カーネルを作成して) 並列処理を行います。この記事では、Python* コードをリファクタリングして numexpr の機能を活用する方法を説明します。

数式の並列化

Python* のシンプルな構文による柔軟性により、開発者は NumPy* や SciPy* のようなライブラリーを利用して計算のプロトタイプを迅速に作成できます。しかし、Python* 言語は、現代のベクトル・プロセッサーやマルチコア・プロセッサーのパフォーマンスを引き出す必須条件である並列処理を考慮して開発されていません。では、どうすれば Python* を使用して数式をベクトル化できるのでしょうか。

数式は、計算を実行する数と数学記号を含む数学的な文です (例 : **`11*a-42*b`**)。Python* では、この式は numexpr パッケージで定義された配列 **`a`** と **`b`** を利用して処理できます。この場合、配列の式は、組込みの並列処理とベクトル化を利用することで、標準 Python* の計算よりも高速に処理できます。

パフォーマンスを高めるため、numexpr は**インテル® マス・カーネル・ライブラリー (インテル® MKL)** に含まれる最適化されたベクトル・マス・ライブラリー (VML) を使用しています。これにより、メモリーに隣接して格納されたベクトルを操作する数学関数 (正弦、指数、平方根など) の評価を高速化します。

numexpr の一般的な NumPy* 呼び出しのリファクタリング

numexpr パッケージを利用するには、計算文字列を **`evaluate`** 関数に渡します。すると、計算全体は低レベルコードのままオブジェクトにコンパイルされ、結果が Python* レイヤーに返されます。そのため、Python* インタープリターの呼び出しを回避できます。

NumPy* 配列の単純な式を計算する例を見てみましょう。

```
import numpy as np
import numexpr as ne
a = np.arange(1e6)
b = np.arange(1e6)
%timeit 11*a-42*b
14.2 ms ± 826 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
%timeit ne.evaluate("11*a-42*b")
3.51 ms ± 248 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

このケースでは、VML のベクトル組込み関数により 4 倍にスピードアップしています。ライブラリーは、コピーのオーバーヘッドが低いインプレースの操作を実行することもできます。

次に、VML の利点が明確に現れる、数学関数を使用した場合の numexpr のスピードアップを見てみましょう。

```
import numpy as np
import numexpr as ne
a = np.arange(1e6)
b = np.arange(1e6)
%timeit np.exp(a)**2 + np.sqrt(b)**3
498 ms ± 11.2 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
%timeit ne.evaluate("exp(a)**2 + sqrt(b)**3")
26.4 ms ± 2.23 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

このケースでは、インテル® MKL の **sqrt** 関数を使用することで、さらに高いパフォーマンスを引き出しています。スピードアップは約 19 倍です。これは、一部の式で NumPy* ライブラリーが期待されるアクセラレーションを提供していないことを示しています。さらに、numexpr 実装は中間結果のメモリー割り当てを回避することで、キャッシュを効率良く利用し、メモリーのオーバーヘッドを軽減します。これらの最適化の恩恵は、大きな配列の計算でより顕著に表れます。

numexpr エバリュエーターの制御

numexpr は VML を内部的に使用しているため、ライブラリーで利用可能な型の数学関数のみ計算します。ユニット・インクリメントを含む実数型と複素数型のベクトル引数、整数型、ブール型も処理します。配列の型が評価式で一致しない場合、推論規則に従ってキャストされます。

パフォーマンスは、ベクトル化およびメモリーのオーバーヘッドを含む、多くの要因に依存するため、チューニングと精度 (およびスレッド数) の制御に VML の一部の関数を使用します。

VML のバージョン情報を取得するには、インストール状態を確認する **get_vml_version()** 関数を呼び出します。ベクトル関数はすべて、**set_vml_accuracy_mode(mode)** 関数を介して次の精度モードをサポートします。

- **High**。High Accuracy (HA) に相当。デフォルトモード。
- **Low**。Low Accuracy (LA) に相当。最下位 2 ビットの精度を下げてパフォーマンスを向上します。
- **Fast**。Enhanced Performance (EP) に相当。精度を大幅に下げて優れたパフォーマンスを提供します。仮数のビットの約半分が正しい値を示します。

詳細は、インテル® MKL デベロッパー・リファレンス[2] および numexpr の公式ドキュメント[3] を参照してください。

numexpr はスレッド数の制御にも使用できます。**`set_num_threads(nthreads)`** 関数は VML によって使用される最大スレッド数を設定します。戻り値は、現在の環境のスレッド数の以前の設定です。さらにパフォーマンスが向上するように、前の例のスレッド数を変更します。

```
import numpy as np
import numexpr ad ne
a = np.arange(1e6)
b = np.arange(1e6)
ne.set_num_threads(4)
%timeit ne.evaluate("exp(a)**2 + sqrt(b)**3")
7.2 ms ± 137 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

スピードアップは 3.7 倍、並列処理の効率は 93% になりました。この例では、スレッド数を増やすとパフォーマンスが向上します。

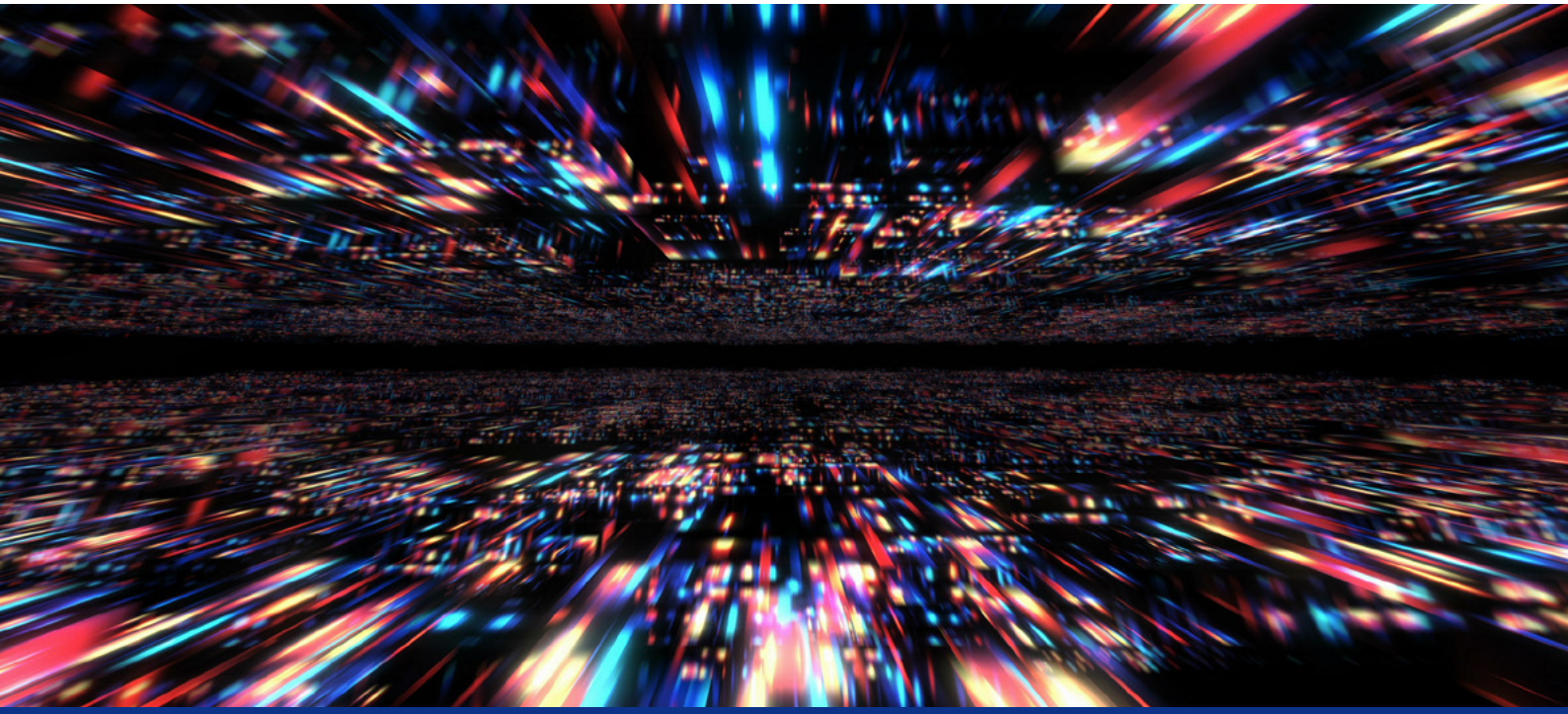
```
ne.set_num_threads(8)
%timeit ne.evaluate("exp(a)**2 + sqrt(b)**3")
3.94 ms ± 191 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

NumPy* の代わりに numexpr を使用すると、VML パフォーマンス・ライブラリーの効果により、配列と数式の計算パフォーマンスが大幅に向上します。numexpr の構文は NumPy* に類似しており、コードにいくつかの単純な関数呼び出しを追加するだけで、numexpr に対応させることができます。

参考資料

1. **The Parallel Universe 36 号**
2. **インテル® MKL デベロッパー・リファレンス (英語)**
3. **numexpr 2.0 ユーザーガイド (英語)**

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。



インテル® Xeon® プロセッサーでインテル® AVX-512 を利用してオープン・シェーディング言語を加速

フル・レンダリングのデジタルコンテンツ制作を最大 2 倍高速化

Steena Monteiro インテル コーポレーション ソフトウェア・エンジニア
Alex M. Wells 同主任エンジニア

オスカー受賞実績のあるオープン・シェーディング言語 (OSL)[1] は、デジタルコンテンツ制作の事実上のオープンソース標準です。OSL は業界全体で広く採用され、Pixar の RenderMan* やソニー・ピクチャーズ・イメージワークスの Arnold などのレンダラー、および 100 本を超える映画で使用されています[2]。

インテルは、最近のインテル® プロセッサーで利用可能な**インテル® アドバンスト・ベクトル・エクステンション 512 (インテル® AVX-512)** を活用する SIMD (Single Instruction Multiple Data) 命令を利用するため、OSL アーキテクチャーの再構築に取り組んでいます。SIMD OSL は、SPMD (Single Program Multiple Data) を使用して、既存の OSL シェーダーと OpenMP* による OSL ライブラリー関数の明示的なベクトルを処理します。この取り組みは次の 2 つのステップに要約できます。

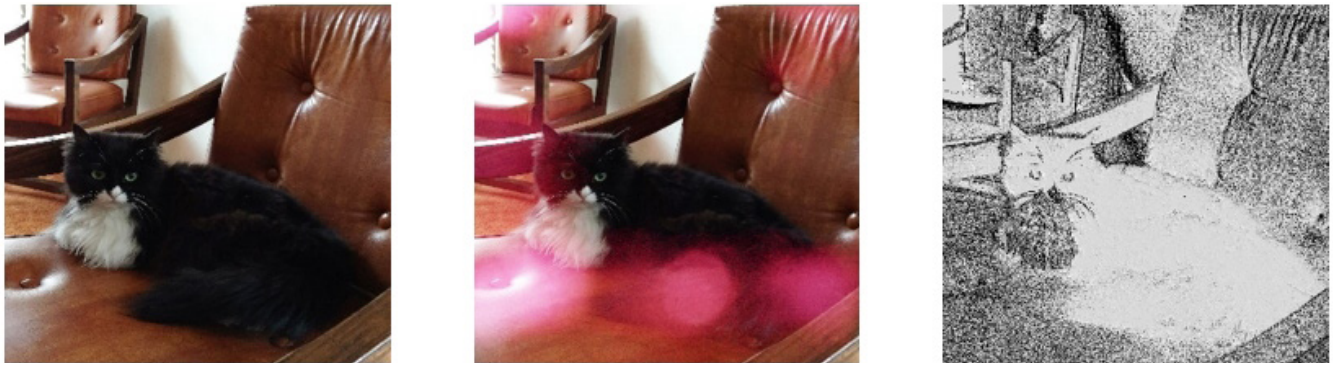
1. レンダリング時間最適化中の JIT (ジャストインタイム) コンパイルでの**ベクトル LLVM IR の生成**
2. OSL ランタイムへの**バッチ・インターフェイスの追加**

この取り組みは SIGGRAPH 2016 で開始され、現在ではインテル® アドバンスト・ベクトル・エクステンション (インテル® AVX)、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)、およびインテル® AVX-512 をネイティブにサポートし、拡張ライブラリー機能、デバッグのサポート、広範なテスト・フレームワークを含むように改良されています。SIMD OSL は、インテルの SIMD ハードウェア上でシェーディング・ネットワーク全体を並列に実行でき[4]、CPU の性能に基づいて単一の CPU 命令で 8 つまたは 16 のデータポイントに対する同時処理を動的にスケジュールします。Pixar によれば、Pixar の RenderMan* 22.5 には「OSL の SIMD ハードウェア・アクセラレーション機能 (**インテル® Xeon® スケーラブル・プロセッサー**でインテル® アドバンスト・ベクトル・エクステンションを利用した場合、フル・レンダリングが最大 2 倍高速化、平均スピードアップが 15%) が含まれています。」[3,5]

この記事では、SIMD ハードウェア上でインテル® AVX-512 の機能を活用するために OSL のアーキテクチャーを再構築した際のインテルの取り組みを紹介します。特に、OSL データ型向けのメモリーレイアウト、分岐制御フローのマスク、ベクトルコード生成のための LLVM バックエンドの追加を含む、SIMD OSL で使用されるソフトウェア・エンジニアリング手法について説明します。

シェーディングとレンダリング・ソフトウェアにおける役割

物理ベースのレンダラーでのシェーディングは、3D シーンのオブジェクトにサーフェスの記述を提供することを意味します。サーフェスの記述は、色の値、ライティングの値 (反射、拡散、スポット)、テクスチャー (金属、セラミック、大理石など) を含みます (**図 1**)。大きなシーンのシェーディングは、特定のシェーディング動作を表す、いくつかの個々のシェーダーノードを使用して行われます。個々のシェーダーは、手続きに従って複雑なシェーディング効果を作成するため、有向非巡回グラフで接続されます。プロダクション・シェーダーは、多くのシェーディング動作を表す数百から数千のシェーダーになることがあります。プロダクション・レンダラーは、OSL によるシェーディング時間に処理時間の最大 80% を費やします。シェーディング・ネットワークの複雑さにより、レンダリングでのシェーディングは、オンサイトとクラウド上の両方のレンダリング・ファームで非常に多くの CPU 時間を消費します。



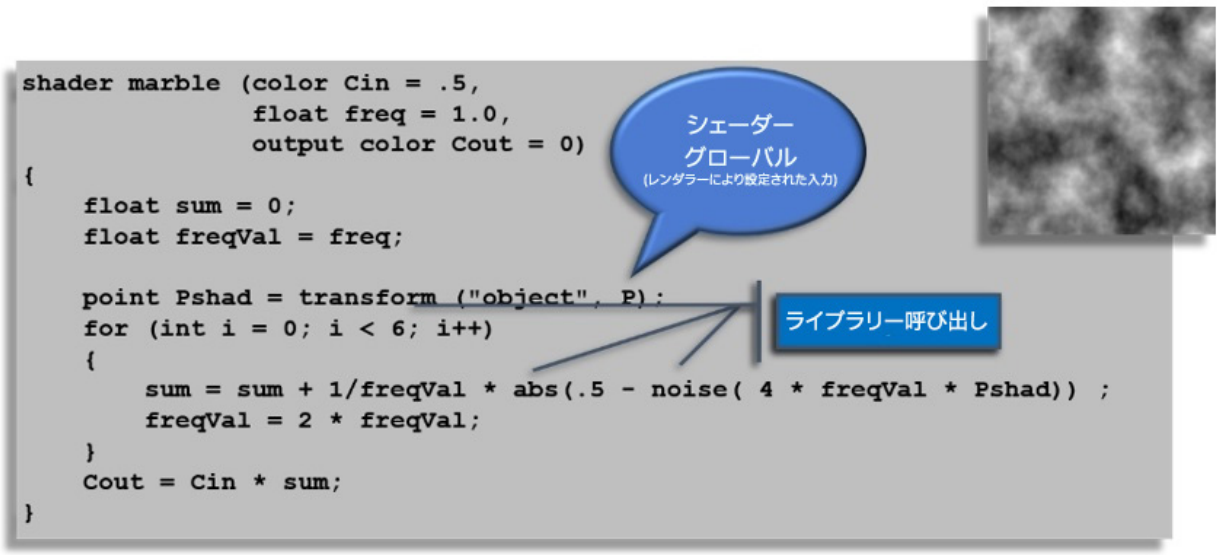
1 **ライティングとテクスチャーの例**

シェーディング・ネットワークのシェーダーは C++ で記述できますが、C++ には多くの課題があります。これらの C++ シェーダーには、入力パラメーターの値やコンパイル時にシェーディングされているシーンの形状などの、関連するシェーディング情報が含まれません。シェーダーは要求されたシェーディングのモードを知らず、周囲のシェーディング・ネットワークの状態に気付きません。C++ で記述されたシェーダーのパフォーマンスを保守および最適化することは困難です。そもそも、これらのシェーダーには移植性がなく、複雑な入れ子の制御フロー (分岐の多いテスト) をテストする必要があります。シェーダーのパフォーマンスを向上するには、C++ を最適化する際に高度な専門知識が必要になります。

ソニー・ピクチャーズ・イメージワークスにより設計された OSL[6] は、パフォーマンス重視のシェーディング・ネットワークを記述する困難さを軽減します。C スタイルで構成された OSL は、シェーダーを記述するために設計された領域特化型の言語です。物理ベースのレンダリング向けに設計された OSL は、シェーディングに特化しており、(レンダラーに存在する) レイ・トレーシング、サンプリング、統合、タイトループを含みません。OSL は、JIT でコンパイルされた広範なランタイム最適化を備えたマシンコードにより最大限のパフォーマンスを引き出し、遅延評価を備えたシェーディング・ネットワークを生成します。

オープン・シェーディング言語

OSL のシェーダーは、シーンをレンダリングするときに特定のタスクを実行する、入力と出力を含むプログラムです[6]。**図 2** は、大理石テクスチャーを表す単純な OSL シェーダーを示しています[7]。このシェーダーの重要な要素はシェーダーグローバル **P** および OSL ライブラリーの **abs()** 関数と **noise()** 関数です。シェーダーグローバルは、位置、サーフェスの法線、光線の方向などの変数で、レンダラーにより提供され、シェーダーにより処理されます。



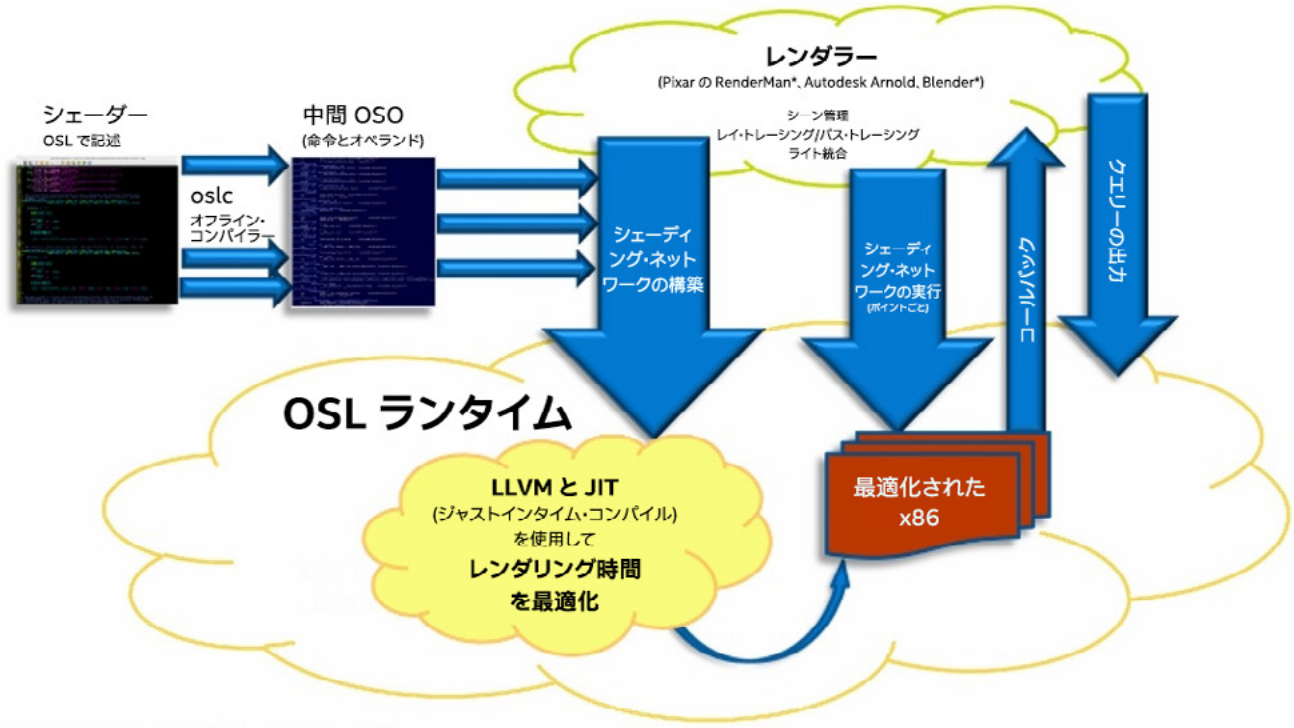
2 **OSL で記述されたシェーダー (marble.osl[7])**

C++ シェーダーに対する OSL シェーダーの利点

OSL で記述されたシェーダーは OSLC (OSL コンパイラー) によりシェーダー操作を表すオペランドと命令を含む中間 oso ファイルにコンパイルされます。複数のシェーダーがコンパイルされて大きなシェーディング・ネットワークを構築します。OSL ランタイムは、LLVM を使用してシェーダーの中間表現 (IR) を生成し、最適化された x86 コードを生成します (**図 3** を参照)。レンダリング時間の最適化により、プロダクション・シーンは操作数、シンボル数、空のシェーダー・インスタンスの大幅な減少の恩恵を受けます。このシーンが受けた恩恵を数字で示すと次のようになります。

- **操作数が 99% 減少** (2 億 8000 万から 268 万)
- **シンボル数が 98.8% 減少** (1 億 6100 万から 190 万)
- **63% の最適化** (空のシェーダー・インスタンスの除去による)[8]

レンダリング時間の最適化に LLVM と JIT を活用できるため、OSL のパフォーマンスはプリコンパイル済み C++ シェーダーよりも高くなります。



3 **OSL から最適化された x86 コードを生成する OSL フレームワーク**

SIMD OSL の概要

C++ シェーダーに対する利点はありますが、OSL のオリジナル形式では、新しい SIMD 命令を十分に活用できません。インテル® ストリーミング SIMD 拡張命令 (インテル® SSE) を使用したブロックのベクトル化は、4 つのレーンのみ使用し、複雑なノイズ、数学、文字列、テクスチャー関数を制限付きでサポートします。SIMD OSL は SPMD モデルを使用してレンダラーに対する (シェーディング・ネットワークの複数のポイントを処理する) バッチ・インターフェイスを作成します。SIMD OSL には次のような機能があります。

- **OSL 言語仕様の保持 :** SIMD OSL は、OSL ライブラリーのユーザー・インターフェイスを変更しません (つまり、オリジナルの OSL シェーダーと同じです)。
- **新しいバッチ・インターフェイス**により、レンダラーはシェーディング・ネットワークのポイントのバッチを処理できます。その場合でも、オリジナルのシングルポイント・インターフェイスが保持され、シェーディング・ネットワークの 1 つのポイントがレンダラーで処理されます。
- **ワイドなバックエンドによる SIMD コードの生成 :** SIMD OSL は、ベクトル化される中間表現 (IR) のデータ型に LLVM ベクトルデータ型 **`<16 * float>`** を使用します。
- **新しいワイドなライブラリー :** アーキテクチャーの再構築により、SIMD OSL は、OpenMP* の明示的なベクトル化を使用して、OSL 関数 (数学、ノイズなどの関数ファミリー) にワイドなインターフェイスを提供します。
- **包括的なテスト・フレームワークの作成 :** uniform (均一)、varying (可変)、定数オペランドの組み合わせに対して OSL ライブラリー関数をテストします。

SIMD OSL のアーキテクチャー

SIMD OSL のアーキテクチャーの再構築により、既存のシングルポイント OSL の 3 つの主な構造が変更されます。

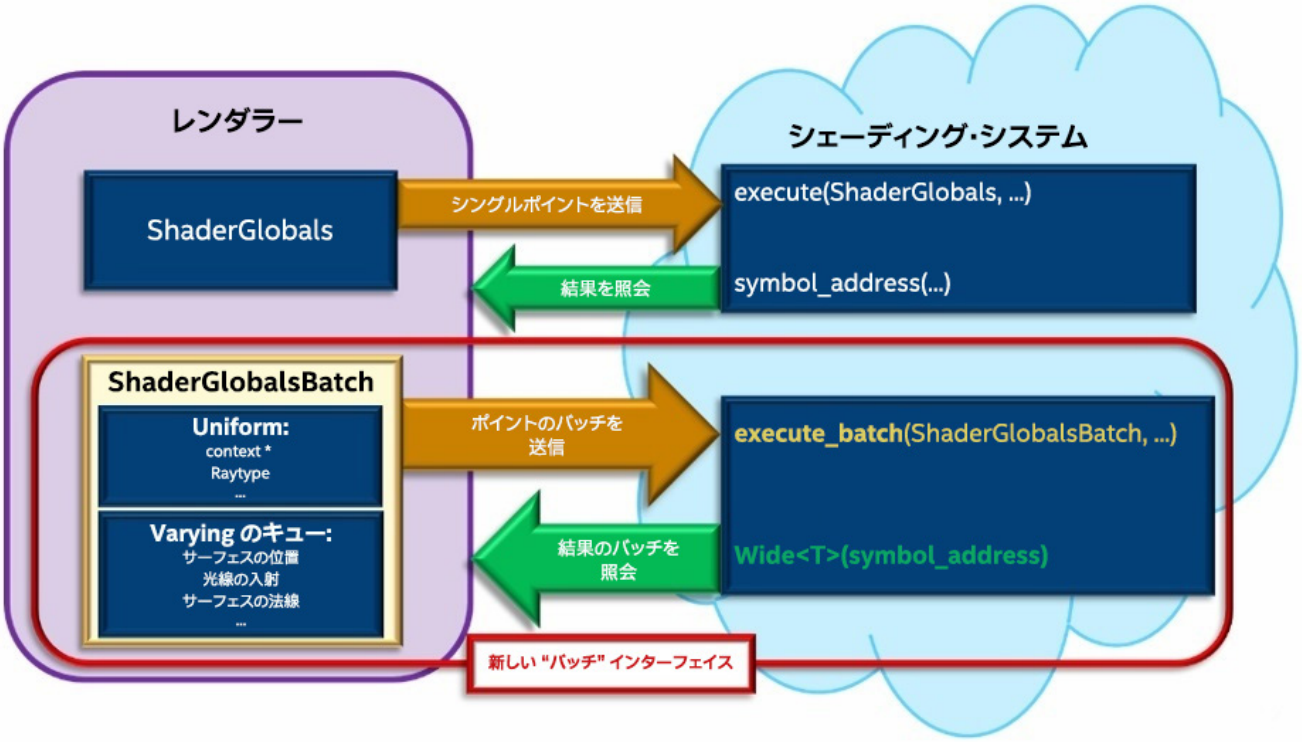
- バッチ・インターフェイス (**図 4**) の**提供**
- 可変データ型を表現およびアクセスするワイドなアクセサーの**追加**
- 異なる SIMD レーンでの値の分散の**制御**

SIMD OSL のバッチ・インターフェイス

SIMD OSL の変更の 1 つは、均一または可変によって、バッチ・サブシステムにシェーダーグローバルを格納する方法が異なることです。ポイントのバッチを使用することは、レンダラーがポイントのセットをシェーディング・システムに送り、シェーディング・システムがレンダラーの可変グローバルのキューから結果のセットを照会できることを意味します。レンダラーが SIMD OSL を使用できるようにするには、新しいワイドなインターフェイスをサポートし、ワイドなコールバックにも対応する必要があります。

SIMD OSL のデータ型

可変であると実証されない限り、変数はすべて均一であると見なされます。可変変数は、依存性がシェーダーグローバルまでトレースできる (可変であることが知られている) 変数です。例えば、サーフェスの位置、光線の入射、サーフェスの法線などのシェーダーグローバルは常に可変です。



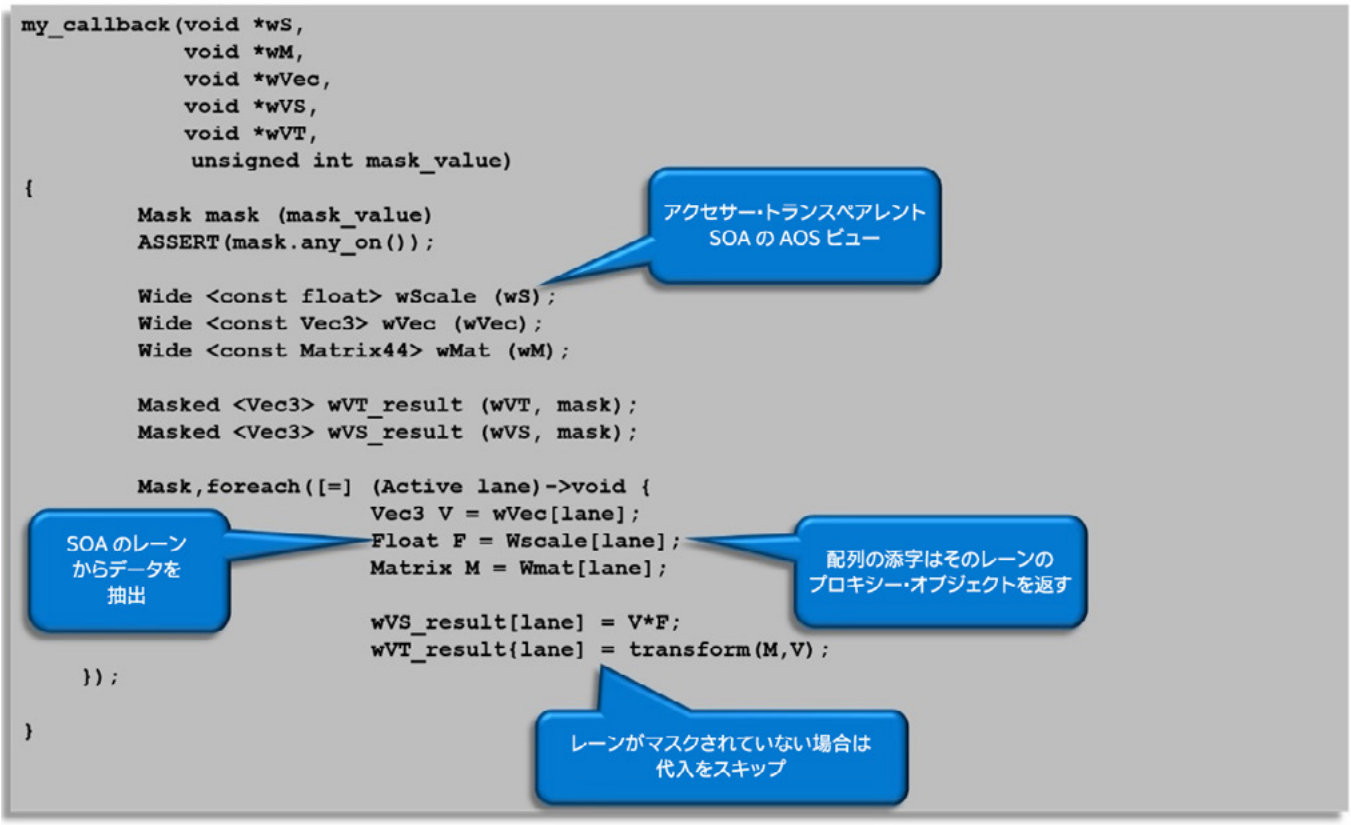
4 **SIMD OSL のバッチ・インターフェイス**

OSL は、タプルの集約データ型 (ベクトル、色、ポイント、行列など) を含んでいます。配列で使用された場合、データは構造体配列 (AOS) 形式で格納されます。AOS 形式の値をベクトルレジスターに格納することによるループのオーバーヘッドは、ベクトル化を妨げます。インテルの SIMD Data Layout Templates (SDLT)[9] は、SIMD 対応の内部メモリーレイアウトを備えたコンテナーを使用します。伝統的に、この形式のデータ型は値の配列として表現されます。SDLT コンテナーは、基本的なメモリーレイアウトとオブジェクトのオリジナル表現の間のインポートおよびエクスポート・プリミティブにアクセサー・オブジェクトを提供します[9]。SDLT ライブラリーから着想を得て、SIMD OSL ライブラリーは、さまざまなデータ型とレンダラーによるコールバックをサポートする、ワイドなアクセサーを提供します。ワイドなアクセサーは、各 SIMD レーンのデータ型の配列に似ていて (**図 5**)、基本的な配列構造体 (SOA) レイアウトを抽象化します。マスクされたアクセサーは、マスクによりアクティブでないデータレーンをスキップします。

SIMD OSL のアルゴリズムをマスキングして入れ子の制御フローを追跡

SIMD OSL は、マスキングを使用して **`if`** 条件で分岐するレーンを追跡します。マスクは、一方のパスで実行するポイントを追跡します。次に、各分岐で適切なレーンで有効にされたマスクを使用して両方のコードパスが実行されます。しかし、この手法は入れ子の制御フローでは複雑になります。異なるコードパスのさまざま

なレーンのポイントとマスクを追跡することにより、パフォーマンスがボトルネックになります。ボトルネックを解消し、正しい制御フローで実行する正しいポイントを追跡するため、SIMD OSL はスタックを使用して各条件文でマスクを追跡します (**図 6**)。

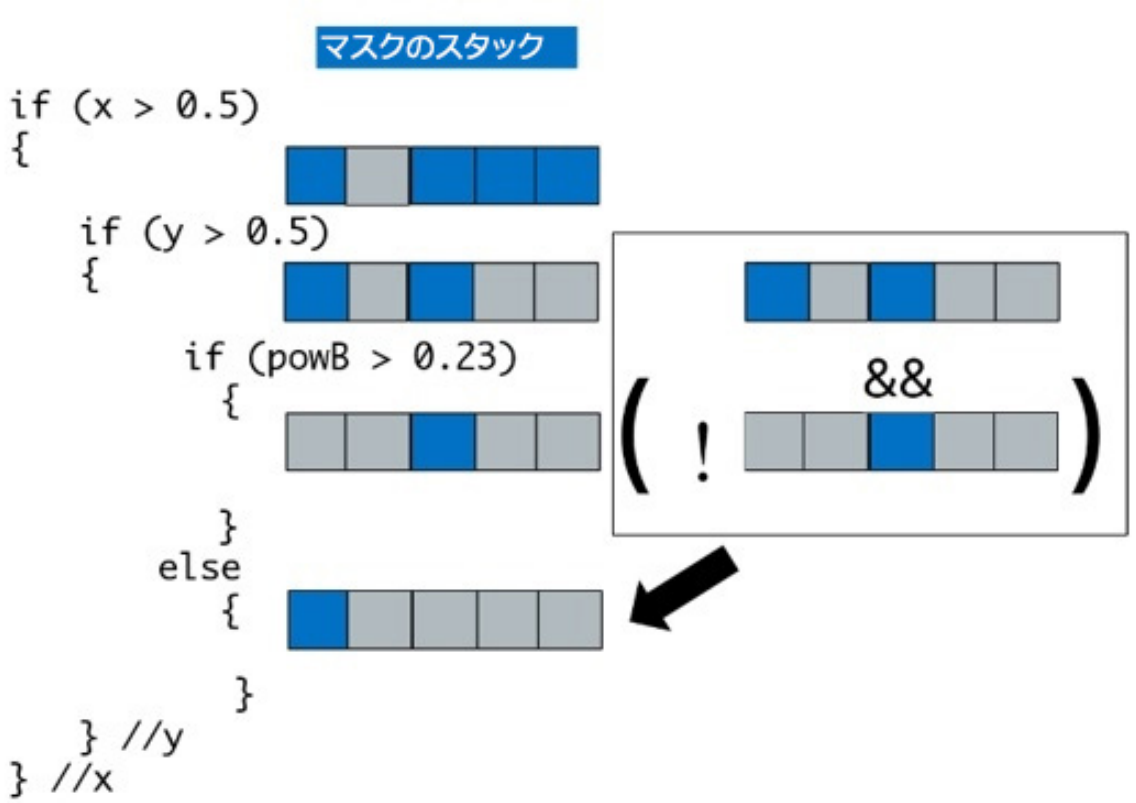


5 **SIMD OSL の可変データのアクセス**

SIMD OSL の LLVM バックエンド

SIMD OSL は、JIT ターゲット固有のコードに LLVM を使用します。プリコンパイル済みライブラリー関数について、SIMD OSL はサポートしているプラットフォーム (インテル® AVX、インテル® AVX2、およびインテル® AVX-512) ごとに異なる共有ライブラリーを生成します。適切なライブラリーが実行時にロードされ、各 OSL プリコンパイル済みライブラリー関数のアドレスと JIT コードがリンクされます。

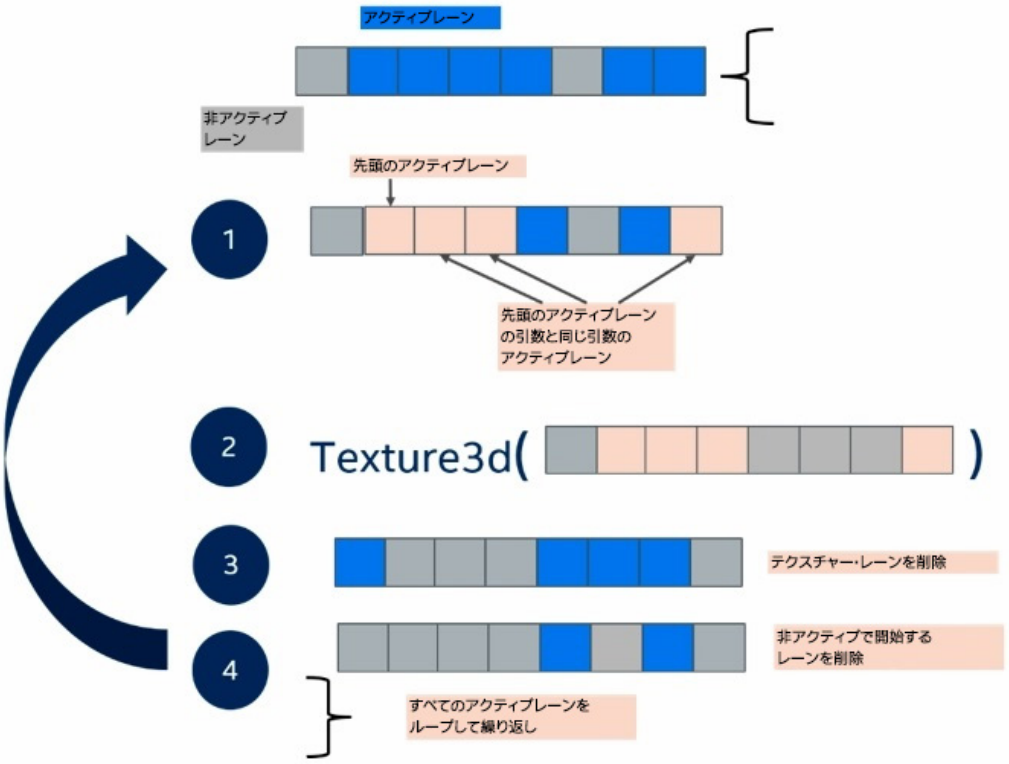
OSL はオリジナル形式で、すべての関数ファミリーをサポートする LLVM バックエンドを含んでいます。インテルは、可変オペランドと制御フローに取り組む際に、ワイドなデータ型とマスキングによる制御をサポートするように、このバックエンドを再設計しました。OSL の **andor** 関数では、4 ステップのプロセスを使用して SIMD サポートを追加しました。



6 **SIMD OSL の制御フローの分散を追跡するマスクのスタック**

1. オペランドと結果が均一かどうか**チェック**します。
2. 型 (均一または可変) に対応するオペランド値を**ロード**します。
3. IR を**出力**して操作を実行するか適切なプリコンパイル済みライブラリー関数を呼び出します。
4. 結果が可変の場合、格納する前に結果を**拡張**します。

andor 関数はシンプルで、均一化するオペランドは 1 つのみのため、SIMD LLVM バックエンドでのサポートは複雑ではありません。しかし、複数のオペランドを含む **texture3d()** のような関数では、LLVM バックエンドでのサポートは複雑になります。**texture3d()** 関数は、3D 座標 **p** でインデックスされたボリューム・テクスチャーの 3D ルックアップを実行します[6]。**texture3d()** を呼び出すと、関数はオプションのセットを想定します。その一部は可変 (**blur**、**width**、およびテクスチャー座標 **p**)、一部は均一 (**wrap** など) になると想定されます。ともに実行できる、同じ引数設定のレーンを最初にスキャンします。引数が一致しない残りのレーンはオフにされます。制御フローの説明は**図 7** を参照してください。



7 **SIMD OSL の `texture3d()` で LLVM レーンをマスキング / フィルタリング**

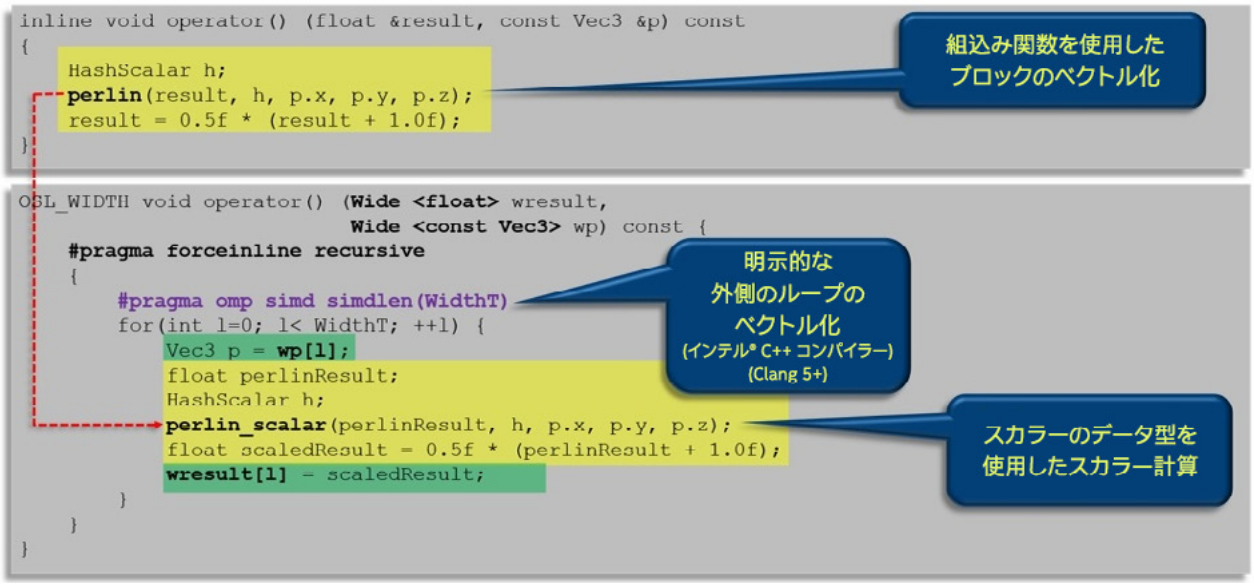
SIMD OSL ループと制御フローの LLVM

OSL ライブラリー関数で、アクティブなレーンを検出し、レーンの使用率に応じて異なる方法で関数呼び出し本体を実装します。例えば、パーリンノイズ関数 (詳細は後述) では、アクティブなレーンの数が 4 未満の場合、デフォルトのブロックでベクトル化されたパーリンノイズ実装を使用します。各レーンを個別に処理してベクトル化できます。要約すると、SIMD OSL バックエンドに LLVM を利用すると、レーンの内部およびレーン間の両方で方向を変更して、異なるベクトルの範囲を指定できます。

SIMD OSL のパーリンノイズ

非 SIMD OSL のパーリンノイズのオリジナルバージョンは、インテル® SSE 組込み関数を使用して、アルゴリズム内のブロックのベクトルを実行するように最適化されています (**図 8**)。オリジナルのパーリンノイズのパフォーマンスを保持したまま外側のループのベクトル化を可能にするため、インテル® SSE 組込み関数を削除し、**`perlin_scalar`** ヘルパーを作成することによりアルゴリズムのオリジナル C++ バージョンに戻しました。次に、基本的な SOA データレイアウトのデータ型のインポートおよびエクスポートにワイドなアクセサーを活用しました。外側のループのベクトル化は OpenMP* #pragma で実装され、SIMD 幅を指定しています。ループの内側で、現在のレーンのデータをエクスポートし、**`perlin_scalar`** でスカラー計算を行い、レーンの結果をインポートします。実際の **`perlin_scalar`** 計算はデータ形式や外側の SIMD ループを意識しないことに注意してください。すべてインライン展開されると、コンパイラーは複数のターゲット

ISA (インテル® SSE2、インテル® AVX、インテル® AVX2、インテル® AVX-512、その他) 向けに理想的なコードを生成できます。インテル® C++ コンパイラーで適切なインライン展開が行われるように、**`#pragma forceinline recursive`** を使用しています。



8 **OSL のパーリンノイズで SIMD を有効化**

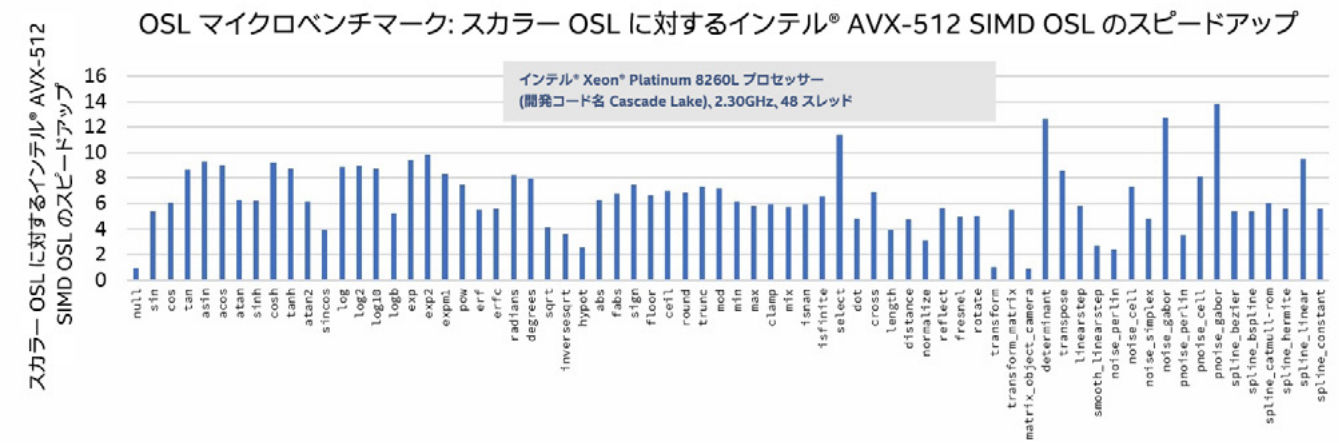
インテル® Xeon® プロセッサーにおける SIMD OSL のパフォーマンス

2 ソケットのインテル® Xeon® プロセッサー上で OSL ベンチマークと個々の OSL シェーダーのパフォーマンスを評価しました。

- インテル® Xeon® Gold 6248 プロセッサー (2.50GHz、2 ソケット、40 コア)
- インテル® Xeon® Platinum 8260L プロセッサー (2.30GHz、2 ソケット、48 コア、インテル® ハイパースレッディング・テクノロジー無効)

OSL は、シェーダーおよびシェーダーグループを扱うテストハーネス testshade で実行されました。testshade はレンダラーのシェーディング・モジュールの代わりと見なすこともできます。testshade はシングルスレッドまたはマルチスレッドで実行できます。次のベンチマークとシェーダーを使用して、SIMD OSL の優れたパフォーマンスを紹介します。

- 重要な OSL 関数を含む**マイクロベンチマークのスイート**
- 異なるテクスチャーとパターンを表す**個々のシェーダーのセット**



9 **インテル® Xeon® Platinum 8260L プロセッサー (2.30GHz) における SIMD OSL マイクロベンチマーク**

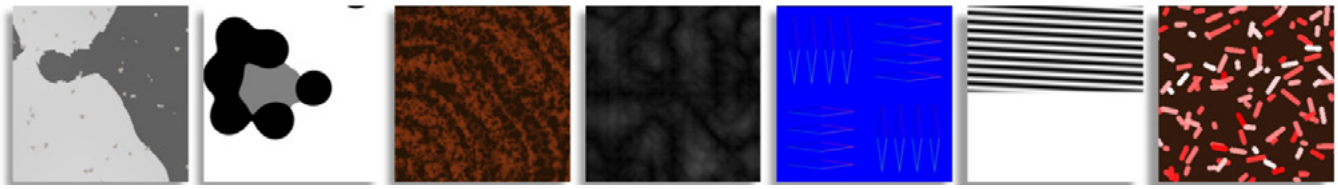
インテル® Xeon® Platinum 8260L プロセッサーにおける SIMD OSL マイクロベンチマークのパフォーマンス

OSL マイクロベンチマーク・スイートは、OSL 関数ファミリー (文字列、ノイズ、数学、三角法、論理演算、二項演算、スプライン、その他) の個々の関数を含みます。**図 9** は、バッチサイズ 16 で 48 スレッドを使用した場合のスカラー OSL に対するインテル® AVX-512 SIMD OSL のスピードアップを示しています。

マイクロベンチマークの 67 の関数のスピードアップは平均 7 倍で、最大のスピードアップは 13.8 倍 (Gabor ノイズ) でした。

インテル® Xeon® Gold 6248 プロセッサー上の個々のシェーダーのシングルポイント・スカラー OSL に対する SIMD OSL のスピードアップ

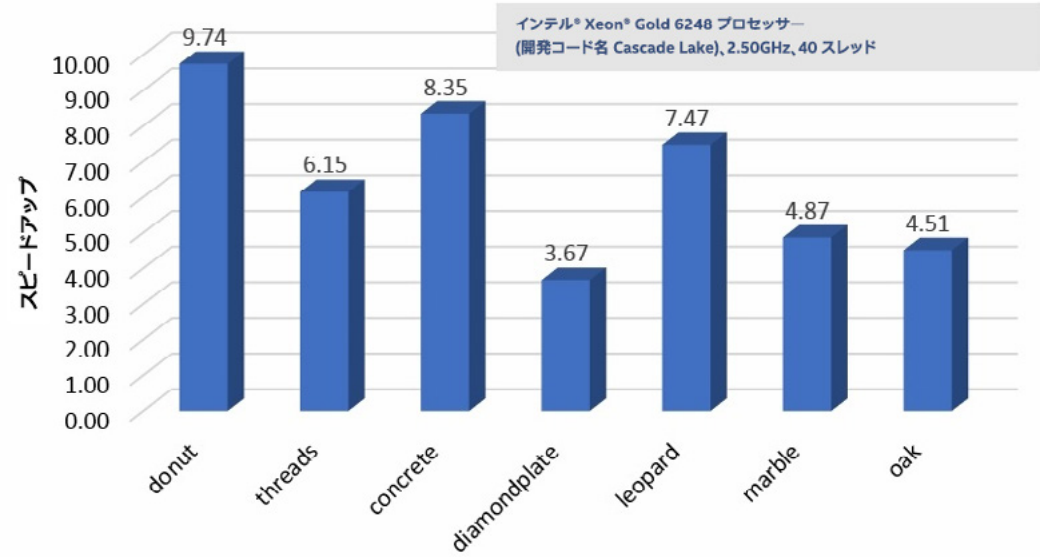
インテル® Xeon® Gold 6248 プロセッサー (2.50GHz) 上で、インテル® AVX-512 SIMD OSL、インテル® AVX2 SIMD OSL、およびスカラー OSL を使用してオープンソースのシェーダーのセットを評価しました。**図 10** の各シェーダー (marble[7]、concrete[10]、diamond plate[11]、donut[12]、leopard[13]、oak[14]、threads[15]) は、個々のテクスチャーを表します。各シェーダーは、複雑さとタイプ、および使用する OSL 関数の量が異なります。例えば、thread、marble、oak シェーダーの制御フローは、分岐が 1 つ以下、単一入力、単一出力と非常に単純です。一方、concrete や leopard のようなシェーダーの制御フローは複雑で多くの分岐が含まれます。



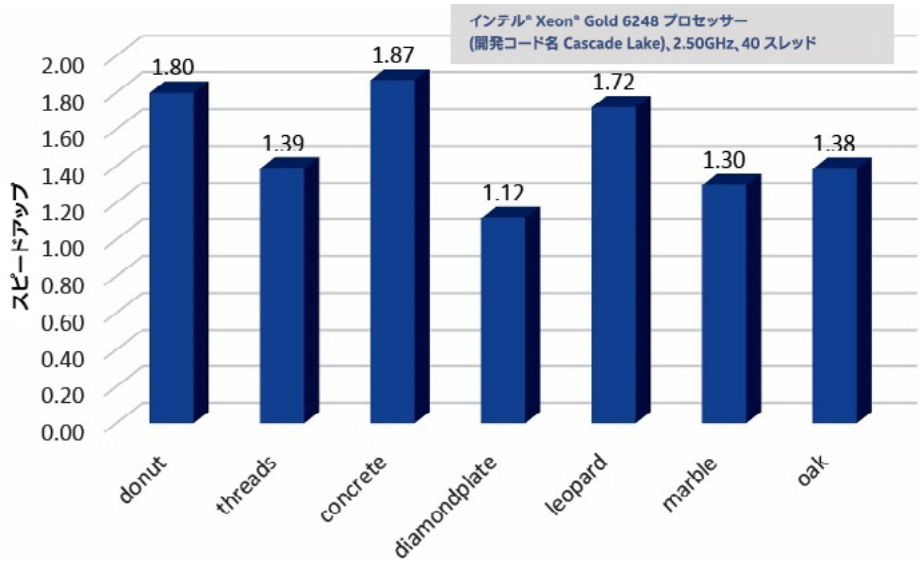
10 **OSL のシェーダー。左から右に、concrete、leopard、oak、marble、diamondplate、threads、donut**

最初に、SIMD OSL を使用して OSL シェーダーのスピードアップを測定し、スカラーのシングルポイント OSL に対するパフォーマンスを比較しました (**図 11**)。最もスピードアップしたのは donut シェーダーの 9.74 倍で、ほかのシェーダーは 3.67 倍から 8.35 倍でした。

次に、インテル® AVX2 SIMD OSL に対するインテル® AVX-512 SIMD OSL のスピードアップを測定しました (**図 12**)。すべてのシェーダーで、インテル® AVX-512 が提供するワイドなバッチサイズを使用した恩恵が得られました。



11 **インテル® Xeon® Gold 6248 プロセッサー (2.50GHz) 上の個々の OSL シェーダーのスカラー OSL に対する SIMD OSL のスピードアップ**



12 **インテル® Xeon® Gold 6248 プロセッサー (2.50GHz) 上の個々の OSL シェーダーのインテル® AVX2 SIMD OSL に対するインテル® AVX-512 SIMD OSL のスピードアップ**

オープン・シェーディング言語を加速

インテルは、2016 年から OSL のアーキテクチャーの再構築に取り組んでいます。この取り組みは次の 2 つのステップに要約できます。

- レンダリング時間最適化中の (JIT コンパイルでの) **ベクトル LLVM IR の生成**
- OSL のデフォルトのシングルポイント・インターフェイスへの**バッチ・インターフェイスの追加**

SIMD OSL は、Pixar の RenderMan* のような物理ベースのレンダラーで多くの利点を生み出しました。最近リリースされた SIMD OSL を含む RenderMan* 22.5 は、インテル® Xeon® スケーラブル・プロセッサーでインテル® AVX-512 を利用した場合、フル・レンダリングが最大 2 倍高速化、平均スピードアップが 15% になります[3]。

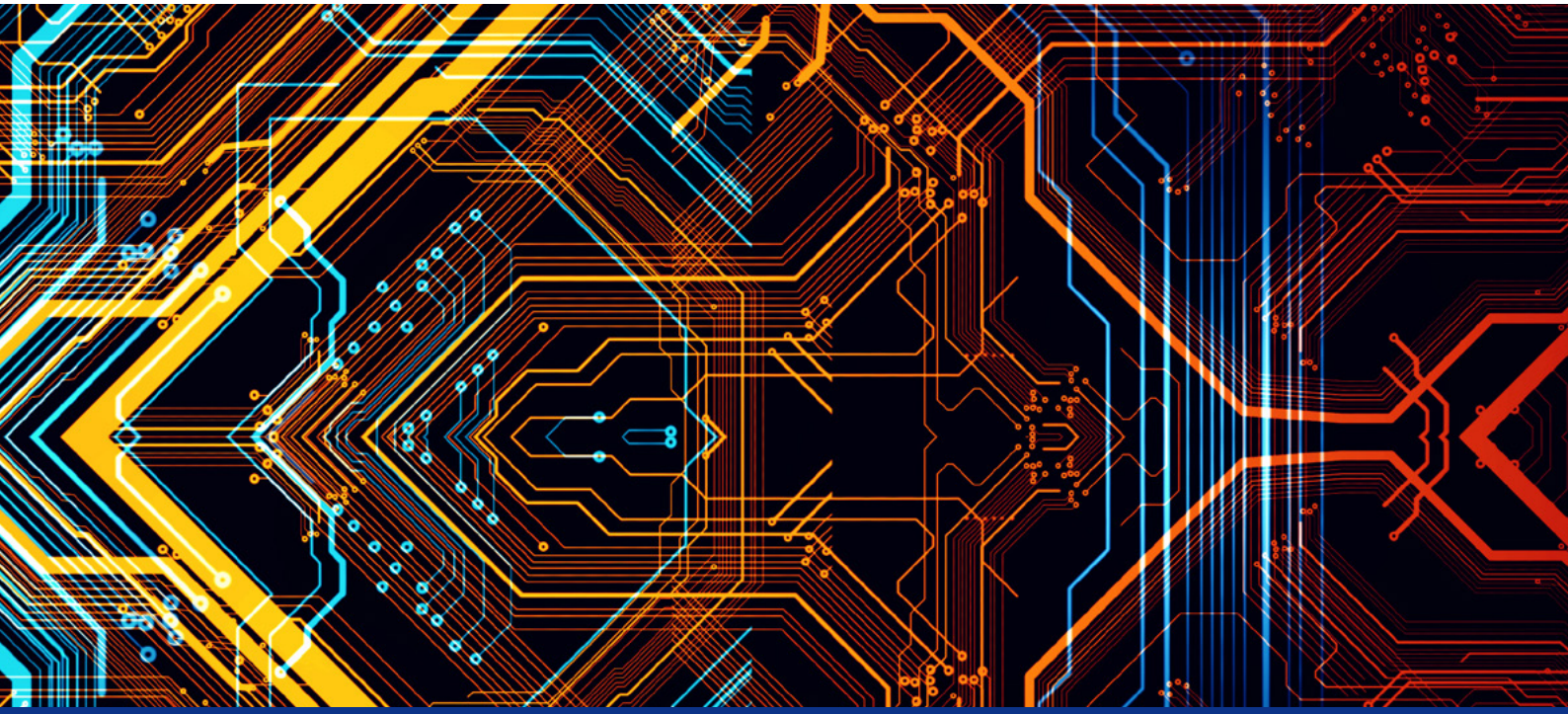
参考資料 (英語)

1. **Open Shading Language Sci Tech Award in 2017**
2. **Open Shading Language Repository**
3. **Pixar Animation Studios Releases RenderMan* 22.5**
4. **RenderMan*: What's New**
5. **FMX 2019**
6. **Open Shading Language Specification**
7. **Marble Shader**
8. **OSL Talk at SIGGRAPH 2018**
9. **SIMD Data Layout Templates (SDLT)**
10. **Concrete.osl**
11. **DiamondPlate.osl**
12. **TheDonutShader.osl**
13. **Leopard.osl**
14. **Oak.osl**
15. **Threads.osl**

インテル® C++ コンパイラー
ビルトインの生産性とパフォーマンス
詳細

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。

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 (software.intel.com/articles/optimization-notice#opt-jp) を参照してください。
Intel、インテル、Intel ロゴは、アメリカ合衆国および / またはその他の国における Intel Corporation またはその子会社の商標です。
* 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。
© Intel Corporation



パフォーマンスの最適化と生産性 (PoP) プロジェクト

終わりのないパフォーマンスの追求

Mike Croucher Numerical Algorithms Group (NAG) Developer Advocate

長い間、パフォーマンスの向上は並列処理により行われてきました。ベクトル化、スレッド、MPI、OpenMP*、GPU、FPGA、その他の多くのハードウェアおよびソフトウェア・テクノロジーが、ユーザーが求めるパフォーマンスを提供しており、テクノロジーを選択し、コードを最適化することで、ユーザーが求める大幅なスピードアップを実現できます。成功事例が多く寄せられ、実現可能と考えられるより大きく高度な問題を解くためコミュニティーが製品を使用していることに満足し、すべてが順調です。

しかし、パフォーマンスの追求に終わりはありません。まもなく、ユーザーからさらなるスピードアップを求められるでしょう。ユーザーが構築しているモデルは、さらに規模が大きく複雑になっています。モデルを実行するハードウェアには、新しいベクトル化機能が追加され、コア数も多くなっています。コードベースは膨大で、予算には限りがあり、容易に適用できる最適化はすべて適用済みです。

このような状況では、どこから開発を始めればよいのでしょうか。

PoP プロジェクト

パフォーマンスの最適化と生産性 (PoP) プロジェクト (英語) は、欧州連合が出資し、並列ソフトウェアのパフォーマンスを向上するため、次のような作業を行っている国際的なパートナーのグループです。

- 並列パフォーマンスを理解するための**一般的な手法の開発**
- PoP 手法を適用するための**オープンソースのツールの開発**
- PoP エキスパートが教育機関および業界団体のクライアントのコードを監査してリファクタリングすることにより、これらの開発を紹介する**詳細なケーススタディーのセットの作成** (EU 内のクライアントは無料で利用可能)

PoP 手法は広範な並列化手法とプログラミング言語に適用できます。Fortran、C、および C++ では OpenMP* と MPI が最もポピュラーですが、PoP は MATLAB*、Python*、および Perl などで記述されたアプリケーションも対象です。

PoP 手法

スケーリング・テスト、プロファイル、トレースなどのアプリケーションに関する情報を収集する方法はいくつかあり、**インテル® VTune™ Amplifier** のような製品や PoP パートナーにより開発されたオープンソースのツールが使用されています。これらは命令カウンターからキャッシュミスまですべてを含む、調査すべき膨大な量のデータを生成します。この膨大な情報から、開発者がコードを改善する最適な方向性を判断するのに役立つヒントを見つけ出すことは容易ではありません。

PoP 手法は、この膨大なデータから必要な情報を抜き出して、並列化に固有の異なる要因の相対的な影響を測定する**小さなメトリックの階層** (英語) を作成します。各メトリックは 0 から 1 までの効率の基準で、1 に近いほど効率が優れています。通常、PoP は、0.8 未満のものをすべて、注意が必要な項目と見なします。

これらのメトリックを理解するには次のケーススタディーが役立つでしょう。

PoP メトリックと zCFD

PoP パートナーの 1 つ、**Numerical Algorithms Group (NAG)** (英語) は最近、Zenotech により開発された商用計算流体力学ソルバー **zCFD** (英語) のパフォーマンスの最適化に取り組みました。インテル® VTune™ Amplifier のデータから PoP メトリックを生成してオリジナルの開発者と協力することにより、特定のシミュレーションの実行時間を 3 分の 1 に短縮しました。

最初のステップは、インテル® VTune™ Amplifier データの収集を対象領域 (ROI) のみに制限することでした。zCFD は、C++ で記述された計算カーネルを呼び出す Python* パッケージ (**zCFD-driver** (英語)) を使用しています。そのため、チームは **インテル® VTune™ Amplifier のインストルメントとトレース・テクノロジー (ITT) API の Python* バインディング** (英語) を使用して ROI 外のトレースを無効にしました。

さまざまなコア数でのシミュレーション実行でインテル® VTune™ Amplifier データを収集できるようになると、PoP メトリックの最初のセットが計算できるようになりました (**表 1**)。(インテル® VTune™ Amplifier データから PoP メトリックを計算する方法については、この記事の本題から外れるため取り上げません。詳細は、**このケーススタディーの PoP ウェビナー** (英語) を参照してください。別の方法は、**Scalasca を使用した PoP メトリックの自動計算** (英語) で紹介されています。)

表 1. PoP メトリック

スレッド数	1	2	4	6	8	10	12
グローバル効率	0.97	0.71	0.66	0.52	0.55	0.49	0.33
並列処理効率	0.97	0.80	0.77	0.64	0.67	0.60	0.50
計算効率	1.00	0.89	0.85	0.82	0.82	0.82	0.66

グローバル効率は、並列処理効率と計算効率の積です。並列処理効率は、コードの並列化が実行時間に与える影響を測定します。次のような要因の影響を含みます。

- **いかにバランスよく**計算ロードがスレッド間で配分されているか
- **どのくらいの時間**が並列処理のオーバーヘッドにより失われているか

この効率は、スレッドが有用な計算で費やした時間の平均と、アプリケーションの合計実行時間の比率として計算されます。

計算効率は、アプリケーションの計算ロードがスレッド数で適切にスケールされているかを示します。この効率は、すべてのスレッドのコードが計算で費やした合計時間と、シリアルコードが計算で費やした時間の比率です。

グローバル効率は、スレッド数が 2 つ以上になるとしきい値 (0.8) を下回っています。このほとんどは並列処理効率の対応する値の低下によるものです。計算効率は、12 スレッドを除いてしきい値を下回っていません。

つまり、これらの効率は、各スレッドが実行する計算ではなく、ワークをスレッド間に分割する方法に改良の可能性があることを示しています。例えば、10 スレッドでは、並列処理の効率 0.56 は並列化問題に対処すると、実行時間を 44% 改善できる余地がありますが、計算効率 0.82 は計算に関連する問題に対処しても、実行時間には 18% しか改善の余地がありません。とは言え、12 コアの計算効率は非常に不可解です。

並列処理の効率

並列処理の効率に注目することが最も適切であると理解できたところで、この効率が低い理由を詳しく調べることにしましょう。インテル® VTune™ Amplifier から得られる単純なメトリックは、コードのシリアル領域で費やされた実行時間の割合です (**表 2**)。

表 2. シリアルコードの実行時間

スレッド数	1	2	4	6	8	10	12
シリアル実行時間の割合	—	88.6	84.6	75.0	74.2	70.1	66.6

12 コアに達した時点で、実行時間の 33% がシリアルコード領域で費やされています。詳細な調査により、開発者が並列化しようとしたが、実際にはまだシーケンシャルに実行している領域があることが判明しました。コードをリファクタリングして、この部分を修正しました。

ロードバランスの効率 (**表 3**) は、ワークがスレッド間で均等に分割されていないことを示しています。

表 3. ロードバランスの効率

スレッド数	1	2	4	6	8	10	12
ロードバランスの効率	1.00	0.88	0.89	0.85	0.89	0.86	0.85

調査により、**pow()** 関数を呼び出しているコード領域でロード・インバランスが発生していることが判明しました。原因は **slow コードパス** (英語) でした。基数と指数の両方が 1 に近いため、**pow()** は結果が高精度になるように時間をかけて計算を行っていました。しかし、それほどの精度はこの計算には必要ありません。この問題は、基数をスケーリングして累乗を求めた後、再度スケーリングを行うことで解決しました。

```
double pow(double a, double b) {
    a *= 5.0;
    tmp = pow(a,b);
    return tmp/pow(5.0,b);
}
```

pow() の 2 つの呼び出しはベクトル化を使用して同時に計算できるため、この変更により発生したコストは 1 つの除算のみです。

計算効率

メトリックから、このケースでは計算効率は並列処理の効率ほど重要でないことが示されましたが、12 コアの計算効率が非常に奇妙であることも示されました。詳しく見ることにしましょう。容易に修正できる単純な問題であることを期待していましたが、幸いにもその望みはかなえられたようです。

計算効率は次の 3 つのサブメトリックから構成されます (**表 4**)。

1. サイクルあたりの命令数 (IPC) の効率
2. 命令数の効率
3. CPU 周波数の効率

命令数の効率は、基準ケース (1 プロセッサー) の有用な命令数の合計とプロセス数を増やした場合の値の比率です。命令数の効率が減少すると、計算問題を解くために必要な命令数の合計が増加します。

IPC 効率は、IPC とリファレンスの比較で、低い値は計算レートが遅くなったことを示します。この典型的な要因はキャッシュヒット率の低下およびメモリー帯域幅不足により、プロセスがストールしてデータを待つためです。

CPU 周波数効率は、スレッド数が増えたときにクロックの速度がどのように変化するかを示します。

表 4. 計算効率を構成するサブメトリック

スレッド数	1	2	4	6	8	10	12
IPC 効率	1.00	0.94	0.93	0.92	0.91	0.90	0.91
命令数効率	1.00	1.00	1.00	1.00	1.00	1.00	1.00
CPU 周波数効率	1.00	0.94	0.91	0.89	0.90	0.91	0.72

IPC 効率と命令数の効率に注目すべき点はありませんが、CPU 周波数効率は 10 コアから 12 コアで急激に低下しています。

Zenotech は、調査に使用したマシンでは CPU 周波数ガバナーがデフォルトでオンデマンドに設定されていることが周波数が急激に低下した原因であることを突き止めました。そこで Slurm* コマンドに **`--cpu-freq=performance`** を追加し、コアがすべて利用されている場合でもベース周波数で実行するように CPU に指示することにより、この問題を解決しました。

結果

これらのメトリックを参考にして、zCFD の開発者は上記のようにコードと計算環境に変更を加えました (一部の変更はスペースの関係で含まれていません)。新しいコードでメトリックを再計算した結果を**表 5** に示します。

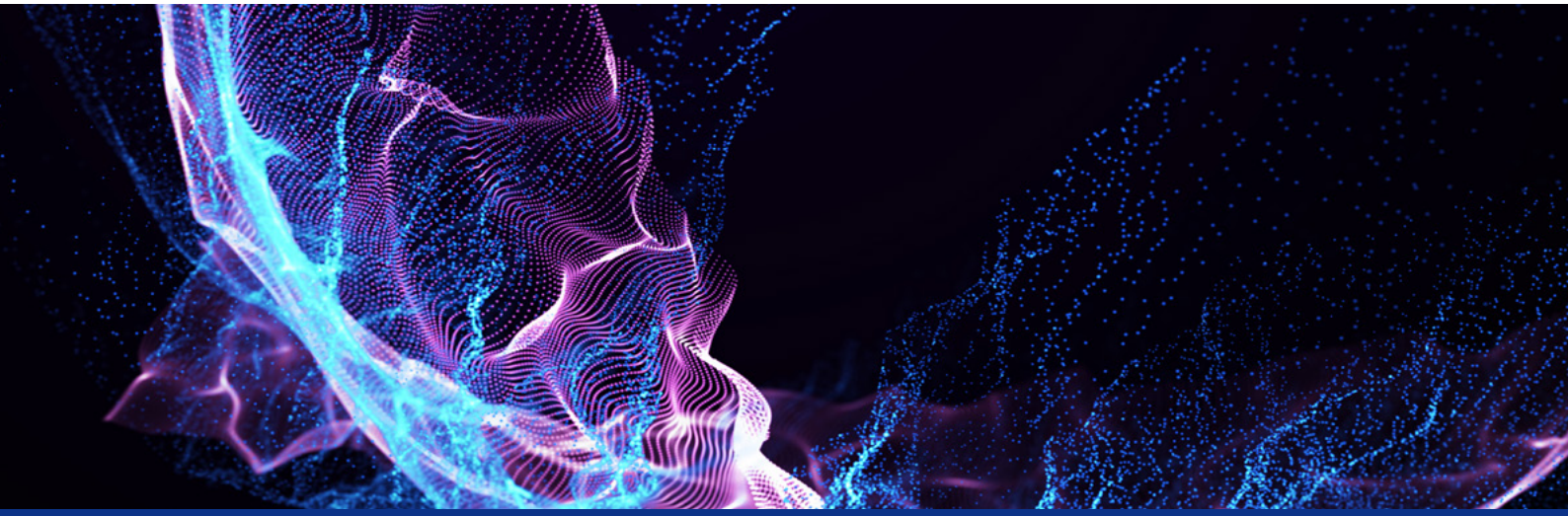
表 5. 効率

スレッド数	1	2	6	12
グローバル効率	1.00	0.89	0.73	0.56
並列処理効率	1.00	0.98	0.89	0.76
計算効率	1.00	0.91	0.82	0.74

表 5 を**表 1** と比較すると、全面的に効率が改善されたことが分かります。Zenotech がはるかに大きな問題で新しいコードを実行したところ、オリジナルのコードと比較して最大 3 倍のスピードアップが達成されました。しかし、この成功の後も、メトリックは向上の余地がまだ残されていることを示しています。パフォーマンスの追求はまだ続いています。

PoP コード監査の申し込み

PoP プロジェクトでは、すべての分野の教育機関および業界団体のコード向けに、パフォーマンスの最適化と生産性サービスを提供しています。ユーザーの並列ソフトウェアの最適化およびパフォーマンス問題の理解を支援することを目的とした、さまざまなサービスが用意されています。EU 内の教育機関、研究機関、営利団体は、サービスを無料で利用できます。サービスの申し込みは **Web サイト**で受け付けています。



インテル® ソフトウェアへの投資が HPC ソフトウェア開発者にもたらす 7 つのメリット

インテルおよび HPC ソフトウェアを見直す

James Reinders The Parallel Universe 名誉編集長

インテルは HPC ソフトウェアのサポートに精力的に取り組んできましたが、そのサポート体制はこの数年で大きく変化しました。この記事では、インテルのソフトウェア・チームの変化について簡単に取り上げます。

HPC は、ベクトル・スーパーコンピューター、シストリック・アレイ、ハイパーキューブの黎明期を思い起こさせる、多様性に富んだハードウェアの新しい黄金時代の夜明けを迎えています。これまでとの大きな違いは、膨大なインストール・ベースのミッション・クリティカルな科学およびエンジニアリング・ソフトウェアが存在することです。HPC における成功は、ハードウェアをソフトウェアのニーズに適応させることです。

では、インテルが HPC ソフトウェア開発者にもたらす 7 つのメリットを説明しましょう。

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。

1 コミュニティーのサポート : コミュニティーによるコードのパフォーマンスの向上および学習機会の提供

インテルは、**インテル® Parallel Computing Centers (IPCC)** (英語) および**インテル® Modern Code** (英語) を設立して、HPC ソフトウェアが新しいハードウェアを活用できるように支援しています。IPCC は、HPC で使用されるさまざまなオープンソース・アプリケーションの変更に貢献しています。インテル® Modern Code の **Web サイト** (英語) では、基本的な手法を学習するための多くのオンラインリソース (インタビュー、トレーニング、ホワイトペーパー、その他) を提供しています。

IPCC の学習ポイントは、非常に読みやすい 2 冊の書籍として出版されています (『**High-Performance Parallelism Pearls**』(英語))。これらの書籍では、並列化、ベクトル化、アルゴリズムを使用してコードを現代化するための手法を詳細に説明しています。これらの情報はすべて、この号の注目記事で取り上げている第 2 世代**インテル® Xeon® スケーラブル・プロセッサー** (開発コード名 Cascade Lake) 向けの開発を行っているすべての開発者に役立つでしょう。

2 詳細なプログラム解析ツール : エキスパートによるアプリケーションとシステムのチューニングを支援

アプリケーションを実行するときに何が実際に起きているか理解できるほど価値のあることはありません。インテルのツールは、最大スピードアップの予測、ボトルネックの特定、並列プログラミング問題の指摘 (潜在的なデータ競合およびデッドロックをピンポイントで指摘) によりチューニングを支援します。インテル® VTune™ Amplifier は、数あるプロファイル・ツールの中で伝説となっています。最新のツールには、ルーフライン解析、アプリケーション・パフォーマンス・スナップショット、ストレージ・パフォーマンス・スナップショット、MPI 通信解析 (**インテル® Trace Analyzer & Collector**)、Python* と Java* が混在したコンパイル済みコードのプロファイルなどの機能が含まれています。アプリケーション・パフォーマンス・スナップショットは最初に利用する機能として優れていますが、パフォーマンスを追求するうちに、さらに多くのことが知りたくなるでしょう。**インテル® VTune™ Amplifier** でチューニングを開始した後、必要に応じて**インテル® Inspector** や**インテル® Advisor** を使用することを推奨します。

3 高度にチューニングされたライブラリー : 追加してプログラムを高速に実行

ベンダーにより最適化されたライブラリーを使用してプログラムを高速に実行することほど簡単なことはありません。**インテル® マス・カーネル・ライブラリー (インテル® MKL)** は、以前から、高度にチューニングされた BLAS、FFT、およびソルバーを提供しています。現在、**インテル® パフォーマンス・ライブラリー** (英語) のスイートは、すべての HPC 開発者にとって必携のツールキットです。

- **インテル® MKL:** 演算処理ルーチンを高速化し、アプリケーションのパフォーマンスを向上して、開発期間を短縮します。
- **インテル® MPI ライブラリー :** インテル® アーキテクチャー上で、柔軟で、効率良い、スケーラブルなクラスターメッセージ処理を実現します。
- **インテル® スレッディング・ビルディング・ブロック (インテル® TBB):** 高速で、スケーラブルな並列アプリケーションを作成するための高度なスレッド化を提供します。
- **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):** 画像、ビジョン、信号、セキュリティー、ストレージ・アプリケーションのパフォーマンスを高速化します。
- **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):** 使いやすいライブラリーによりマシンラーニングとデータ解析のパフォーマンスを向上します。

4 最適化コンパイラー : コンパイルしてプログラムを高速に実行

開発プロセスでインテルのコンパイラーを利用することで、より速く効率的に動作するアプリケーションを作成できます。このツールは、インテル® プロセッサーの増え続けるコア数とベクトルレジスターの幅を活用する最適化されたコードを生成します。コンパイラーは一般的な開発環境に統合して利用でき、Microsoft* Visual C++* (Windows*) や GNU* コンパイラー (Linux* および macOS*) など、ほかのコンパイラーと互換性があります。詳細は、**こちら**を参照してください。

5 ソフトウェア定義の可視化

HPC の多くの可視化作業は CPU で行われることを、これまで何度も実感してきました。インテルは、ハイパフォーマンスな科学的な可視化のサポートに多額の投資を行っています。データ移動が最小化されるため、これらのワーク (および洗練されたレンダリング・タスク) は本来、CPU レンダリングが適しています。また、CPU のレンダリング・パイプラインはハードウェアに組込まれていません。

レンダリング・フレームワークの詳細は**こちら** (英語) を、一般的なソフトウェア定義の可視化の詳細は **http://sdvis.org/** (英語) をそれぞれ参照してください。

6 インテルにより最適化されたディープラーニング・フレームワークと Python* によるハイパフォーマンス AI

インテルは、ハイパフォーマンスなトレーニングおよび推論を行うため、TensorFlow* や PyTorch* などの**一般的なディープラーニング・フレームワーク** (英語) の最適化に投資しています。詳細は、**インテル® AI Developer Program** (英語) を参照してください。インテルは、コードを変更することなく科学計算コードを高速に実行できるように、(NumPy*、SciPy*、scikit-learn に注目して) Python* の最適化も行っています。インストール方法は、**こちら**を参照してください。

7 クロスアーキテクチャー・ツール : OpenVINO™ ツールキットと一度コードを書けば各種ハードウェアで高速に実行できるように支援するビジョン

一度コードを書けばどこでも実行できるという概念は新しいものではありませんが、そのソリューションはすべて、少なくともいくつかのプラットフォームでパフォーマンス・ペナルティーが発生します。インテルは、**OpenVINO™ ツールキット** (Open Visual Inference and Neural network Optimization—オープン・ビジュアル・インファレンスとニューラル・ネットワークの最適化—の略) として知られるオープンソース・プロジェクトを開始しました。このプロジェクトは、CPU、GPU、FPGA、VPU と、さまざまなハードウェアを対象としています。多くのアーキテクチャーでニューラル・ネットワークをポータブルに実行する方法を探している人には、一見の価値があります。

今のところ、それほど注目されていませんが、インテルは、**インテル® Architecture Day 2018** (英語) で、「no transistor left behind (ハードウェアを意識する必要がない)」を目標とした **oneAPI プロジェクト**のビジョンを発表しました。このビジョンは、現時点で市場に存在し、我々の役に立っている、HPC ソフトウェアのサポートへの精力的な取り組みを論理的に延長したものと言えます。

インテルと HPC ソフトウェア開発

多くの点で、インテルと HPC はともに成長を続け、どちらも非常に多様で複雑になりました。インテルのソフトウェアへの投資により、HPC ソフトウェア開発者を支援する方法は拡大を続けています。この記事では、インテルの支援により、生産性を高めるためにダウンロード、学習、使用できる 7 つの項目をリストしました。最後の項目からも分かるように、インテルの HPC コミュニティーへの貢献は今後も続くことでしょう。

Software

THE PARALLEL UNIVERSE

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、www.intel.com/benchmarks (英語) を参照してください。性能の測定結果は 2019 年 3 月 25 日時点のテストに基づいています。また、現在公開中のすべてのセキュリティー・アップデートが適用されているとは限りません。詳細については、公開されている構成情報を参照してください。絶対的なセキュリティーを提供できる製品はありません。* 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。

テストでは、特定のシステムでの個々のテストにおけるコンポーネントの性能を文書化しています。ハードウェア、ソフトウェア、システム構成などの違いにより、実際の性能は掲載された性能テストや評価とは異なる場合があります。購入を検討される場合は、ほかの情報も参考にして、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。

* パフォーマンスおよびベンチマークの結果に関する詳細は、www.intel.com/benchmarks (英語) を参照してください。

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。

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

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)* は、特定のプロセッサー演算で高いスループットを示します。プロセッサーの電力特性の変動により、AVX 命令を利用すると、a) 一部の部品が定格周波数未満で動作する、b) インテル® ターボ・ブースト・テクノロジー 2.0 を使用する一部の部品が任意または最大のターボ周波数に達しない可能性があります。

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・データまたはウェブサイト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ウェブサイトにアクセスし、本資料で参照しているデータが正確かどうかを確認してください。

本資料には、開発中の製品、サービスおよびプロセスについての情報が含まれています。本資料に含まれる情報は予告なく変更されることがあります。最新の予測、スケジュール、仕様、ロードマップについては、インテルの担当者までお問い合わせください。

本資料で説明されている製品およびサービスには、エラッタと呼ばれる設計上の不具合が含まれている可能性があり、公表されている仕様とは異なる動作をする場合があります。現在確認済みのエラッタについては、インテルまでお問い合わせください。

本資料で紹介されている資料番号付きのドキュメントや、インテルのその他の資料を入手するには、1-800-548-4725 (アメリカ合衆国) までご連絡いただくか、www.intel.com/design/literature.htm (英語) を参照してください。

© 2019 Intel Corporation. 無断での引用、転載を禁じます。Intel、インテル、Intel ロゴ、Intel Core、Intel Optane、Xeon、OpenVINO、VTune 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。

OpenCL および OpenCL ロゴは、Apple Inc. の商標であり、Khronos の使用許諾を受けて使用しています。

* 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/1909/PDF/XL/MKTG/SK