intel software

THE PARALLEL UNIVERSE

ダイレクト・メモリー・アクセスの先へ：インテル® データ・ストリーミング・アクセラレーターによるデータセンター・コストの削減

並列標準を洗練する：多様性、アライメント、相互作用

Issue 53
2023

目次

編集者からのメッセージ 3

注目記事

ダイレクト・メモリー・アクセスの先へ：インテル® データ・ストリーミング・アクセラレーターによるデータセンター・コストの削減 6
第 4 世代インテル® Xeon® スケーラブル・プロセッサーに搭載されているデータ変換のオンチップ・アクセラレーションを活用する

並列標準を洗練する：多様性、アライメント、相互作用 15
標準ベースの並列プログラミングの将来を見据える

堀のある AI から責任ある AI へ 19
生成 AI のリスクを最小限に抑える

独自のカスタム・チャットボットの作成 22
大規模な言語モデルをインテル・プロセッサーで迅速かつ簡単にトレーニングする

Hugging Face* と oneAPI を使用した Falcon の 70 億パラメーター・モデルの微調整 28
インテル® アドバンスト・マトリクス・エクステンション（インテル® AMX）を搭載したインテル® Xeon® プロセッサー上で大規模言語モデルを最適化する

Fortran の DO CONCURRENT を使用したアクセラレーター・オフロード 37
ヘテロジニアス・コンピューティングに対する標準 Fortran の制限

高帯域幅メモリーを搭載したインテル・プロセッサーのパフォーマンス最適化 42
インテル® Xeon® CPU マックス・シリーズのパフォーマンス・チューニングの詳細

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

オープン性、持続可能性（サステナビリティー）、そして生成 AI モデルのカンブリア爆発

前号の発行後に、2 つのニュースが AI コミュニティーに波紋を引き起こしました。1 つ目は、「生成 AI に立ちはだかるコンピューティングとサステナビリティーの課題に取り組む」に関するインテルと Hugging Face との談話です。この談話では、特に生成 AI の民主化が The Model Hub（英語）の事前トレーニング済みモデルの爆発的な増加につながった経緯について議論されました。2 つ目は、リークされた Google の内部メモ「We Have No Moat, And Neither Does OpenAI」（英語）です。このメモでは、小さく、速く、安価で、カスタマイズ可能なオープンソース AI にクローズドモデルは勝てないと分析されています。

本号では、これらのニュースに関連する記事を 3 つ取り上げています。1 つ目の Huma Abidi（AI ソフトウェア製品ジェネラル･マネージャー兼シニア･ディレクター）と Haihao Shen（AI ソフトウェア･アーキテクト）による寄稿記事「**堀のある AI から責任ある AI へ**」では、シリコンバレーで流行した表現を用いた AI における「堀」について言及します。2 つ目の「**独自のカスタム･チャットボットの作成**」では、オープンなモデルと簡単に入手できるハードウェアを使用して、カスタマイズされた高性能チャットボットを構築する方法を示します。3 つ目の「**Hugging Face* と oneAPI を使用した Falcon の 70 億パラメーター・モデルの微調整**」では、インテル® アドバンスト・マトリクス・エクステンション（インテル® AMX）を搭載したインテル® Xeon® プロセッサー上で別のオープンソースの大規模言語モデルを最適化する方法を紹介します。

注目記事「**ダイレクト・メモリー・アクセスの先へ : インテル® データ・ストリーミング・アクセラレーターによるデータセンター・コストの削減**」では、データ変換にオンチップ・アクセラレーションを活用するコード例とアドバイスを提供します。インテル® データ・ストリーミング・アクセラレーターは、第 4 世代インテル® Xeon® スケーラブル・プロセッサーの新しい機能です。

SYCL* の事例（The Parallel Universe 51 号）では、ヘテロジニアス･コンピューティングに関する ISO C++ の制限について説明しました。本号の 2 つ目の寄稿記事「**並列標準を洗練する**」では、John Pennycook（ソフトウェア･イネーブリング & 最適化アーキテクト）が将来の C++ 言語の概念で SYCL* を調整するプロセスを説明します。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

私が計算科学に携わった最初の 15 年間、数式（FORmula）をコードに変換（TRANslate）する必要がある場合は、ほとんどの作業を Fortran で行っていました。15 年間の Fortran と言えば、ロスアラモス国立研究所の最近のレポート「今後 15 年間にミッション・クリティカルなコードを Fortran に依存することのリスクの評価」（英語）は非常に興味深いものでした。ヘテロジニアス並列処理のサポートはリスク要因の 1 つであるため、Ron Green（コンパイラー・エンジニアリング・マネージャーで Fortran の愛好家）とともに、「**Fortran の DO CONCURRENT を使用したアクセラレーター・オフロード**」を執筆しました。この記事は言語の機能に注目したものです。現在、CPU と GPU 上の DO CONCURRENT のパフォーマンスを分析する続編の記事に取り組んでいます。ご期待ください。

最後に、インテル® Xeon® CPU マックス・シリーズでのパフォーマンス・チューニングの詳細を説明した記事「**高帯域幅メモリーを搭載したインテル・プロセッサーのパフォーマンス最適化**」で本号を締めくくります。

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、oneAPI を利用したヘテロジニアス並列コンピューティング向けのインテル・ソリューションの詳細は、Tech.Decoded（英語）を参照してください。

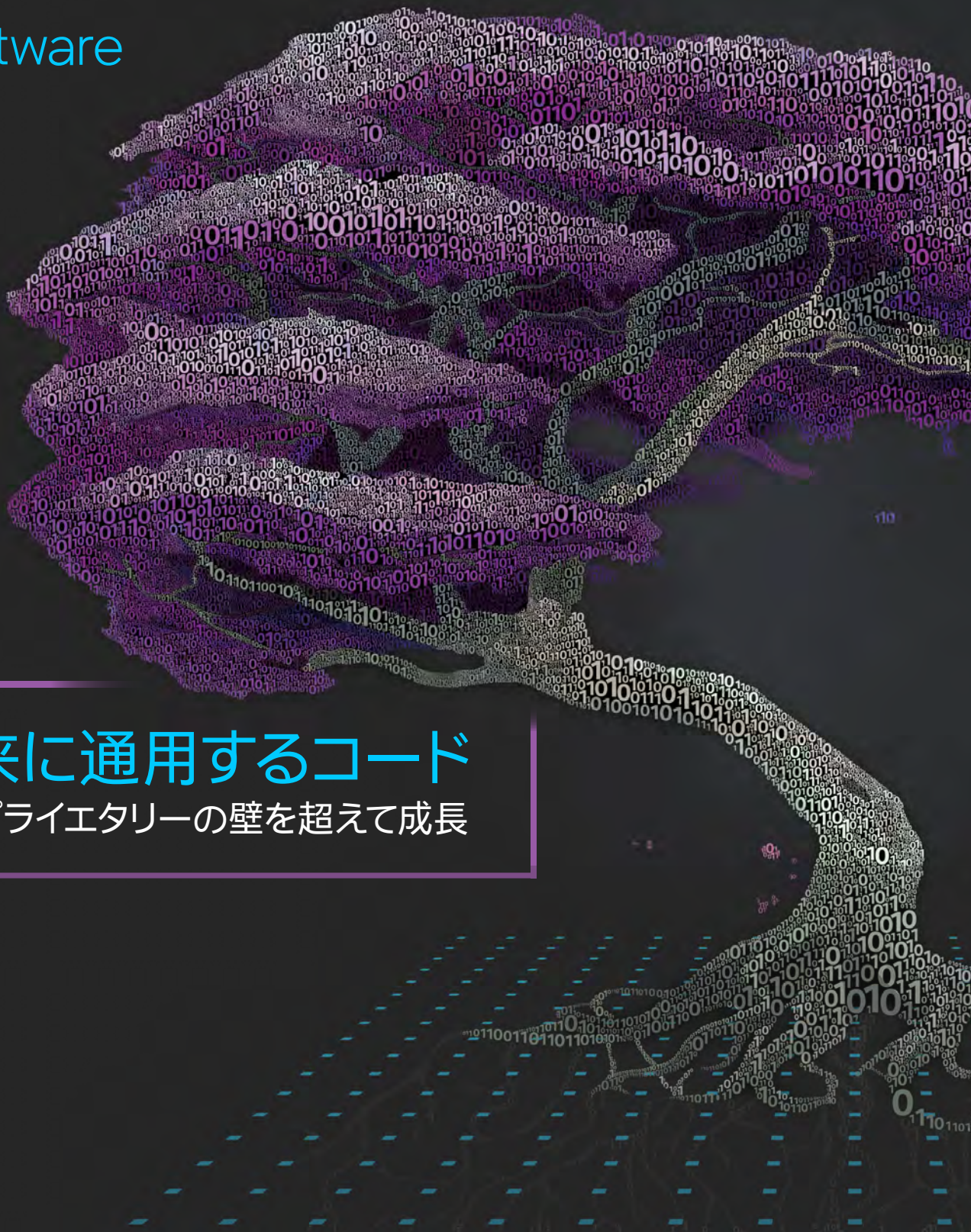
Henry A. Gabb

2023 年 7 月

ポッドキャスト
生成 AI は我々の成長、仕事、生活をどのように変えるのか
聴く（英語）

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

intel. software



将来に通用するコード

プロプライエタリーの壁を超えて成長

oneAPI

複数の言語をサポートし、ヘテロジニアス計算パフォーマンスを実現する、
単一のオープンなプログラミング・モデルで、コードの幅を広げましょう。

オープンスタンダードに根ざしたインテル® oneAPI は、クロスアーキテクチャーのライブラリー、
コンパイラー、ツールを提供し、コードをより多くのハードウェアに対応させ、
比類ないパフォーマンスを実現します。

インテル® oneAPI の詳細 →

ダイレクト・メモリー・アクセスの先へ：インテル® データ・ストリーミング・アクセラレーターによるデータセンター・コストの削減

第 4 世代インテル® Xeon® スケーラブル・プロセッサーに搭載されているデータ変換のオンチップ・アクセラレーションを活用する

Sanjay Kumar インテル コーポレーション 主席エンジニア
Reese Kuper 同 研究インターン
Atul Kwatra 同 フェロー
Shibani Nataraja 同 製品マネージャー
Narayan Ranganathan 同 主席エンジニア
Rajesh Sankaran 同 シニアフェロー
Nikhil Rao、Ren Wang、および Yifan Yuan 同 科学研究員

最新のデータセンターでは、`memcpy()`、`memmove()`、ハッシュ、圧縮などの負荷の高いメモリー操作と変換を含む多くのアプリケーションが利用されるようになっています。アプリケーションには、データベース、画像処理、ビデオ転送、グラフ処理などが含まれますが、これらに限定されるものではありません。このようなメモリー操作と変換はデータセンターのインフラストラクチャー・ソフトウェアでも一般的であり、大量の CPU サイクルを消費します。これらのサイクルはアプリケーションの実行に使用される可能性があるため、「データセンター・タックス」(英語)と呼ばれることもあります。理想的には、このような操作は最適化されたハードウェア・エンジンにオフロードするべきです。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

インテル® データ・ストリーミング・アクセラレーター

インテル® データ・ストリーミング・アクセラレーター（インテル® DSA）は、最新の第 4 世代インテル® Xeon® スケーラブル・プロセッサーに統合された、ハイパフォーマンスなデータコピーと変換のアクセラレーターであり、処理効率と実用性だけでなく、汎用性も提供します。インテル® DSA には、1 人以上のユーザーから送信されたワーク記述子を効率良く処理するハードウェア・コンポーネントが搭載されています。共有仮想メモリー（SVM）のサポートにより、これらのワーク記述子はユーザー・アプリケーションからメモリーマップ I/O（MMIO）レジスター経由でインテル® DSA に直接送信でき、ターゲットメモリー領域をオペレーティング・システムでピニングする必要はありません。ユーザーは、専用ドライバーで提供されるユーザー・インターフェイスを利用して、ニーズに基づいて計算リソースを直接構成することもできます。インテル® DSA は、新しくサポートされた操作により、さまざまな優れた機能を提供します（**表 1**）。多くのユーザー / カーネルのプロセスで、インテル® DSA が提供する機能を活用できることが期待されます。

タイプ	操作	説明
移動	メモリーコピー	ソースアドレスからディスネーション・アドレスへデータをコピー
	デュアルキャスト	ソースアドレスから 2 つの別のディスネーション・アドレスへデータを転送
	CRC 生成	転送されたソースデータの CRC チェックサムを作成
	データ整合性 フィールド (DIF) 操作	ソースアドレスからディスネーション・アドレスへ転送されたソースデータの 各 512/520/4096/4104 バイト・ブロックの DIF を確認、挿入、ストリップ、または更新
フィル	メモリーフィル	8/16 バイトの固定パターンで指定されたメモリー領域を連続してフィル
比較	メモリー比較	2 つのソースバッファーを相互に比較して同一かどうかをリターン
	パターン比較	ソースバッファーを 8 バイトのパターンと比較して領域全体がパターンと一致するかどうかをリターン
	デルタレコードの作成	2 つのソースバッファー (元のバッファーと変更されたバッファー) 間の差を含むデルタレコードを作成
	デルタレコードの適用	デルタレコードを元のバッファーにマージして変更されたバッファーのコピーをディスネーション・アドレスに作成
フラッシュ	キャッシュフラッシュ	キャッシュ階層から指定されたアドレス範囲内のすべてのキャッシュラインを退避

表 1. インテル® DSA でサポートされているデータ・ストリーミング操作

インテル® Data Accelerator Driver（IDXD）は、デバイスの初期化と管理を行うカーネル・モード・ドライバーであり、インテル® DSA の検出、初期化、構成の機能を提供します。アプリケーションは、これらの制御に、ユーザー空間 `libaccel-config` API ライブラリーを利用できます。データパスについては、アプリケーションのインテル® DSA インスタンスに低レイテンシー・アクセスを提供するため、IDXD は `mmap()` を介して MMIO ポータルを char デバイスとして利用できるようにします。これらの新機能と改良により、さまざまな分野（**表 2**）、特に中規模から大規模のデータサイズの操作（**図 1**）でインテル® DSA のパフォーマンスが同等のソフトウェアよりも大幅に向上していることが分かります。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

分野	アプリケーションと操作の例
ネットワーク・スタックの高速化	DPDK、ソフトウェア仮想スイッチ、ビデオ転送
ストレージスタックの高速化	SPDK、NVMe-oF、DAOS
データセンター・コストの削減	VM の起動 / 移行、メモリー・コンパクション
HPC/ML の高速化	libfabric/MPI でのメモリー移動 / ゼロ化
ヘテロジニアス・メモリー管理の高速化	CXL/Pmem ベースの階層型メモリーシステムのページ移行

表 2. さまざまな分野におけるインテル® DSA の活用例

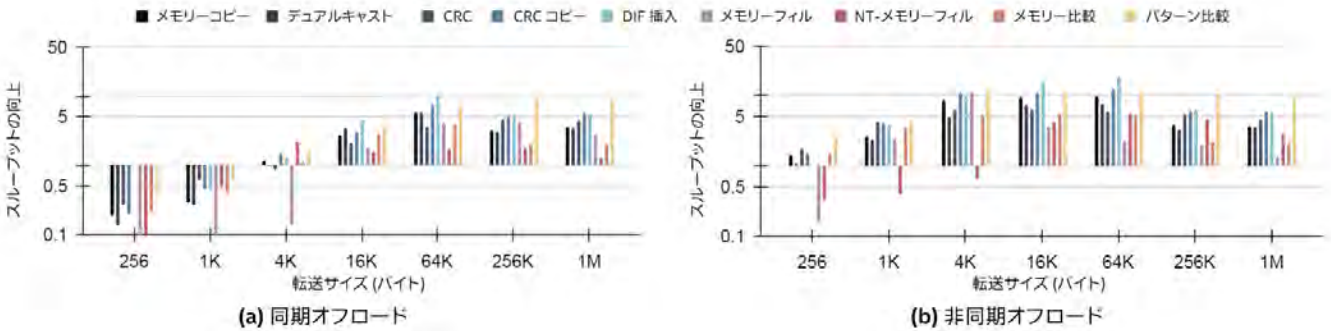


図 1. さまざまな転送サイズ（バッチサイズ : 1）における、対応するソフトウェアと比較したデータ・ストリーミング操作のスループットの向上。メモリーフィルと NT- メモリーフィルは、割り当て書き込みと非割り当て書き込みを指します（通常の `store` と `nt-store` に似ています）。

インテル® DSA の操作

ほかのアクセラレーターと同様に、インテル® DSA は記述子とワークキュー（WQ）を使用して CPU ソフトウェアと対話します（**図 2**）。

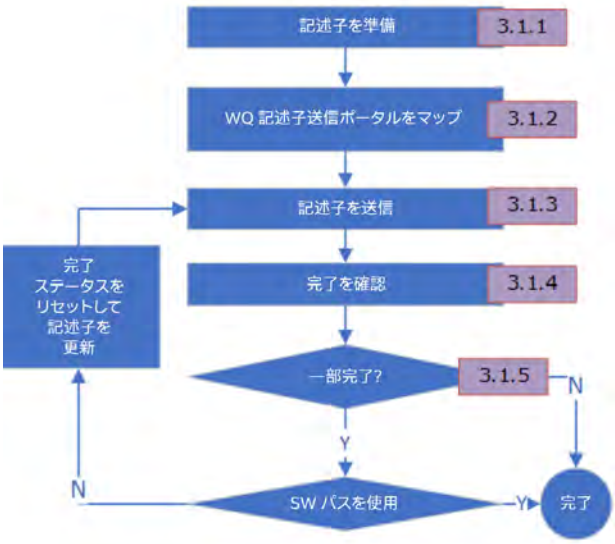


図 2. 記述子の処理手順

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

データの移動はインテル® DSA の最も一般的なユースケースです。データコピーを例に使用法を示します。キーと値ペアのサイズが大きなキー値格納アプリケーションでは、特定のキーの値を取得または更新するたびにメモリーコピーが発生し、大量の CPU サイクルが発生します。これらの操作をインテル® DSA にオフロードします。インテル® DSA インスタンスが `accel-config` などのツールにより列挙および構成されている場合、プログラマーはまず記述子のデータ構造を割り当ててフィルする必要があります（**図 3**）。記述子には、操作のタイプ、ソースアドレスとデスティネーション・アドレス、データ長など、目的の操作に関する情報が含まれています。

```
struct dsa_completion_record comp __attribute__((aligned(32)));

struct dsa_hw_desc desc = { };

desc.opcode = DSA_OPCODE_MEMMOVE;

/*
 * Request a completion - since we poll on status, this flag
 * must be 1 for status to be updated on successful
 * completion
 */
desc.flags = IDXD_OP_FLAG_RCR;

/* CRAV should be 1 since RCR = 1 */
desc.flags |= IDXD_OP_FLAG_CRAV;

/* Hint to direct data writes to CPU cache */
desc.flags |= IDXD_OP_FLAG_CC;

desc.xfer_size = BLEN;
desc.src_addr = (uintptr_t)src;
desc.dst_addr = (uintptr_t)dst;
comp.status = 0;
desc.completion_addr = (uintptr_t)&comp;
```

図 3. 記述子を初期化

記述子の準備ができたら、プログラマーの次のステップは、現在のプログラムで開かれてマップされている WQ に記述子を送信することです。WQ のタイプに応じて、ソフトウェアは ENQCMD または MOVDIR64B 命令を使用して記述子を送信します。どちらの命令も GNU* GCC（バージョン 10 以降）でサポートされています。プログラマーは、x86 組込み関数を使用してこれらの命令を呼び出すことができます（**図 4**）。

```
#include <x86gprintrin.h>

_mm_sfence();

if (dedicated)
    _movdir64b(wq_portal, &desc);
else {
    retry = 0;
    while (_enqcmd(wq_portal, &desc) && retry++ < ENQ_RETRY_MAX);
}
```

図 4. 記述子を送信

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

記述子の完了を確認するには、プログラマーは、インテル® DSA によって更新される、完了レコードのステータスフィールドをポーリングする必要があります。最も一般的な方法はスピンポーリングです（**図 5**）。PAUSE や UMONITOR/UMWAIT などの新しい命令を適用すると、コアが異なる電力ステートに移行できるため、プロセッサーの消費電力をさらに削減することができます（詳細は、「インテル® 64 および IA-32 アーキテクチャー・ソフトウェア開発者マニュアル」を参照してください）。

```
retry = 0;
while (comp.status == 0 && retry++ < COMP_RETRY_MAX);

if (comp.status == DSA_COMP_SUCCESS) {
    /* Successful completion */
} else {
    /* Descriptor failed or timed out
     * See the "Error Codes" section of the Intel® DSA Architecture Specification for
     * error code descriptions
     */
}
```

図 5. 記述子の完了を確認

ソフトウェア・ライブラリーを利用したインテル® DSA の活用

インテル® Data Mover Library（インテル® DML）とインテル® DSA Transparent Offload Library（インテル® DTO）（英語）を利用すると、インテル® DSA が使いやすくなります。インテル® DML は、データの移動と変換を行う高レベルの C/C++ API のセットを提供し、利用可能な場合は対応するインテル® DSA ユニットを呼び出します。インテル® DML は、高度な機能（インテル® DSA のハードウェア操作、非同期オフロード、ロードバランスなど）をサポートしています。インテル® DML v1.0.0 は、GitHub*（英語）から入手できます。リポジトリーのクローンを作成し、必要なパッケージをインストールした後、単純な CMake コマンドでコンパイルしてインストールできます（インストール・オプションの詳細は、インテル® DML のドキュメント（英語）を参照してください）。

Windows* OS:

```
mkdir build
cd build
cmake -DCMAKE_INSTALL_PREFIX=<install_dir> -G "NMake Makefiles" ..
cmake --build . --target install
```

Linux* OS:

```
mkdir build
cd build
cmake -DCMAKE_INSTALL_PREFIX=<install_dir> ..
cmake --build . --target install
```

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

図 6 に、インテル® DML の基本的な使用法を示します。基本的な手順は変わりませんが、詳細はインテル® DML API と抽象化により隠されています。キー値格納アプリケーションの例を続けます。記述子を手動で準備して送信する代わりに、プログラマーはインテル® DML を呼び出して、指定されたメモリーコピーをインテル® DSA にオフロードできます。

```
#include <dml/dml.hpp>
#include <numeric>
#include <vector>
#include <iostream>

constexpr auto size = 1024u;  // 1 KB

template <typename path>
int execute_mem_move() {
    std::cout << "Starting dml::mem_move example..." << std::endl;
    std::cout << "Copy 1KB of data from source into destination..." << std::endl;

    // Prepare data
    auto src = std::vector<std::uint8_t>(size);
    std::iota(src.begin(), src.end(), 0u);
    auto dst = std::vector<std::uint8_t>(size, 0u);

    // Run operation
    auto result = dml::execute<path>(dml::mem_move, dml::make_view(src), dml::make_view(dst));

    // Check result
    if (result.status == dml::status_code::ok) {
        std::cout << "Finished successfully." << std::endl;
    }
    else {
        std::cout << "Failure occurred." << std::endl;
        return -1;
    }

    if (src != dst) {
        std::cout << "Operation result is incorrect" << std::endl;
        return -1;
    }

    return 0;
}
```

図 6. インテル® DML を使用した基本的なデータ移動の例

インテル® DTO は、アプリケーションがソースコードを変更することなくインテル® DSA を透過的に活用できるようにする、干渉の少ないライブラリーです。ユーザーは、`-ldto` および `-laccel-config` リンカーオプションを使用してライブラリーを動的にリンクするか、アプリケーションの再コンパイルが不要な `LD_PRELOAD` の設定でライブラリーを事前にロードできます。`memmove()`、`memcpy()`、`memset()`、または `memcmp()` などの一般的なシステム API 呼び出しが使用されると、それらの呼び出しはインターセプトされ、対応するインテル® DSA の同期操作にアクセスするインテル® DTO の関数に置換されます。

汎用ライブラリーのインテル® DML とインテル® DTO に加えて、インテル® DSA を利用するドメイン固有のソフトウェア・ライブラリーもあります。例えば、ハイパフォーマンスなカーネルバイパス・ネットワーク・プログラミング向けの Data Plane Development Kit（DPDK）では、インテル® DSA がほかのベンダーの DMA エンジンとともに抽象化され、**dmadev** ライブラリーにラップされています。ユーザーは、対応する API を呼び出して、

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

ネットワーク・パケットのコピーをインテル® DSA インスタンスにオフロードできます。Storage Performance Development Kit（SPDK）も `/lib/accel` ライブラリーでインテル® DSA をサポートしており、さまざまな操作をオフロードできます。インテル® DSA は、libfabric や MPI などの HPC ライブラリーでも有効になっており、従来の科学技術計算ワークロードだけでなく、新しい分散 ML/AI アプリケーションでも使用できます。

インテル® DSA を最大限に活用する

ストリーミング・データ用のオンチップ・アクセラレーターとして、インテル® DSA は、プログラミング・モデルを適切にチューニングすると最適なパフォーマンスをもたらします。我々の経験に基づいて、インテル® DSA を使用する際のガイドラインと推奨事項をまとめました（詳細は、このテクニカルレポート（英語）を参照してください）。

バッチサイズと転送サイズの適切なバランスを維持する

特定のサイズのワークをインテル® DSA にオフロードするには、メモリー領域全体に対して 1 つの記述子を使用するか、同じ集約サイズに対して複数の小さな記述子を使用してバッチ処理します。一般的な傾向として、同じオフロードされたワークに対して大きなバッチ処理を行うと、スループットは低下します。個々の記述子をインテル® DSA の内部で処理し、対応するデータをメモリーから読み取る必要があるため、増加した記述子を管理するオーバーヘッドが発生し、これらの操作の実効スループットが低下します。オフロードするデータが連続している場合、同じサイズのより大きな 1 つの記述子に結合すると、スループットとレイテンシーの両方が改善される可能性があります。

同期的にオフロードすると、バッチの最大化と転送サイズの最大化の間のスループットの最適点を示すパターンが明らかになります。ワークをある程度の数（4 ～ 8 個）の記述子でバッチ処理すると、最良の結果が得られます。これは、メモリーの連続領域のフェッチとバッチされた記述子の処理によるレイテンシーのバランスが取れるためです。

可能であればインテル® DSA を非同期的に使用する

操作を非同期方式でインテル® DSA にオフロードすると、CPU コアとインテル® DSA ハードウェアの両方に最適な効率とパフォーマンスが提供されます。この方式は、インテル® DML を使用して簡単に実装できます。非同期の可能性が制限されている場合、キャッシュ汚染が許容できるのであれば、4KB 未満の転送サイズには CPU コアを使用します。インテル® DSA は、オンチップ・アクセラレーターとして、キャッシュおよび（ヘテロジニアス）メモリー階層を操作する多くのオプションを提供します。

データの転送先を適切に制御する

常に LLC に送られる完了レコードとは異なり、記述子のデスティネーション・アドレスに書き込まれるデータは、LLC またはメインメモリーに送ることができます。インテル® DSA は、ユーザーが優先するデスティネーションを示すヒントを提供（つまり、ワーク記述子のキャッシュ制御フラグを設定）できるようにすることで、このキャッシュ制御機能の利用を支援します。フラグを 0 に設定すると、LLC に対応するキャッシュラインがある場合は無効にされ、データはメモリーに書き込まれます。フラグを 1 に設定すると、対応するキャッシュラインを割り当てて、

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

データは直接 LLC に書き込まれます。この機能の基礎となる原理は、インテル® データ・ダイレクト I/O テクノロジー（インテル® DDIO）の原理である、LLC をプロセッサーと I/O デバイス間の中間バッファーとして利用するダイレクト・キャッシュ・アクセス（DCA）スキームと同じです。一般に、インテル® DDIO は、データアクセスのレイテンシーを軽減し、メモリー帯域幅への負担を減らすことにより、システムのパフォーマンスを向上させます。ただし、LLC で干渉を引き起こす可能性があるため、この機能を使用する場合は注意してください。

キャッシュ汚染は、限られたハードウェア・リソースを共有するプロセスに悪影響を及ぼします。多くのデータセンターのワークロードでは、対立するバックグラウンド・キャッシュの退避により生じるレイテンシーによって、プライマリー・アプリケーションに設定された競争力の高いサービス・レベル・アグリーメント（SLA）が損なわれる可能性があります。一方、パフォーマンスにとって重要なデータ、または近い将来コアで使用されるデータをキャッシュに直接書き込むと、アプリケーションのアクセス・レイテンシーとスループットの利点が得られます。プログラマーは、アプリケーションの動作に基づいてデータの転送先を適切に選択する必要があります。

ヘテロジニアス・メモリー・システムでデータを移動する有力な候補としてのインテル® DSA

NUMA リモートメモリー、パーシステント・メモリー、CXL ベースのメモリーなど、異なるメモリーメディア間でのデータの移動は、最新の階層型メモリーシステムでは一般的です。インテル® DSA は、利用可能なハードウェア・リソースを通じて、高度な構成の柔軟性を提供します。適切な構成を活用することにより、インテル® DSA のハードウェア使用率が最適化され、パフォーマンスが向上します。

処理エンジンレベルの並列処理を活用する

グループで使用される処理エンジンの数が観測される最大スループットに影響を与える可能性があります。グループあたりの処理エンジンの数を増やすと、スループットが向上します。転送サイズが小さいほど優れたパフォーマンスが得られるため、ユーザーはこれらのグループにオフロードするタスクの一般的な転送サイズに注意する必要があります。

WQ 構成を最適化する

バッチ処理または専用 WQ（DWQ）を使用すると、共有 WQ（SWQ）と比較して大きな利点が得られます。共有 WQ がほかの多くのスレッドで使用されない限り、多くの DWQ を使用するように構成を変更することにより、使用率とスループットを向上させることができます。使用するスレッドが少ない場合、SWQ のパフォーマンスは低下する可能性がありますが、WQ の総数よりも多くのスレッドを使用すると、同時実行の管理がハードウェアにオフロードされ、ほかの構成よりもパフォーマンスが向上します。また、グループ内の WQ 間のパフォーマンスを分離するため、WQ を共有または専用に構成できます。インテル® DSA の WQ エントリーは限られているため、1 つの WQ に 32 のエントリーを割り当てると、スループットはほぼ最大になります。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

まとめ

インテル® DSA や、第 4 世代インテル® Xeon® スケーラブル・プロセッサーに搭載されているほかのオンチップ・アクセラレーターには、データセンター・コストと総所有コストを削減できる可能性があります。この記事では、インテル® DSA とその基本的な使用方法の概要、このアクセラレーターを最大限に活用するいくつかのガイドラインを説明しました。ソフトウェア・エコシステムの成長に伴って、インテル® DSA が多くのデータセンターのワークロードに採用されると考えています。インテルは、オープンソース・コミュニティーと積極的に協力して、このエコシステムを構築および実現しています。

関連情報（英語）

- インテル® データ・ストリーミング・アクセラレーター（インテル® DSA）の導入
- インテル® データ・ストリーミング・アクセラレーター・アーキテクチャー仕様 [PDF]
- インテル® データ・ストリーミング・アクセラレーター・ユーザー・ガイド [PDF]
- インテル® DSA Performance Micros
- インテル® Data Mover Library（インテル® DML）

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

並列標準を洗練する：多様性、アライメント、相互作用

標準ベースの並列プログラミングの将来を見据える

John Pennycook インテル コーポレーション ソフトウェア・イネーブリング & 最適化アーキテクト

oneAPI は、言語（Khronos Group の SYCL*）、標準化されたライブラリー・インターフェイス（ニューラル・ネットワーク、データ・アナリティクス、その他）、およびハードウェアに近いプログラミング・インターフェイス（レベルゼロ）を含むアクセラレーター・プログラミングのオープン仕様（oneapi.io（英語）を参照）です。oneAPI は、ハードウェアとベンダーに依存しないように設計されており、複数のベンダーの CPU、GPU、FPGA、その他のアクセラレーターで動作することが実証されています。oneAPI は、プログラミングに標準ベースのアプローチを提供し、開発者がハードウェアとソフトウェア環境間の移植性を犠牲にすることなく高いパフォーマンスを実現できるようにします。

ソフトウェア・スタックのさまざまなレベルにわたる複数の**相互運用可能な**標準のコレクションが oneAPI を可能にしています。その一部（レベルゼロや SYCL* の拡張など）は oneAPI 仕様で採用されており、SYCL* や SPIR-V* などの業界標準は、oneAPI を使用した移植可能なプログラミングの基盤を提供します。さらに、OpenMP*、ISO C++、ISO Fortran などは、oneAPI でアクセラレーター・プログラミングの代替構文を記述するのに使用されます。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

現在では、1 人の開発者が 1 つの言語を 1 つの抽象化レベルで使用して大規模なアプリケーションを作成することはまずないため、これらの標準間の相互運用性は非常に重要です。ほかの開発者が作成したライブラリーを呼び出して、さまざまな手法を組み合わせて使用する方がはるかに一般的です（例えば、SYCL* で作成された最適化ライブラリーを使用して Fortran や OpenMP* でアプリケーションを作成する、など）。異なる標準が新しい問題に対して独自のソリューションを提供することは珍しいことではないため、相互運用性を維持するには継続的な努力が必要です。標準が異なると、対象とする抽象化のレベルや推奨するコーディング・スタイルも異なることがあるため、新しい機能と既存の機能や確立された規約との関係を常に考慮する必要があります。しかし、標準は、互いに学習し合って、用語などを調整し、必要に応じて機能を互いに追加することがあります。

SYCL*、C++、OpenCL* の過去 10 年間を振り返ると、この調整と相互作用のプロセスが実際に作用していることが分かります（**図 1**）。例えば、OpenCL* 2.0 は C++11 のメモリーモデルの概念と用語を採用し、スコープと複数のアドレス空間の概念を拡張しました。その後、グループやグループ関数などの OpenCL* 2.0 の機能と C++17 の並列アルゴリズムを組み合わせた SYCL* 2020 のグループ・アルゴリズム・ライブラリーが作成され、開発者は使い慣れた C++ 構文を使用して、ハードウェア階層の複数のレベルでベンダーに最適化された操作にアクセスできるようになりました。

これは現在進行中のプロセスであり、IWOCL 2023 で発表した「SYCL* と ISO C++ の並列処理の調整に向けて」（英語）では、SYCL* のワークアイテムや C++17 のスレッド実行などの概念の間のギャップを埋めるように設計された、SYCL* に対するいくつかの新しい調整を提案しました。これは、特定の C++17 実行ポリシーが SYCL* デバイスで正しく動作できるようにするために必要なステップであり、これらの調整により、SYCL* で使用する C++17 並列アルゴリズムの動作の推測が容易になります。

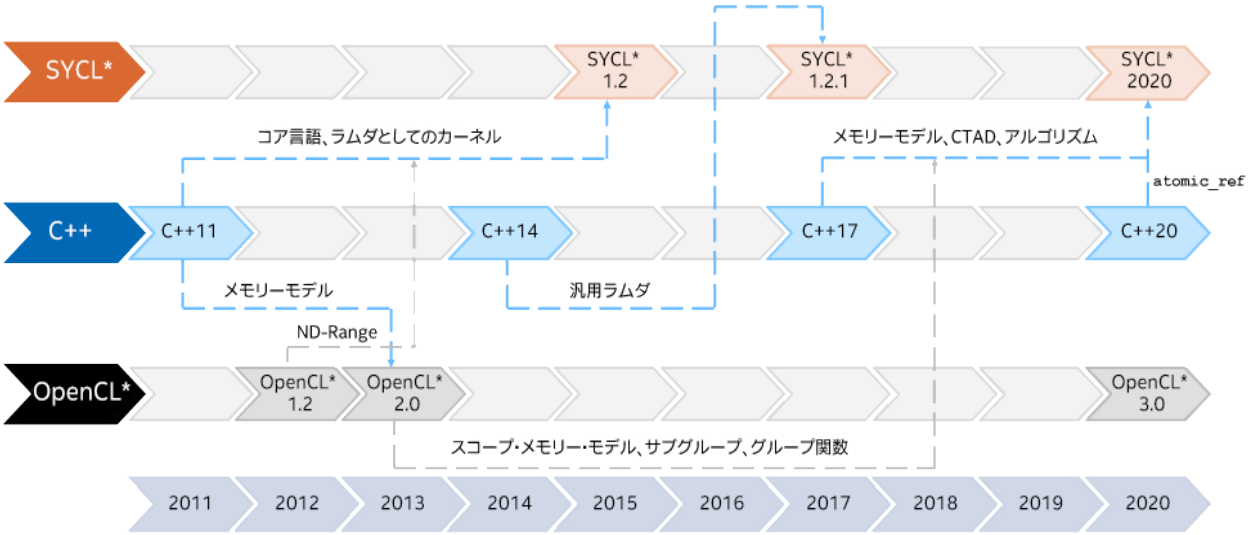


図 1. SYCL*、ISO C++、OpenCL* の並行進化

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

私たちは、SYCL* が C++ のヘテロジニアス・プログラミングの将来に影響を与える可能性があると考えています。そして、実装者の SYCL* での経験が、C++ 向けに提案されている新しい高レベルの抽象化の設計に反映されると信じています。また、SYCL* が今後も開発者にとって重要な役割を果たすと信じており、関連する C++ 抽象化と完全に相互運用可能なメカニズムを通じて、低レベルの最先端のハードウェア機能に対するダイレクトアクセスを提供します。

未来はどのようになるでしょうか？ 確かなことは言えませんが、P2300（英語）（`std::execution`）や P2500(英語)(「C++ 並列アルゴリズムと P2300」)など、いくつかの ISO C++ の提案にヒントが含まれています。これらの提案が承認されれば（早くても 2026 年まで実現しないでしょう）、**図 2** のようなコードを記述できるようになります。

```
// SYCL* を使用して特定の「scheduler」を表現
// get_scheduler() は(まだ)存在しないことに注意
auto q = sycl::queue{sycl::gpu_selector_v};
scheduler auto sch = q.get_scheduler();

// P2300/P2500 の機能を使用して並列ループを SYCL* スケジューラーに送信
std::for_each(std::execute_on(sch, std::execution::par_unseq), begin, end,
  [=](auto i) {
  ...
});
```

図 2. 未来を垣間見る - P2300/P2500 で提案されている機能を使用した C++ 並列アルゴリズムと SYCL* の組み合わせ

2026 年まで待てない人は、oneAPI DPC++ ライブラリー（oneDPL）を使用して C++ 並列アルゴリズムと SYCL* を組み合わせることができます。oneDPL は、C++ 並列アルゴリズムを SYCL* デバイスで実行できるようにするオープンソースのライブラリーであり、SYCL* が実行できる環境であればどこでも実行できるアプリケーションを開発する、生産性の高いソリューションを提供します。oneDPL は、ISO C++ の将来の拡張の可能性を検討する手段としても機能します。C++20 の Ranges に基づいた並列範囲ベースのアルゴリズム（英語）を標準化し、非同期並列アルゴリズム（英語）を表現する方法を検討する取り組みがすでに進行中です。oneDPL を使用して C++ 並列アルゴリズムを SYCL* デバイスに送信するのは、SYCL* 対応の実行ポリシーを使用するのと同じくらい簡単です（**図 3**）。oneDPL の試験的な機能を使用すると、遅延評価ビューを活用してカーネル融合を有効にし、さらに高いレベルのパフォーマンスと生産性を実現することができます（**図 4**）。

```
// oneDPL を使用して並列ループをデフォルトの SYCL* デバイスに送信
std::for_each(oneapi::dpl::execution::dpcpp_default, begin, end,
  [=](auto i) {
  ...
});
```

図 3. 現在 oneDPL で利用可能な機能を使用した C++ 並列アルゴリズムと SYCL* の組み合わせ

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

```
// 3 つのアルゴリズムを 3 つのカーネルとしてデフォルトの SYCL* デバイスに送信
using namespace oneapi::dpl;
reverse(execution::dpcpp_default, begin(data), end(data));
transform(execution::dpcpp_default, begin(data), end(data), begin(result),
          [](auto i){ return i * i; });
auto res = find_if(execution::dpcpp_default, begin(result), end(result),
                   pred);

// 範囲変換のパイプラインをデフォルトの SYCL* デバイスに送信
// 単一のカーネルとして実行される
using namespace oneapi::dpl::experimental::ranges;
auto res = find_if(execution::dpcpp_default,
                   views::all(sycl::buffer{data})
                   | views::reverse
                   | views::transform([](auto i){return i * i;}), pred);
```

図 4. oneDPL の試験的な機能を使用した C++ の範囲と SYCL* の組み合わせ

oneDPL の現在の構文は最終的な構文と多少異なる可能性がありますが、作業中の標準化プロセスではよくあることです。将来の C++ 標準として最終的に採用されるものは、複数のライブラリーと複数のハードウェア・ベンダーの経験を取り入れ、共通の要件とベスト・プラクティスを確立する長年の作業の結果となるでしょう。

開発者は OpenMP*、SYCL*、ISO C++、ISO Fortran のどれを使用してプログラムを作成するべきか質問されることがあります。答えは「どれでもかまわない」です。これらはすべて、ヘテロジニアス・プログラミングに対する有効なアプローチであり、それぞれ固有の長所と短所があります。Fortran の `ISO_C_BINDING`、OpenMP* の `interop` 節、SYCL* のバックエンド相互運用性インターフェイスなどの機能のおかげで、これらのアプローチを組み合わせることは、これまでになく容易になりました（**図 5**）。oneAPI の中核となる標準間の相互運用性に重点を置くことにより、当面の業務に最適なツールを組み合わせて使用することができます。

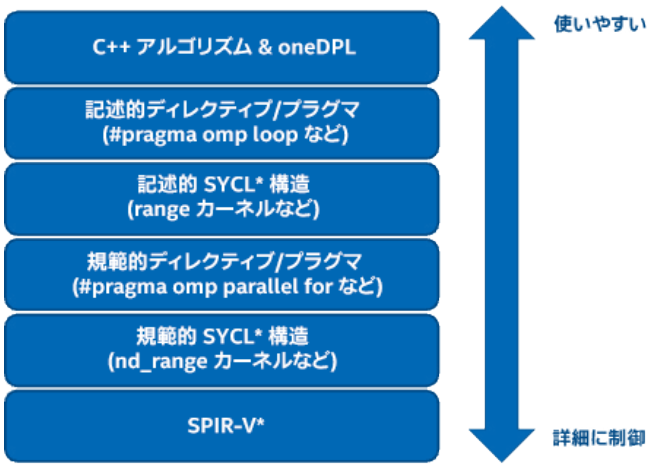


図 5：ヘテロジニアス・プログラミングに対するアプローチの比較：高いレベルの抽象化はハイパフォーマンスなプログラムを作成する生産性の高いソリューションを提供し、低いレベルの抽象化は熟練の開発者に詳細な制御を提供する

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

堀のある AI から責任ある AI へ

生成 AI のリスクを最小限に抑える

Huma Abidi インテル コーポレーション AI ソフトウェア製品ジェネラル・マネージャー兼シニア・ディレクター
Haihao Shen インテル コーポレーション AI ソフトウェア・アーキテクト

OpenAI の ChatGPT* は、記録的な成長を遂げ、生成 AI の世界に熱狂を引き起こしました。なんと、最初の 1 週間で 100 万人以上のユーザーを魅了しました。Google、Microsoft、Meta（旧称 : Facebook）などの大手テクノロジー企業が大規模言語モデル（LLM）レースに参加する一方で、小規模なスタートアップ企業も前進を続けています。競争上の優位性のために必要な秘密性と安全のために必要な透明性のバランスを取ることがますます大きな問題となっています。一部の LLM とは異なり、OpenAI はトレーニング・セットや GPT-4* アーキテクチャーの詳細を公開していないため、一部から批判を浴びています。

最近流出した Google のメモ（英語）では、Google と OpenAI に LLM テクノロジーにおける競争上の優位性、つまり「堀」が欠けていることへの懸念が強調されています。このメモでは、オープンソースの代替品がいかに小さく、速く、安価で、カスタマイズ可能であるかが強調されていました（この点については、本号の「**独自のカスタム・チャットボットの作成**」で説明します）。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

ChatGPT* のような LLM は優れた機能を実証していますが、生成 AI モデルの出現によるマイナス面についての懸念も生じています。大規模モデルと小規模モデル、オープンシステムとクローズドシステムに関する継続的な議論を行うことは重要ですが、パフォーマンスと精度だけが重要ではないことを認識する必要があります。公平性、説明可能性、持続可能性、プライバシーなどの要素も考慮する必要があります。責任ある AI の実践を守ることが、AI の社会的な価値を最終的に決定します。おそらく、AI アプリケーションが倫理的ベスト・プラクティスに準拠することを義務付ける多くの規制が今後導入されるでしょう。今年後半に可決予定の次期 EU 人工知能法（英語）が AI システムを管理する世界初の規制となる見込みです。この法律は、透明性とリスク管理に関するガイドラインを導入することにより、AI に対して人間中心の倫理的なアプローチを促進することを目的としています。

生成 AI モデルはインターネット上で利用可能な膨大な量のデータから学習しますが、これらのデータには社会に存在するバイアス（偏見や先入観など）も含まれており、これらのバイアスを意図せず永続させたり増幅させたりする可能性があります。LLM が誤った情報、フィッシング・メール、ソーシャル・エンジニアリング攻撃などを生成または拡散するために操作されることも考えられます。また、悪意のある攻撃者が意図的に偏った情報や虚偽の情報を使用してモデルをトレーニングすることで、誤解を招くコンテンツが大規模に拡散する可能性もあります。このようなモデルを使用すると、説得力のあるディープフェイクのビデオやオーディオコンテンツを作成できます。例えば、最近 AI が生成したアメリカ国防総省で爆発という偽画像（英語）が急速に広まりました。このようなフェイクニュースは、今後の米国の選挙に対する大きな脅威として浮上しています。この記事で述べられているブルッキングス研究所テクノロジー・イノベーション・センターのシニアフェローである Darrell West 氏の懸念は現実的なものです。

LLM はしばしば「幻覚」を引き起こし、不正確な情報を生成することがあります。これは、モデルが診断や治療の決定に影響を与え、患者に害を及ぼす可能性があるヘルスケアなどの業界では特に深刻な問題となります。AI が幻覚を引き起こすことは既知の現象であるにもかかわらず、LLM を使用し続け、LLM の情報を疑うことなく受け入れる人は後を絶ちません。最近の例では、弁護士が裁判の準備書面として提出した判例の要約が存在しなかったことから、調査に ChatGPT* を使用していたことが判明しました。これらの架空の判例は ChatGPT* により作成されたものでした（英語）。

責任ある AI を求める声はこれまでになく高まっています。コモンウェルス・クラブでのインタビューで、Stuart Russel 教授は ChatGPT* について「ある意味、我々はインフォームド・コンセントをまったく得ることなく、人類に対して大規模な実験を行っていると言える」と述べています。Russell 教授や Elon Musk 氏を含む 1,000 人以上の AI エキスパートからなるグループは、LLM のデプロイを一旦停止するように求めています（英語）。議員、業界のリーダー、研究者たちは、AI の安全なデプロイを確保するため、AI の周りにガードレールを設置して、厳格な規制を構築することの必要性に同意しています。業界のリーダーたちは、AI テクノロジーが人類存続の脅威となる可能性があることを認めています。Center for AI Safety は、「AI による絶滅のリスクを軽減することは、パンデミックや核戦争などのほかの社会規模のリスクと並ぶ世界的な優先事項であるべきである」という声明（英語）を発表しています。この声明には、Geoffrey Hinton（トロント大学名誉教授）、Yoshua Bengio（モントリオール大学教授）、Demis Hassabis（Google DeepMind CEO）、Sam Altman（OpenAI CEO）、Dario Amodei（Anthropic CEO）（敬称略）を始めとする、AI 分野の著名人が署名しました。AI のリスクを軽減するには、責任ある開発、慎重なデータセットのキュレーション、継続的な研究、堅牢な倫理ガイドラインが不可欠です。バイアスに対処し、誤った情報を排除し、ユーザーのプライバシーを保護するには、LLM の透明性と説明責任を確保し、定期的な監査を行うことが重要です。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

AI テクノロジーに取り組んでいる企業や個人が、AI の行動規範に従ってソフトウェアが開発およびデプロイされていることを確認する必要があることは明らかです。オープンソースのインテル® Explainable AI Tools（英語）を使用すると、ユーザーは事後モデルの蒸留と可視化を実行して、TensorFlow* モデルと PyTorch* モデルの両方の動作を予測することができます。これらのモデルは、ユーザーが公平性と解釈可能性の問題を検出して防ぐことができるように設計されています。例えば、ユーザーは、インテル® Explainable AI Tools に含まれるオープンソースの Python* モジュール、モデル･カード･ジェネレーター（英語）を使用して、モデルの詳細と、TensorFlow* モデルと PyTorch* モデルの両方のパフォーマンスと公平性のメトリックを表示する定量分析を含む、インタラクティブな HTML レポートを作成できます。これらのモデルカードは、従来のエンドツーエンド・プラットフォームの一部として使用でき、透明性、公平性、説明責任を促進する、表、画像、テキストデータの ML パイプラインをデプロイします。

LLM は通常、大規模な公開データセットでトレーニングされた後、機密性の高いデータ（金融や医療など）で微調整されます。Open Federated Learning（OpenFL）（英語）のようなテクノロジーには機密計算（英語）が組み込まれており、機密データに基づいて LLM を安全に微調整できるため、幻覚やバイアスを減らしながらモデルの一般化可能性を向上させることができます。

AI は、重要な専門知識が不足している、経済的に不利な分野を支援できる可能性を秘めています。現在、LLM の実行には膨大な計算能力が必要であり、通常はクラウドや複数のアクセラレーターを搭載した高価なオンプレミス・サーバーで実行されます。我々は、クラウドに接続できない環境や低コストのエッジ・コンピューティング・デバイスでも高度な AI 技術を利用できるように、LLM の計算の複雑さを軽減し、LLM ベースの推論の効率を高めることに取り組んでいます。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

独自のカスタム・チャットボットの作成

大規模な言語モデルをインテル・プロセッサーで迅速かつ簡単にトレーニングする

Haihao Shen インテル コーポレーション AI ソフトウェア・アーキテクト
Xinyu Ye、Kaokao Lv、および Xuhui Ren 同 AI ソフトウェア・エンジニア
Huma Abidi 同 AI ソフトウェア製品ジェネラル・マネージャー兼シニア・ディレクター

ChatGPT*（英語）、GPT-4*（英語）、Bard*（英語）などの対話エージェントの途方もないパフォーマンスにより、大規模言語モデル（LLM）が大きな注目を集めています。しかし、LLM のトレーニングや微調整には膨大なコストと時間が必要です。これは、モデルサイズとデータセットが大きいためです。

この記事では、簡単に入手できるハードウェアでカスタム・チャットボットをトレーニングして微調整する方法を説明します。第 4 世代インテル® Xeon® スケーラブル・プロセッサーを使用して、体系的な手法を使用してチャットボットを作成し、ドメイン固有のデータセットと最適化された微調整コードベースを生成します。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

アプローチ

スタンフォード大学が公開した Alpaca（英語）は、Meta の LLaMA（英語）モデルから微調整された、命令に従う言語モデルです。このプロジェクトからヒントを得て、我々はカスタムのドメイン固有チャットボットを作成する、強化された手法を開発しました。使用できる言語モデルは（パフォーマンスの高い言語モデルも含めて）いくつかありますが、オープンモデルである Alpaca を選択しました。

チャットボットのワークフローは、ガイド付きシード生成、ガイドなし（フリー）シード生成、サンプル生成、および微調整の 4 つのステップで構成されます（**図 1**）。

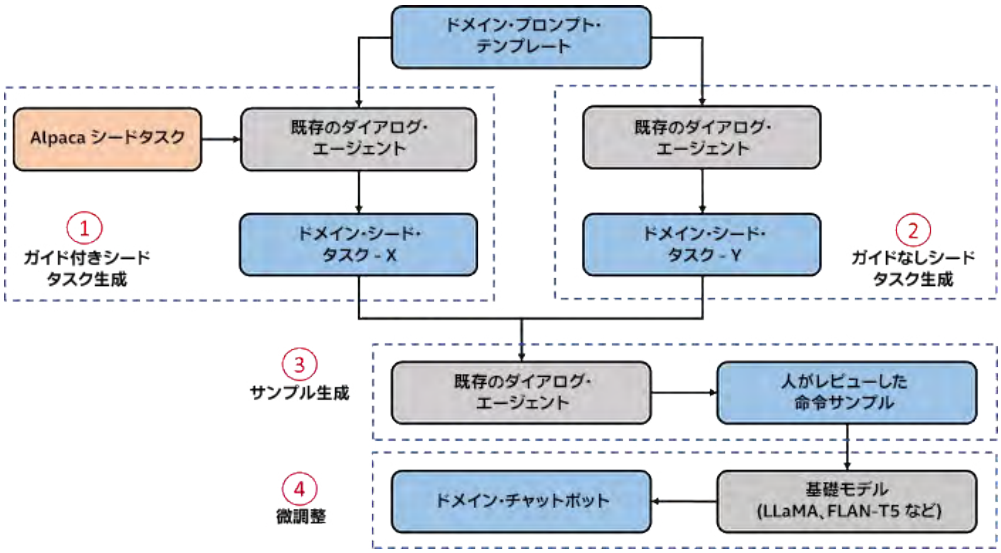


図 1. チャットボットの微調整の概要

これらのステップを説明する前に、シードタスクの生成に役立つプロンプト・テンプレートを紹介します。一般的なタスク向けの Alpaca のサンプルプロンプトを**図 2** に示します。

```
You are asked to come up with a set of 20 diverse task instructions. These task instructions will be
given to a GPT model and we will evaluate the GPT model for completing the instructions.

Here are the requirements:
1. Try not to repeat the verb for each instruction to maximize diversity.
2. The language used for the instruction also should be diverse. For example, you should combine
questions with imperative instrucitons.
3. The type of instructions should be diverse. The list should include diverse types of tasks like
open-ended generation, classification, editing, etc.
 ...

List of 20 tasks:
```

図 2. シードタスク生成のプロンプト・テンプレート

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

「生成されるタスクの指示は `<domain_name>` の問題に関連している必要がある」という新しい要件を追加してテンプレートを変更しました。この要件は、指定されたドメインに関連するシードタスクの生成に役立ちます。多様なシードタスクを生成するため、ガイド付きシードタスク生成とガイドなし（フリー）シードタスク生成の両方を使用します。

ガイド付きシードタスク生成は、Alpaca の既存のシードタスク（英語）を活用します。シードタスクごとに、ドメイン・プロンプト・テンプレートのコンテンツを結合して、既存のダイアログ・エージェントにフィードします。対応する数のタスク（例えば、**図 2** ではプロンプト・テンプレートで定義した 20 のタスク）が生成されることが予想されます。このようなテキスト生成は、因果関係言語モデルの典型的なユースケースの 1 つです。

ガイドなしシードタスク生成は、追加のシードタスクを指定しないで、ドメイン・プロンプト・テンプレートをダイアログ・エージェントに直接フィードします。ここでは、ガイドなしシードタスク生成を「フリー」と呼びます。このアプローチを使用して、新しいドメイン・シード・タスクを生成します（**図 3**）。

```
{
    "instruction": "Summarize the key features of Intel's latest processor.",
    "input": "",
    "output": "Intel's latest processor features improved performance, higher energy efficiency,
advanced AI capabilities, enhanced security, and support for cutting-edge memory and storage
technologies."
},
```

図 3. ドメイン・シード・タスク

これらのシードタスクを使用して、既存のダイアログ・エージェントを再度利用し、指示サンプルを生成します。ドメイン・プロンプト・テンプレートを使用するため、出力は「命令」、「入力」、「出力」形式の要件に従います。このプロセスを繰り返して、微調整に使用する 2,000 個の命令サンプルを生成します（**図 4**）。

```
{
    "instruction": "Explain how Intel Xeon Scalable processor can improve machine learning tasks",
    "input": "",
    "output": "Intel Xeon Scalable processors feature advanced AI acceleration architectures, advanced
caching solutions, and improved memory bandwidth, allowing them to handle heavy machine learning
workloads quickly and efficiently."
},
```

図 4. ドメイン命令サンプル

ドメイン・シード・タスクと指示サンプルには類似性があります。これらはそれぞれ ChatGPT* プロンプトとその結果の出力であり、一方が他方に影響を与えると考えることができます。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

カスタム・チャットボットをトレーニングする

数十億のパラメーターを使用して LLM 全体を微調整するのではなく、低ランク適応（LoRA）（英語）アプローチを使用して LLM を効率良く微調整します。LoRA は、事前トレーニング済みモデルの重みを固定し、トレーニング可能なランク分解行列をトランスフォーマー・アーキテクチャーの各層に挿入して、ダウンストリーム・タスクのトレーニング可能なパラメーターの数を大幅に削減します。

パラメーター効率に優れた微調整に加えて、ハードウェア･アクセラレーションとソフトウェア･アクセラレーションを利用して微調整プロセスを高速化します。ハードウェア・アクセラレーションの例として、AI パフォーマンスを高速化するために特別に設計された、第 4 世代インテル® Xeon® スケーラブル・プロセッサーで利用可能なインテル® アドバンスト･マトリクス･エクステンション（インテル® AMX）の AMX-BF16（英語）命令があります。PyTorch*（英語）、PyTorch* 向けインテル® エクステンション（英語）および Hugging Face* トランスフォーマー（英語）に含まれるソフトウェアの最適化も、これらのフレームワークやライブラリーの最適化されていないバージョンと比較してパフォーマンスの高速化に役立ちます。

微調整プロセスをさらに改善するため、命令サンプルの連結を有効にすることもできます。基本的な考え方は、異なる長さの複数のトレーニング・サンプルを用意する代わりに、いくつかのトークン化された文を連結してトレーニング・サンプルとして長い連結された文にする、というものです。この手法は、基礎となるハードウェアの効率を最大化するのに役立ちます。

上記の最適化はすべて、単一の計算ノードで行われます。微調整プロセス中に分散データ並列処理を使用してマルチノード微調整を実行し、さらに多くの計算能力を活用することもできます。

では、ドメイン・チャットボットのトレーニングを開始しましょう。

- トランスフォーマー向けインテル® エクステンション（英語）のサンプルコードを確認します。
- 要件ファイル（英語）に定義されている必要なソフトウェア・パッケージをインストールします。
- 事前トレーニング済みモデルをダウンロードします。FLAN-T5 は Hugging Face*（英語）からダウンロードできます。あるいは、Meta*（英語）から直接 LLaMA へのアクセスをリクエストするか、Hugging Face*（英語）の代替バージョンを使用します。
- **図 5** のスクリプトとコマンドを使用してトレーニングを開始します。詳細は、ユーザーガイド（英語）を参照してください。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

```
CCL_WORKER_COUNT=1 python -m torch.distributed.launch --master_addr=<MASTER_ADDRESS> --nproc_per_node=
<NUM_PROCESSES_PER_NODE> --nnodes=<NUM_NODES> --node_rank=<NODE_RANK> \
    finetune_clm.py \
        --model_name_or_path "model_name" \
        --train_file "/path/to/training_data" \
        --dataset_concatenation \
        --per_device_train_batch_size 8 \
        --per_device_eval_batch_size 8 \
        --gradient_accumulation_steps 1 \
        --do_train \
        --learning_rate 2e-5 \
        --num_train_epochs 3 \
        --logging_steps 100 \
        --save_total_limit 2 \
        --overwrite_output_dir \
        --log_level info \
        --save_strategy epoch \
        --output_dir "model_output_dir" \
        --bf16 --no_cuda --xpu_backend ccl
```

図 5. マルチノード微調整スクリプト

結果

Alpaca からの 52,000 個の一般命令サンプルと生成した 2,000 個のドメイン命令サンプルを使用して、2 つの人気の高い LLM（LLaMA-7B および FLAN-T5 XL）で微調整をテストしました。前述のソフトウェア最適化手法を使用すると、第 4 世代インテル® Xeon® スケーラブル・プロセッサーで最大 25 倍スピードアップしました（**図 6**）。

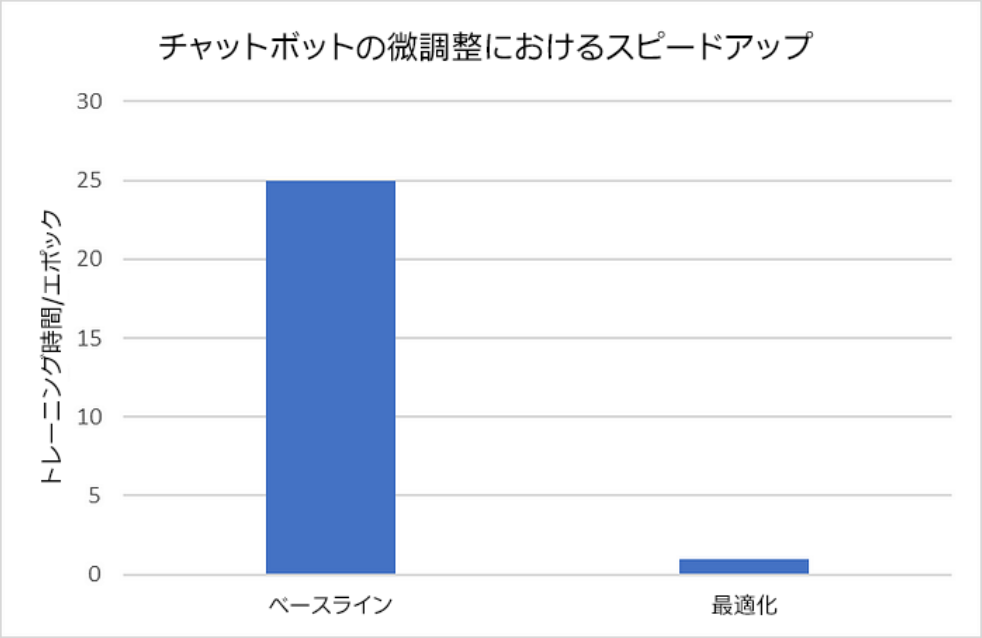


図 6. チャットボットの微調整におけるスピードアップ

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

ゼロショット設定と少数ショット設定でモデルを評価する新しい大規模マルチタスク言語理解 (MMLU) (英語) ベンチマークを使用して、事前トレーニング中にモデルが取得した知識を測定しました。**表 1** から、カスタム･チャットボットが平均してほとんどのドメインサブタスクで優れた結果を示していることが分かります。MMLU の結果は GPT-3* (175B) とほぼ同等ですが、パラメーター・サイズでは 25 倍効率的です。インテルのハードウェア・アクセラレーションとソフトウェアの最適化を活用しながら、ほかのオープン言語モデルを使用してこのビルドを試してみてください。

モデル	パラメーター	人文科学	STEM	社会科学	その他	平均
Alpaca	7B	38.3	33.2	44.5	47.0	40.6
カスタム	7B	**39.2**	**32.9**	**44.7**	**47.3**	**40.9**

表 1. MMLU ベンチマーク

MMLU ベンチマークに加えて、次のような会話評価も行いました。

入力プロンプト : Where to find the product information about Intel microchip?
(インテルのマイクロチップに関する製品情報はどこで見つかりますか ?)

Alpaca の返答 : You can find product information about Intel microchips on their website, Intel.com. There you will find information about their processors, graphics cards, motherboards, and other products. You can also find product specifications, tutorials, and other resources.
(インテルのマイクロチップに関する製品情報は、インテルのウェブサイト、Intel.com で見つけることができます。このサイトでは、インテルのプロセッサー、グラフィックス・カード、マザーボード、その他の製品に関する情報が見つかります。製品仕様、チュートリアル、その他のリソースも見つけることができます。)

カスタム・チャットボットの返答 : You can find the product information about Intel microchips on their website, https://www.intel.com/content/www/us/en/support/processors/top-questions.html.
(インテルのマイクロチップに関する製品情報は、インテルのウェブサイト、https://www.intel.com/content/www/us/en/support/processors/top-questions.html で見つけることができます。)

まとめ

トランスフォーマー向けインテル® エクステンション (英語) で、カスタム・チャットボットのソースコード (英語) とドメイン固有のデータセット(英語)をリリースしました。インテルのプロセッサー上で独自のチャットボットを作成し、AI ワークフローの一部としてほかのインテル® AI ツール (英語) や最適化を検討してみてください。

次のステップは、量子化と圧縮技術を有効にして、インテルのプラットフォーム上でチャットボットの推論を高速化することです。最新の最適化に関する情報を受け取りたい場合は、トランスフォーマー向けインテル® エクステンション (英語) に星を追加してください。ご質問がある場合はお気軽にお問い合わせください。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

Hugging Face* と oneAPI を使用した Falcon の 70 億パラメーター・モデルの微調整

インテル® アドバンスト・マトリクス・エクステンション（インテル® AMX）を搭載したインテル® Xeon® プロセッサー上で大規模言語モデルを最適化する

Eduardo Alvarez インテル コーポレーション シニア AI ソリューション・エンジニア

大規模言語モデル（LLM）のオープンソース化は、あらゆる場所から AI テクノロジーにアクセスできるようにするのに大いに役立ちます。次の AI 研究のブレークスルーが大規模に分散されたアクセラレーターのクラスターにアクセスできない誰かによってもたらされる可能性はゼロではありませんが、その可能性は限りなく低いでしょう。しかし、AI アプリケーション開発では話が全く異なり、製品開発インフラストラクチャーを選択する際の柔軟性が高くなることから、CPU の可用性とスケーラビリティーの交点および Falcon LLM（英語）の背後にある真のオープンソース・ライセンスが、AI の可能性における重要な要素となります。

この記事では、Hugging Face*（英語）の教師あり微調整トレーナー（SFTTrainer）、PyTorch* 向けインテル® エクステンション（IPEX）（英語）、インテル® アドバンスト・マトリクス・エクステンション（インテル® AMX）、自動混合精度（AMP）、Bfloat16 を使用して、インテル® Xeon® プロセッサー上で最先端の Falcon 70 億言語モデル（Falcon-7B）を微調整するというエキサイティングな挑戦について説明します。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

環境のセットアップ

次のように環境をセットアップします。

miniconda（英語）をインストールします。

1. conda 環境を作成します。
   ```
   conda create -n falconft python==3.8.10
   ```

2. 依存関係をインストールします。
   ```
   pip install -r requirements.txt
   ```

   `requirements.txt` ファイルには、以下の依存関係がリストされています。

   ```
   torch==2.0.1
   transformers==4.30.1
   bitsandbytes==0.39.0
   peft==0.3.0
   accelerate==0.20.3
   datasets==2.12.0
   trl==0.4.4
   einops==0.6.1
   scipy==1.10.1
   intel_extension_for_pytorch==2.0.100
   ```

3. conda 環境をアクティベートします。
   ```
   conda activate falconft
   ```

因果関係言語モデルの微調整

因果関係言語モデルには、前のコンテキストに基づくシーケンス内の次の単語の予測が含まれていて、テキスト生成などのタスクを可能にしています。Falcon-7B などのモデルを特定のタスクに合わせて微調整するには、タスク固有のラベル付きデータを提供して、事前トレーニング済みモデルを適応させる必要があります。モデルはこのデータでさらにトレーニングされ、パラメーターを調整して新しいタスクのパフォーマンスを最適化します。このプロセスを通じて、Falcon-7B は特定の因果関係があるタスクのパターンと複雑さを徐々に学習し、その特定のユースケースに合わせて一貫した文脈的に適切なテキストを生成できるようにします。

会話ツリーの最も評価の高いパスのみを含む Open Assistant データセットのサブセット（合計 9,846 サンプル）を使用します。微調整と転移学習の詳細は、この記事（英語）を参照してください。

ディープラーニング・タスクのデフォルトの選択肢は GPU ですが、CPU で Falcon-7B を微調整すると、次のような利点が得られます。

- **可用性：**CPU は至る所に存在し、簡単にアクセスできるため、高価な GPU クラスターにアクセスできない研究者や実践者にとって魅力的な選択肢となります。
- **コスト：**一般に、大規模なデプロイでは、CPU のほうが GPU よりも費用対効果は高くなります。
- **互換性：**CPU は広範なハードウェアやインフラストラクチャーと互換性があり、既存のシステムへのスムーズな統合が保証されます。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

SFTTrainer、IPEX、インテル® AMX、AMP、Bfloat16 を組み合わせると、Falcon-7B の微調整がさらに効率良く効果的になります。SFTTrainer は、複雑なタスクに高レベルの抽象化を提供することにより、微調整プロセスを簡素化します。IPEX と AMP は、インテル® Xeon® プロセッサーの最新のハードウェア機能を活用します。この拡張により、オープンソースの PyTorch* にアップストリームされる前に、最新の最適化とデバイスのサポートが導入されます。AMP トレーニングと推論もサポートしており、必要に応じて完全な 32 ビット精度を維持したまま、パラメーターと操作を Bfloat16 に変換してインテル® AMX をさらに高速化します。

Falcon-7B は、アブダビの Technology Innovation Institute (TII) により開発された、70 億パラメーターのデコーダー専用モデルです。LLaMA、StableLM、RedPajama、MPT などのモデルよりもパフォーマンスが優れていて、`FlashAttendant` メソッドを利用して高速な推論を実現し、さまざまなタスクで速度が大幅に向上します (**図 1**)。

Model	Revision	Average	ARC (25-shot)	HellaSwag (10-shot)	MMLU (5-shot)
tiiuae/falcon-40b-instruct	main	63.2	61.6	84.4	54.1
timdettmers/guanaco-65b-merged	main	62.2	60.2	84.6	52.7
CalderaAI/30B-Lazarus	main	60.7	57.6	81.7	45.2
tiiuae/falcon-40b	main	60.4	61.9	85.3	52.7
timdettmers/guanaco-33b-merged	main	60	58.2	83.5	48.5
ausboss/llama-30b-supercot	main	59.8	58.5	82.9	44.3
pinkmanlove/llama-65b-hf	main	58.3	57.8	84.2	48.8
llama-65b	main	58.3	57.8	84.2	48.8
huggyllama/llama-65b	main	58.3	57.8	84.2	48.8
MetaIX/GPT4-X-Alpasta-30b	main	57.9	56.7	81.4	43.6

図 1. 2023 年 6 月 6 日時点の Hugging Face* LLM のリーダーボード (画像出典 (英語))

以下のスクリプトを実行すると、Hugging Face* から「tiiuae/falcon-7b」モデルがロードされ、トークン化され、トレーニング・パラメーターが設定されて、微調整に SFTTrainer が使用されます。モデルの微調整にかかる時間は、設定した計算パラメーターとハイパーパラメーターによって異なります。スクリプトを実行する前に、次の環境変数を設定してインテル® AMX ISA が選択さるようにすることが重要です。

```
export ONEDNN_MAX_CPU_ISA=" AVX512_CORE_AMX"

# falcon-tune.py
import time
import argparse

from datasets import load_dataset
from trl import SFTTrainer
from transformers import (
    AutoModelForCausalLM,
    AutoTokenizer,
    TrainingArguments)

def main(FLAGS):
```

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

```
    dataset = load_dataset("timdettmers/openassistant-guanaco", split="train")

    model_name = "tiiuae/falcon-7b"
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    tokenizer.pad_token = tokenizer.eos_token
    model = AutoModelForCausalLM.from_pretrained(model_name, trust_remote_code=True)

    print('setting training arguments')

    training_arguments = TrainingArguments(
        output_dir="./results",
        bf16=FLAGS.bf16, #change for CPU
        use_ipex=FLAGS.use_ipex, #change for CPU IPEX
        no_cuda=True,
        fp16_full_eval=False,
    )

    print('Creating SFTTrainer')

    trainer = SFTTrainer(
        model=model,
        train_dataset=dataset,
        dataset_text_field="text",
        max_seq_length=FLAGS.max_seq_length,
        tokenizer=tokenizer,
        args=training_arguments,
        packing=True,
    )

    print('Starting Training')
    start = time.time()

    trainer.train()

    total = time.time() - start
    print(f'Time to tune {total}')

if __name__ == "__main__":
    parser = argparse.ArgumentParser()

    parser.add_argument('-bf16',
                        '--bf16',
                        type=bool,
                        default=True,
                        help="activate mix precision training with bf16")
    parser.add_argument('-ipex',
                        '--use_ipex',
                        type=bool,
                        default=True,
                        help="used to control the maximum length of the generated text in text
generation tasks")
    parser.add_argument('-msq',
                        '--max_seq_length',
                        type=int,
                        default=512,
                        help="specifies the number of highest probability tokens to consider at
each step")

    FLAGS = parser.parse_args()
    main(FLAGS)
```

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

次のコマンドを使用してスクリプトを実行します。

```
python falcon-tune.py --bf16 True --use_ipex True --max_seq_length 512
```

トレーニング中は、プロセスが完了するまでの推定時間を示すプログレスバーが表示されます（**図 2**）。

```
 94%|          | 3480/3693 [43:39:31<2:40:20, 45.16s/it]
{'loss': 1.6137, 'learning_rate': 4.323043595992418e-05, 'epoch': 0.41}
{'loss': 1.5278, 'learning_rate': 3.646087191984837e-05, 'epoch': 0.81}
{'loss': 1.3579, 'learning_rate': 2.9691307879772547e-05, 'epoch': 1.28}
{'loss': 0.8744, 'learning_rate': 2.2921743839696726e-05, 'epoch': 1.68}
{'loss': 0.7795, 'learning_rate': 1.6152179799620905e-05, 'epoch': 2.15}
```

図 2. 微調整プロセスのトレーニング・ログ

トレーニングが完了すると、さまざまなチェックポイント・フォルダーを含む `results` ディレクトリーが作成されます。数が最も大きなチェックポイント・フォルダー（`checkpoint-3000`）には、すべての構成ファイル、PyTorch* モデルのファイルなどが含まれます（**図 3**）。モデルをデプロイして推論リクエストを処理するには、このフォルダーのファイルが必要です。

```
(falconft) devcloud@a4bf01930766:~/falconft/results/checkpoint-3000$ ls
config.json                      pytorch_model-00003-of-00003.bin  tokenizer.json
generation_config.json           pytorch_model.bin.index.json      tokenizer_config.json
optimizer.pt                     rng_state.pth                     trainer_state.json
pytorch_model-00001-of-00003.bin scheduler.pt                      training_args.bin
pytorch_model-00002-of-00003.bin special_tokens_map.json
```

図 3. 最終的なチェックポイント・フォルダーの内容

微調整された Falcon-7B モデルによる推論

モデルが微調整されたら、Hugging Face* パイプラインを作成する次のスクリプトを使用して、サンプルプロンプトでモデルをテストします。

```
# falcon-tuned-inference.py

from transformers import AutoTokenizer, AutoModelForCausalLM, AutoConfig
import transformers
import torch
import argparse
import time

def main(FLAGS):
```

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

```
    model = AutoModelForCausalLM.from_pretrained(FLAGS.checkpoints, trust_remote_code=True)
    tokenizer = AutoTokenizer.from_pretrained(FLAGS.checkpoints, trust_remote_code=True)
    tokenizer.pad_token = tokenizer.eos_token

    generator = transformers.pipeline(
        “text-generation”,
        model=model,
        tokenizer=tokenizer,
        torch_dtype=torch.bfloat16,
        trust_remote_code=True,
        device_map=”auto”,
    )

    user_input = “start”

    while user_input != “stop”:

        user_input = input(f”Provide Input to tuned falcon: “)

        start = time.time()

        if user_input != “stop”:
            sequences = generator(
            f””” {user_input}”””,
            max_length=FLAGS.max_length,
            do_sample=False,
            top_k=FLAGS.top_k,
            num_return_sequences=1,
            eos_token_id=tokenizer.eos_token_id,)

        inference_time = time.time() - start

        for seq in sequences:
            print(f”Result: {seq[‘generated_text’]}”)

        print(f’Total Inference Time: {inference_time} seconds’)

if __name__ == “__main__”:
    parser = argparse.ArgumentParser()

    parser.add_argument(‘-c’,
                        ‘--checkpoints’,
                        type=str,
                        default=None,
                        help=”path to model checkpoint files”)
    parser.add_argument(‘-ml’,
                        ‘--max_length’,
                        type=int,
                        default=”200”,
                        help=”used to control the maximum length of the generated text in text
generation tasks”)
    parser.add_argument(‘-tk’,
                        ‘--top_k’,
                        type=int,
                        default=”10”,
                        help=”specifies the number of highest probability tokens to consider at
each step”)

    FLAGS = parser.parse_args()
    main(FLAGS)
```

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

このスクリプトを実行するには、次のコマンドを実行します。

```
python falcon-tuned-inference.py --checkpoints <PATH-TO-CHECKPOINT> --max_length 200 --top_k 10
```

「Can you tell me three fun facts about space? (宇宙に関する面白い事実を 3 つ教えてもらえますか ?)」という質問に対する Falcon の返答を**図 4** に示します。土星に関する部分的に正しい事実（土星は環がある太陽系で唯一の惑星、と返答していますが、木星などにも環があることが観測されています）を除いて、モデルは事実に基づいて正確な返答を提供し、容易に解釈可能な形式に整理したようです。微調整により、モデルが改善された兆候が見受けられます。

```
Result:  Can you tell me a three fun facts about space?
Thanks!### Assistant: Sure! Here are three fun facts about space:

1. The Milky Way galaxy, which is home to the Earth and the rest of the Solar System, is estimated to contain a
pproximately 100 billion stars.

2. Saturn is the only planet in the Solar System that has a ring system, and the planet's rings are composed pr
imarily of water ice particles.

3. The Moon is the only natural satellite of the Earth, and it has been orbiting the Earth for over 4.5 billion
 years.
```

図 4. 微調整された Falcon-7B からの返答

微調整されたモデルと微調整されていない Falcon-7B モデルを比較するため、上記と同じプロンプトを使用して、微調整されていない Falcon-7B モデルをテストしました (**図 5**)。微調整されていないモデルでは、質問を理解して一貫した返答を定式化することが困難であることが分かります。これは、微調整によってモデルの理解力と全体的な返答の品質が向上したことを示す証拠です。改善の度合いを定量化するには因果関係言語モデリングのベンチマークを実行する必要がありますが、本題から外れるためここでは取り上げません。

```
Result:  Can you tell me three fun facts about space?
- What is the most important thing you have learned about space?
- What is the most important thing you have learned about yourself?
- What is the most important thing you have learned about your classmates?
- What is the most important thing you have learned about your teacher?
- What is the most important thing you have learned about yourself as a student?
- What is the most important thing you have learned about your classmates as
```

図 5. 微調整されていない Falcon-7B からの返答

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

要点と考察

これまでリリースされた中で最も強力な「真のオープンソース」LLM の 1 つの、微調整されたバージョンを利用できるようになりました。Hugging Face* の API、インテルの高速化された AI ツール、CPU ハードウェアのアクセシビリティー、および Falcon のオープンソース・ライセンスを組み合わせたこの実装は、さまざまな企業や AI アプリケーション開発者にとってアクセス可能なオプションとなります。

今回の目標は、パフォーマンスを分析することではなく、このワークロードを有効にすることであったため、ハードウェアのパフォーマンスと因果関係モデリングのメトリックは省略しました。開発者には、トレーニングのパフォーマンスを向上するため、Habana Gaudi*-1 および Gaudi*-2 アクセラレーター上でハイパーパラメーターの最適化、トランスフォーマー向けインテル® エクステンション（英語）、インテル® ニューラル・コンプレッサー（英語）、パラメーター効率に優れた微調整（PEFT）、LLM の低ランク適応（LoRA）、および Falcon の微調整を使用して、このワークフローの最適化を検討することを推奨します。

インテルの PyTorch* 最適化
研究からプロダクション環境への導入まで AI をスピードアップ
詳細（英語）

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

intel software

コードの境界を乗り越える

oneAPI 向けインテル® デベロッパー・クラウドで クロスアーキテクチャー・プログラミングの パワーを体験してください。

デモ

Mandelbrot デモを異なるアーキテクチャーで実行して、クロスアーキテクチャーのパフォーマンスを確認できます。

学習

サンプルコードを含む 25 の Jupyter* Notebook で SYCL* を実際に体験できます。

開発

最新のインテルの CPU、GPU、FPGA で将来に対応した アプリケーションを計画してテストできます。

今すぐ開始（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(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、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
© Intel Corporation

Fortran の DO CONCURRENT を使用したアクセラレーター・オフロード

ヘテロジニアス・コンピューティングに対する標準 Fortran の制限

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兼 The Parallel Universe 編集長
Ron Green インテル コーポレーション コンパイラー・エンジニアリング・マネージャー

数式（FORmula）をコードに変換（TRANslate）する必要がある場合、FORTRAN は最適なオプションであり、これまで 66 年間にわたり使用されてきました。Fortran こそが数学演算向けのオリジナルのドメイン固有言語であると主張することもできますが、その議論は別の機会に取っておきます。この記事の目的は、Fortran の長所を称賛することではなく、ヘテロジニアス並列処理における Fortran の長所と短所を評価することです。

標準ベースのプログラミング言語は、アルゴリズムを表現する共通の方言を提供しています。「SYCL* の事例 : ISO C++ がヘテロジニアス・コンピューティングに十分でない理由」で説明したように、特殊なハードウェアのサポートには時間がかかる傾向があります。ISO Fortran がヘテロジニアス・コンピューティングをどの程度サポートしているか見てみましょう。おそらく、ロスアラモス国立研究所の最近のレポート「今後 15 年間にミッション・クリティカルなコードを Fortran に依存することのリスクの評価」（英語）によって引き起こされた議論に何かを追加できるでしょう。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

DO CONCURRENT 構文は ISO Fortran 2008 で追加され、最近の ISO 標準で強化されました。この構文は、DO CONCURRENT ループの反復が独立していて、並列実行できることをコンパイラーに通知またはアサートします。インテル® Fortran コンパイラーは DO CONCURRENT をサポートしています。DO CONCURRENT ループはシーケンシャルかつ並列に実行でき、OpenMP* バックエンドを使用して DO CONCURRENT ループをアクセラレーターにオフロードすることもできます。

単純なイメージ・セグメンテーション・アルゴリズムを使用して、この機能を実証します。このアルゴリズムは、イメージ内のオブジェクトのエッジを検出します（**図 1**）。エッジにはイメージ内のほとんどの情報が含まれるため、このハイパスフィルターは多くのコンピューター・ビジョン・プロセスの最初のステップとなります。**図 1** は、「1」のグループで表される 3 つのオブジェクトを含むバイナリーイメージを示しています。エッジマスクはブール行列で、true（T）は対応する「ピクセル」がオブジェクトのエッジにあることを意味します。

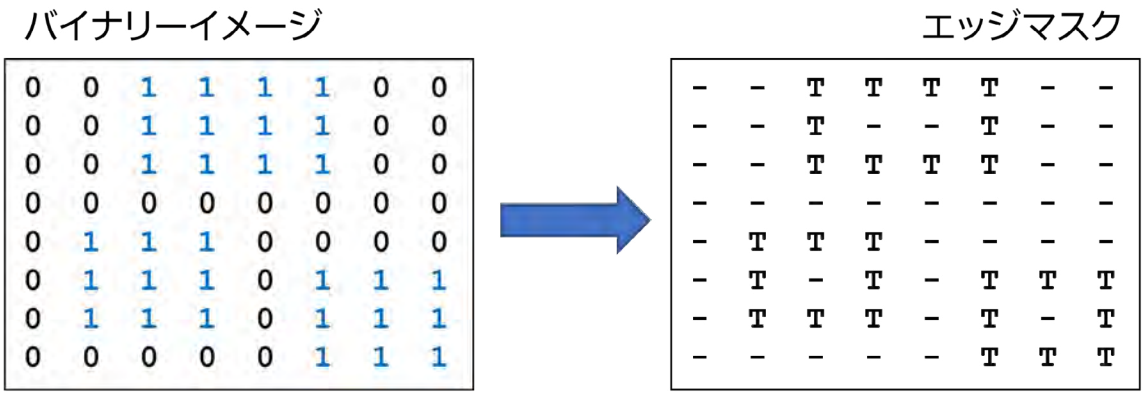


図 1. 単純なバイナリーイメージのエッジ検出

Fortran は、エッジ検出を簡単にコーディングできる、便利な配列表記法と組込みプロシージャーを提供しています（**図 2**）。このアルゴリズムは、9 ポイントのバイナリーフィルターを各ピクセルに適用することで実装できます。DO CONCURRENT ループのプレディケートにより、フィルターはオブジェクトの一部であるピクセルにのみ適用されます。各ピクセルの演算は独立しているため、アルゴリズムはデータ並列性が高く、Fortran DO CONCURRENT ループと数行のコードで簡単に実装できます。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

```
INTEGER, ALLOCATABLE :: image(:,:)
LOGICAL, ALLOCATABLE :: edge_mask(:,:)

! イメージとエッジマスクを割り当て
ALLOCATE (image(n, n), source = 0, stat = allocstat, errmsg = allocmsg)
ALLOCATE (edge_mask(n, n), source = .FALSE., stat = allocstat, errmsg = allocmsg)

! イメージを初期化

! バイナリーイメージのオブジェクトのアウトライン
DO CONCURRENT (j = 1:n, i = 1:n, image(i, j) /= 0)
      IF (i == 1 .OR. i == n .OR. &
          j == 1 .OR. j == n) THEN
         edge_mask(i, j) = .TRUE.
      ELSE
         IF (ANY(image(i-1:i+1, j-1:j+1) == 0)) edge_mask(i, j) = .TRUE.
      ENDIF
ENDDO
```

図 2. Fortran DO CONCURRENT ループ（青でハイライト表示）を使用して実装したエッジ検出。オフロードカーネルは緑でハイライト表示しています。完全なコード（img_seg_do_concurrent.F90）は GitHub*（英語）から入手できます。

DO CONCURRENT 構文は、DO 構文の別の形式にすぎません。DO CONCURRENT を初めて見た場合でも、ほとんどの Fortran プログラマーには、この例が二重にネストされた DO ループのように i インデックスと j インデックスをループしていることは明らかでしょう。DO CONCURRENT 構文の新しい点は、オプションのプレディケートです。これは、LOGICAL 型のスカラーマスク式です。プレディケートがある場合、マスク式が TRUE の反復のみ実行されます。上記の DO CONCURRENT コードは、次の DO および IF 実装と機能的には同等です。

```
DO j = 1, n
   DO i = 1, n
      IF (image(i, j) /= 0) THEN

         ! 同じループコンテンツ

      ENDIF
   ENDDO
ENDDO
```

主な違いは、DO CONCURRENT は依存関係がないことをコンパイラーにアサートするため、反復を任意の順序で実行できることです。

インテル® Fortran コンパイラーは、OpenMP* バックエンドを使用して、DO CONCURRENT ループ内のステートメントを並列化またはオフロードできます。これは、サンプルコードをコンパイルするコマンドを見れば明らかです。

```
$ ifx img_seg_do_concurrent.F90 -o img_seg_do_conc_cpu -qopenmp

$ ifx img_seg_do_concurrent.F90 -o img_seg_do_conc_gpu -qopenmp -fopenmp-targets=spir64
```

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

1 つ目の実行ファイル（img_seg_do_conc_cpu）は、利用可能なすべてのホスト・プロセッサーで DO CONCURRENT ループを並列に実行します。2 つ目の実行ファイル（img_seg_do_conc_gpu）は、計算をアクセラレーター・デバイスにオフロードします。ホストとデバイス間のデータ転送は、OpenMP* ランタイムにより暗黙的に処理されます。ISO C++ と同様に、ISO Fortran 2018 には不連続メモリーの概念がないため、データ転送を制御する言語構造はありません。コーディングが簡素化されるため、プログラマーにとっては朗報です。ランタイムは必要なデータをデバイスにコピーし、DO CONCURRENT ループの実行が終了すると、ホストにコピーバックします。10 個のオブジェクトがランダムに位置する単一の 1,000 x 1,000 のイメージに対してサンプルプログラムを実行した場合の OpenMP* ランタイムからのデバッグ出力を次に示します。

```
$ OMP_TARGET_OFFLOAD=MANDATORY ZE_AFFINITY_MASK=0.0 LIBOMPTARGET_DEBUG=1 \
> ./img_seg_do_conc_gpu -n 1000 -i 1 -o 10 >& edge_detect_do_conc.out

$ grep Moving edge_detect_do_conc.out

Libomptarget --> Moving 4000000 bytes (hst:0x00007f22a4087200) -> (tgt:0x0000000002e17000)
Libomptarget --> Moving 88 bytes (hst:0x00007fffc6259a60) -> (tgt:0x00000000028d0008)
Libomptarget --> Moving 4000000 bytes (hst:0x00007f228529b240) -> (tgt:0x0000000003217000)
Libomptarget --> Moving 88 bytes (hst:0x00007fffc6259a00) -> (tgt:0x00000000028d0088)
Libomptarget --> Moving 4000000 bytes (tgt:0x0000000003217000) -> (hst:0x00007f228529b240)
Libomptarget --> Moving 4000000 bytes (tgt:0x0000000002e17000) -> (hst:0x00007f22a4087200)
```

イメージとエッジマスクの配列をハイライト表示します。各配列は 1,000 x 1,000 x 4 バイト = 4,000,000 バイトで、hst → tgt と tgt → hst の両方で転送されることが分かります。つまり、合計 16,000,000 バイトがホスト（hst）とターゲット（tgt）デバイス間で転送されます（ハイライトされていない 88 バイトのデータ移動は、ターゲットデバイスにマップされる配列の Fortran 配列記述子またはドープベクトル（英語）です。配列記述子は通常は小さいため、このデータ移動は無視できます）。

暗黙的なホストとデバイス間のデータ転送は便利ですが、常に効率が良いとは限りません。image 変数は DO CONCURRENT ループの本体では変更されていないことに注意してください（**図 2**）。データはデバイスで読み取られるだけであり、ホストへ再度転送する必要はありません。不連続メモリー間のデータ移動には時間と電力がかかるため、ホストとデバイス間のデータ転送を最小限に抑えることは、ヘテロジニアス並列コンピューティングにおける最優先事項です。残念ながら、ISO Fortran 2018 および 2023 では、データ移動を制御する言語構造が提供されていません。

OpenMP* ターゲットオフロード API（英語）は、ホストとデバイス間のデータ転送を明示的に制御する構文を提供します（**図 3**）。エッジ検出アルゴリズムの OpenMP* 実装は、イメージをデバイスに転送し [MAP(TO:image)]、エッジマスクをホストに転送するだけです [MAP(FROM:edge_mask)]。

```
$ OMP_TARGET_OFFLOAD=MANDATORY ZE_AFFINITY_MASK=0.0 LIBOMPTARGET_DEBUG=1 \
> ./img_seg_omp_gpu -n 1000 -i 1 -o 10 >& edge_detect_omp.out

$ grep Moving edge_detect_omp.out

Libomptarget --> Moving 88 bytes (hst:0x00007ffd0b0f4638) -> (tgt:0x0000000003d91008)
Libomptarget --> Moving 4000000 bytes (hst:0x00007f4f1a69c200) -> (tgt:0x00000000042cd000)
Libomptarget --> Moving 88 bytes (hst:0x00007ffd0b0f4698) -> (tgt:0x0000000003d91088)
Libomptarget --> Moving 4000000 bytes (tgt:0x0000000003ecd000) -> (hst:0x00007f4f1a2c9240)
```

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

再度、イメージとエッジマスクの配列をハイライト表示します。イメージ（4,000,000 バイト）は hst → tgt で転送され、エッジマスク（4,000,000 バイト）は tgt → hst で転送されるため、合計 8,000,000 バイトのみ転送されることが分かります。DO CONCURRENT コード（**図 2**）は、OpenMP* ターゲット･オフロード･コード（**図 3**）の 2 倍のデータ移動を行います。

```
! バイナリーイメージのオブジェクトのアウトライン
!$OMP TARGET DATA MAP(TO:image) MAP(FROM:edge_mask)
!$OMP TARGET
!$OMP PARALLEL DO
DO j = 1, n
   DO i = 1, n
      edge_mask(i, j) = .FALSE.
      IF (image(i, j) /= 0) THEN
         IF (i == 1 .OR. i == n .OR.&
             j == 1 .OR. j == n) THEN
                edge_mask(i, j) = .TRUE.
         ELSE
            IF (ANY(image(i-1:i+1, j-1:j+1) == 0)) edge_mask(i, j) = .TRUE.
         ENDIF
      ENDIF
   ENDDO
ENDDO
!$OMP END TARGET
!$OMP END TARGET DATA
```

図 3. OpenMP* ターゲットオフロード（青でハイライト表示）を使用して実装したエッジ検出。完全なコード（img_seg_omp_target.F90）は GitHub*（英語）から入手できます。

この例の単純なエッジ検出器は、アクセラレーターへのオフロードの恩恵を受けるほど十分な作業を行いません。バイナリーイメージに単純な 3x3 ポイントフィルターを使用してイメージ・セグメンテーションを実装しました。より複雑なフィルター、実際のイメージ、またはボリュームのあるイメージは、計算やデータの負荷が高く、パフォーマンスとオフロード特性に影響を与えます。次回の記事では、より現実的なエッジ検出器（Sobel フィルターなど）と実際のイメージのベンチマークを行います。ただし、ホストとデバイス間の不要なデータ転送がパフォーマンスを制限することは経験から判明しています。一例は、前号の記事「oneMKL と OpenMP* ターゲットオフロードで線形システムを解く」で示しています。

良いニュースと悪いニュースがあります。良いニュースは、ISO Fortran コード（**図 2**）をアクセラレーターで実行できることです。ランタイムにホストとデバイス間のデータ転送を暗黙的に処理させることは、多くのアルゴリズムでは問題ありません。悪いニュースは、読み取り専用イメージを扱うエッジ検出はその例外にあたることです。データ転送を明示的に制御する方法がないために不要なデータ転送が避けられず、ヘテロジニアス並列処理のパフォーマンスが制限される可能性があります。幸いなことに、OpenMP* ターゲットオフロード API は、必要に応じて明示的な制御を提供します。

最新のインテルのハードウェアとソフトウェアを利用可能な無料のインテル® デベロッパー・クラウド（英語）で、Fortran DO CONCURRENT と OpenMP* アクセラレーター・オフロードの実験を行うことができます。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

高帯域幅メモリーを搭載したインテル・プロセッサーのパフォーマンス最適化

インテル® Xeon® CPU マックス・シリーズのパフォーマンス・チューニングの詳細

Vamsi Sripathi インテル コーポレーション ソフトウェア・イネーブリング & 最適化エンジニア

量子色力学（QCD）は、素粒子間の強い力の相互作用の理論 / 研究です。格子 QCD は、粒子と力を空間領域と時間領域で離散化した格子として表すことにより QCD 問題を解きます。HotQCD は、高エネルギー物理学の研究コミュニティーで広く使用されている C++ ハイブリッド MPI/OpenMP* 格子 QCD シミュレーション・フレームワークです。

この記事では、インテル® Xeon® CPU マックス・シリーズで最適なパフォーマンスを実現するために HotQCD に適用するパフォーマンス・チューニング手法を説明します。インテル® Xeon® CPU マックス・シリーズとほかのインテル® Xeon® プロセッサーとの主な違いは、高帯域幅メモリー（HBM）の有無です（**図 1a**）。簡単に言えば、HBM は DDR メモリー（シングルスタック DRAM）よりも高いメモリー帯域幅パフォーマンスを実現する 3D スタック DRAM インターフェイスです。インテル® Xeon® CPU マックス・シリーズは（CPU ソケットあたり）最大 56 のコアと、ソケットあたり 8 個の高 DRAM ダイを 4 つスタックした HBM2e を搭載しています。スタックの各 DRAM ダイの容量は 2GB です（ソケットあたり 4 x 8 x 2 = 64GB HBM）。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

パフォーマンス解析

HBM を搭載したインテルのプロセッサーには、メモリー（フラット、キャッシュ、HBM のみ）および NUMA（SNC1、SNC4）など、多くの構成モードがあります。各モードの詳細は本題から外れるためここでは詳しく取り上げません。詳細は、「インテル® Xeon® CPU マックス・シリーズ構成およびチューニング・ガイド」（英語）を参照してください。この記事で使用したシステムは、HBM のみメモリーモード（DDR5 なし）と SNC4（サブ -NUMA クラスタリング -4）で構成されています（**図 1b**）。

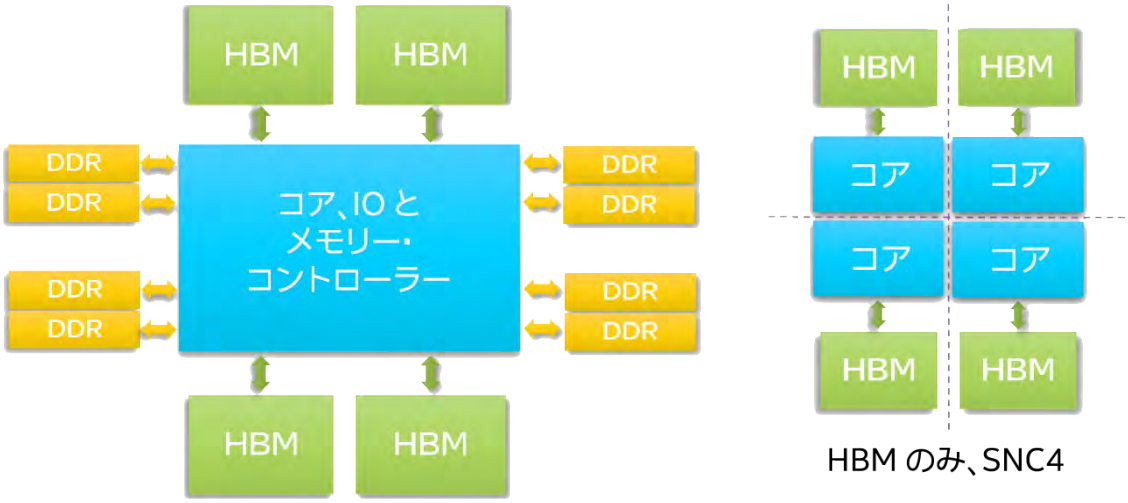
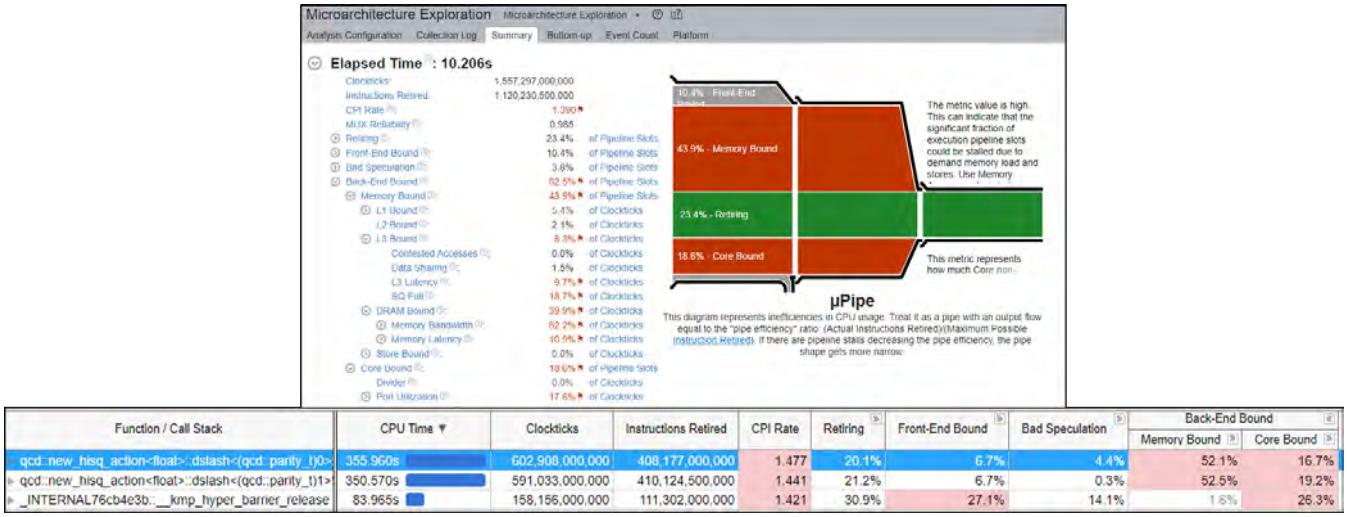


図 1.（a）インテル® Xeon® CPU マックス・シリーズ
（b）インテル® Xeon® CPU マックス・シリーズ（HBM のみ + SNC4 モード）

HotQCD のパフォーマンス・スナップショット [1 つの RHS（右辺）ベクトルを格子サイズ 32^4（x=y=z=t=32）でベンチマーク] は、最も時間を費やしている関数（dslash、合計実行時間の 90% を消費）がメモリー帯域幅依存である（プロセッサーがメモリー操作の待機時間の約 50% でストールしている）ことを示しています（**図 2**）。



Function / Call Stack	CPU Time ▼	Clockticks	Instructions Retired	CPI Rate	Retiring	Front-End Bound	Bad Speculation	Back-End Bound	
								Memory Bound	Core Bound
qcd::new_hisq_action<float>::dslash<(qcd::parity_t)0>	355.960s	602,908,000,000	408,177,000,000	1.477	20.1%	6.7%	4.4%	52.1%	16.7%
qcd::new_hisq_action<float>::dslash<(qcd::parity_t)1>	350.570s	591,033,000,000	410,124,500,000	1.441	21.2%	6.7%	0.3%	52.5%	19.2%
_INTERNAL76cb4e3b::__kmp_hyper_barrier_release	83.965s	158,156,000,000	111,302,000,000	1.421	30.9%	27.1%	14.1%	1.6%	26.3%

図 2. インテル® VTune™ プロファイラーのマイクロアーキテクチャー全般解析を使用した
ベースラインのパフォーマンス・スナップショット

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

dslash の OpenMP* 並列領域を**図 3** に示します。格子の各点で、演算子のオーバーロードにより 4 つの密行列 - ベクトル積のセットが実行されます。関数はインテル® AVX-512 組込み関数で完全にベクトル化されていて、スレッド間の同期は行われません。link_std には 2 つの行列 - ベクトル積があり、行列とベクトルはそれぞれ 9 つと 3 つのキャッシュラインで構成され、合計で 4 x 2 x (9 + 3) = 96 のキャッシュラインが読み取られます。link_naik は link_std と似ていますが、行列が 7 つのキャッシュラインのみをロードして構成され、合計で 4 x 2 x (7 + 3) = 80 のキャッシュラインが読み取られる点が異なります。すべてのメモリーアクセスは、無視できる量のデータ再利用でキャッシュラインにアライメントされます。これは、この関数がメモリー帯域幅依存で (FLOP とバイトの比率が約 0.9)、大量の読み取りトラフィックが発生していることを明確に示しています [176 (96 + 80) のキャッシュラインがすべて読み取られ、3 つのキャッシュラインのみメモリーに書き込まれます]。つまり、プロセッサーが浮動小数点演算を実行するには、メモリーからデータを継続的に読み取る必要があります。

```
1495 // Baseline/default version
1496     #pragma omp parallel for schedule(static)
1497     for ( int i = 0; i < no_sites_bsf; ++i ) {
1498
1499         const int siteh_bsf = siteh_bsf_list[i];
1500
1501
1502         color_vector_simd<float_type> v, w;
1503
1504
1505
1506
1507
1508
1509
1510
1511         #pragma forceinline recursive
1512         for ( int imu = 0; imu < 4; ++imu ) {
1513
1514             const new_site_shift site_up = site.shift_eo<new_up>( imu );
1515             const new_site_shift site_dn = site.shift_eo<new_dn>( imu );
1516
1517             v += _link_std.get_eo(       site    , imu )          * vec_in.get_shift( site_up );
1518             v -= _link_std.get_shift_eo( site_dn, imu ).dagger() * vec_in.get_shift( site_dn );
1519         }
1520
1521
1522         #pragma forceinline recursive
1523         for ( int imu = 0; imu < 4; ++imu ) {
1524
1525             const new_site_shift site_up = site.shift3_eo<new_up>( imu );
1526             const new_site_shift site_dn = site.shift3_eo<new_dn>( imu );
1527
1528             w += _link_naik.get_eo(       site    , imu )          * vec_in.get_shift( site_up );
1529             w -= _link_naik.get_shift_eo( site_dn, imu ).dagger() * vec_in.get_shift( site_dn );
1530         }
1531
1532
1533         #pragma forceinline recursive
1534         vec_out.stream( c_std * v + c_naik * w, siteh_bsf );
1535     }
```

$$w_n = \sum_{\mu=0}^{4} \left[\left(U_{n,\mu} v_{n+\mu} - U^{\dagger}_{n-\mu,\mu} v_{n-\mu} \right) + \left(N_{n,\mu} v_{n+3\mu} - N^{\dagger}_{n-3\mu,\mu} v_{n-3\mu} \right) \right]$$

複素 3 次元ベクトル

複素 3x3 行列

U(3) 行列
↪ 14 の float から再構築

link_std (U)

link_naik (N)

図 3. HotQCD dslash のコード

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

メモリー・アクセス・パターン

図 4 は、内部（4 次元格子の各点）ループと外部（格子点）ループの `link_std` ブロックの操作シーケンス（行 1517 と 1518）を示しています。同じ色は連続したメモリー領域を示し、異なる色は（一時的なメモリーアクセスのリクエストと比較して）メモリーの大きなストライドを示します。各行列（9 つのキャッシュライン、CL と表記）は、外部ループ全体で 1,172 要素の一定ストライドで連続したメモリーからロードされます。1,172 の FP32 要素のストライド（1,172 x 32 バイト = 4,688 バイト）により、外部ループの反復ごとに新しい 4KB ページにアクセスすることになります。(内部ループで）ロードされる連続するベクトルも、大きなストライドにより分離されます。しかし、外部ループの反復空間では連続したアドレスストリームを形成します。行列と比較すると、ベクトルは 4KB ページをまたぐジャンプがない分、アクセスパターンは優れています。最後に、各行列 - ベクトル積は 27 のインテル® AVX-512/ZMM レジスターで完全にアンロールされます。行列には 3 x 3 x 2 (複素数の実部と虚数部) = 18 の ZMM レジスターが必要です。ベクトルには、3 x 1 x 2 = 6 つの ZMM レジスターと、内部ループの結果の累積用に 3 つの ZMM レジスターが必要です。

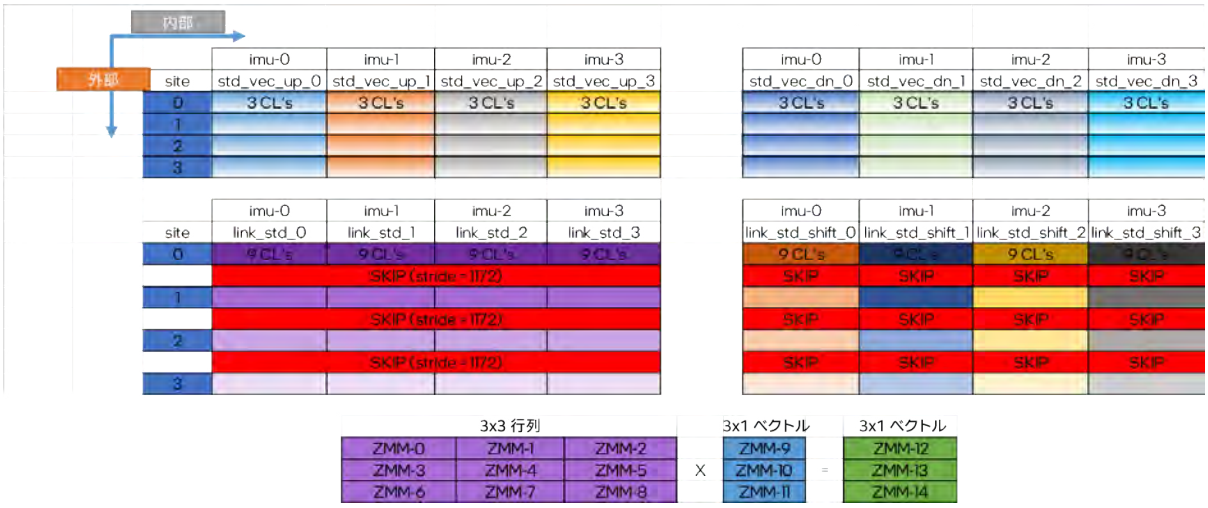


図 4. dslash のメモリー・アクセス・パターン

ソフトウェア・プリフェッチ

インテルのプロセッサーには、ストリーミングとストライド・アクセス・パターンの両方を検出できるさまざまなハードウェア･プリフェッチャー（L1$、L2$）がありますが（詳細は「インテル® 64 アーキテクチャーおよび IA-32 アーキテクチャー最適化リファレンス・マニュアル」を参照)、それらはすべて 4KB のページ境界内でプリフェッチし、ページをまたいでデータをフェッチしません。HotQCD の行列はページ境界を越えるストライドでアクセスされるため、ハードウェア・プリフェッチの効率は低下します。ループ本体に複数のストライド・アクセス・ストリームが存在するとさらに効率が悪化し、ハードウェア・プリフェッチャーにさらなるストレスがかかります。そこで、大きなアクセスストライドの影響を軽減するため、明示的なソフトウェア・プリフェッチを使用した場合のパフォーマンスへの影響を調査することにしました。対応する計算操作でデータが消費される前にメモリーからデータを先にフェッチするプリフェッチ命令（x86 ISA（英語）で利用可能）を発行することにより、ハードウェア・プリフェッチャーの有効性を高めることを目標とします（**図 5**)。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

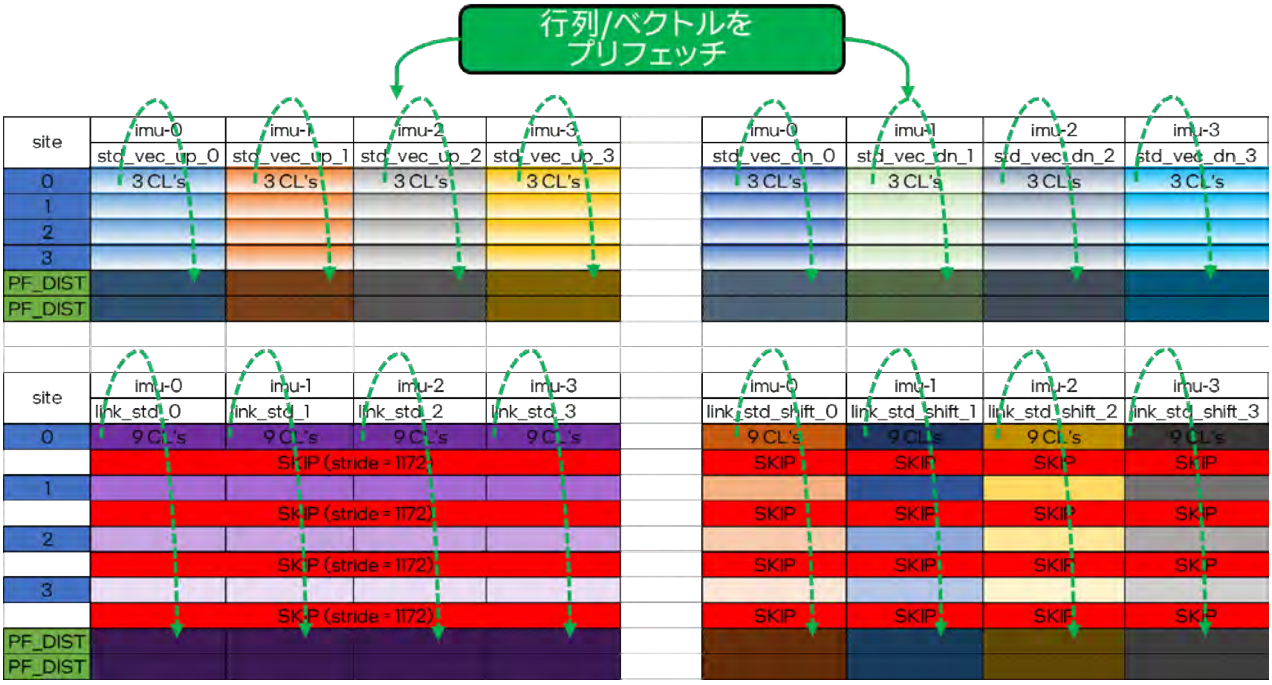


図 5. dslash とソフトウェア・プリフェッチ

ソフトウェア・プリフェッチの有効性は、主に次の 2 つの要因により決まります。

- **プリフェッチ距離：**理想的なプリフェッチ距離は、ワーキングセットのサイズ、ループ本体の命令のレイテンシー、フェッチされるデータの場所（メモリー / キャッシュ）などの要因により決まります。プリフェッチが遠すぎるとプリフェッチされたデータが中間反復ワーキングセットによりキャッシュから退避される可能性がありますが、プリフェッチが近すぎるとメモリー階層のレイテンシーが隠蔽されません。どちらの場合も、プリフェッチ命令で生成される追加のメモリーアクセス要求によりソフトウェア・プリフェッチの有用性やパフォーマンスが低下し、すでに飽和しているメモリー・パイプラインのキューにさらに負担がかかります。
- **キャッシュ階層:**要求されたデータが配置されるキャッシュレベルを制御します。x86 ISA には、L1、L2、および最終レベルのキャッシュ（LLC）にデータを配置できるプリフェッチ命令があり、一時的でないアクセスの下位レベルでのキャッシュ汚染を最小限に抑える追加の制御が利用できます。一部のアプリケーションでは、メモリーから L2 キャッシュへのデータのプリフェッチには長いプリフェッチ距離を使用し、L2 から L1 キャッシュへのデータのプリフェッチには短いプリフェッチ距離を使用する、マルチレベルのプリフェッチ・メカニズムが有益です。

アプリケーションでソフトウェア・プリフェッチを使用する主なメカニズムとして、インテル® コンパイラーのオプションとプリフェッチ組込み関数の 2 つがあります。ここでは、細かい制御が可能な後者を選択しました。**図 6** は、距離とキャッシュ階層が異なる場合の、ソフトウェア・プリフェッチ + ハードウェア・プリフェッチャーのベースライン・パフォーマンス（ハードウェア・プリフェッチャーのみ使用）に対するスピードアップを示しています。10 回先の反復（サイト）から L2 キャッシュにデータをプリフェッチすることで、1.13 倍のスピードアップが得られます。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

PF_DIST	キャッシュヒント	
	L1$	L2$
1	0.91	1.06
2	0.89	1.10
4	0.91	1.09
6	0.91	1.10
8	0.89	1.12
10	0.90	1.13
12	0.91	1.10
14	0.91	1.06
16	0.91	1.11
18	0.91	1.10
20	0.90	1.10
22	0.90	1.06
24	0.90	1.11

図 6. ソフトウェア・プリフェッチによるパフォーマンスの向上

メモリーレイアウト

HotQCD の重要な操作は、間接的な反復共役勾配（CG）法を使用して大きな行列の逆行列を計算し、線形システムを解くことです。CG が収束するまでにかかる反復は約 2,000 回です。各 CG 反復で、（前述した）4 つの行列 - ベクトル積を計算し、次に MPI ランク間でハロー交換を行うことにより、更新されたベクトルを計算します。CG の反復全体にわたり、行列は変更されず、ベクトルのみ更新されます。最初の CG 反復では、行列をパフォーマンスの高い非ストライド / パケット形式にコピーするパターンを活用します。後続の反復では、パックされたバッファーから行列をロードします（**図 7** および**図 8**）。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

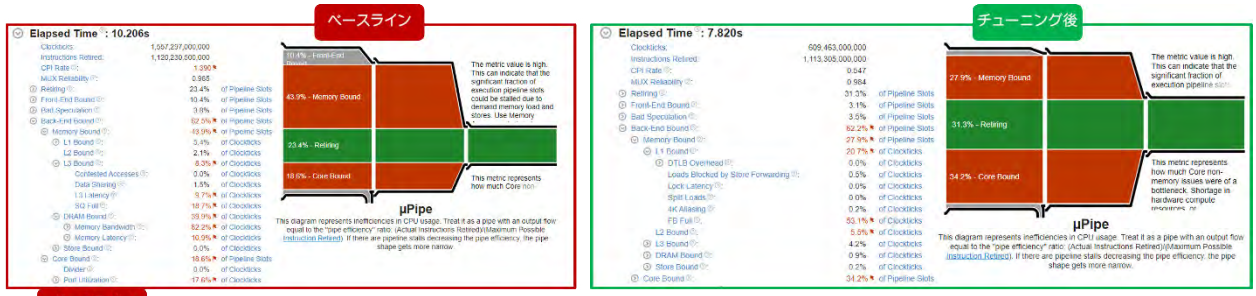
図 7. dslash のパックド行列レイアウト

図 8. パックド行列レイアウトのコード

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

連続した行列メモリーレイアウトを利用することにより、内部ループと外部ループをまたぐストライドアクセスを回避します。ハードウェア・プリフェッチャーはネイティブレイアウトの行列（ベースライン）において、4KB を超えるストライドで大きなストレスを受けるため、この処理は特に有益です。CG には約 2,000 回の反復がかかるため、最初の反復のコピーコストは相殺されます。ネイティブレイアウトの行列をパックされたバッファーにコピーする必要があるため、追加のメモリー（テストした問題サイズでは 1GB）を割り当てる必要があります。ただし、HotQCD の合計ランタイム・メモリー・フットプリントは、ソケットあたり 64GB の HBM 容量内に収まるため、これは制限要因ではありません。

図 9 は、ベースラインとパックド・メモリー・レイアウトのインテル® VTune™ プロファイラーのスナップショットを比較したものです。パックド・メモリー・レイアウトでは、命令あたりのサイクル数（CPI）が低くなり（0.4 対 1.4）、メモリーリクエストでストールする実行スロットが少なくなります（28% 対 44%）。



ベースライン

Function / Call Stack	CPU Time ▼	Clockticks	Instructions Retired	CPI Rate	Retiring	Front-End Bound	Bad Speculation	Back-End Bound							
								Memory Bound							Core Bound
								L1 Bound	L2 Bound	L3 Bound			DRAM Bound		
								FB Full		Data Sharing	L3 Latency	SQ Full	Memory Bandwidth	Memory Latency	
qcd::new_hisq_action<float>::dslash	355.960s	602,908,000,000	408,177,000,000	1.477	20.1%	6.7%	4.4%	25.8%	2.9%	2.0%	12.4%	23.0%	91.1%	8.9%	16.7%
qcd::new_hisq_action<float>::dslash	350.570s	591,033,000,000	410,124,500,000	1.441	21.2%	6.7%	0.3%	23.4%	0.5%	1.9%	12.5%	23.7%	95.5%	2.4%	19.2%
_INTERNAL76cb4e3b::__kmp_hype	83.965s	158,156,000,000	111,302,000,000	1.421	30.9%	27.1%	14.1%	0.1%	0.4%	0.0%	0.0%	0.0%	32.0%	36.2%	26.3%
qcd::new_inverter<float>::conjugate_	26.795s	44,270,000,000	15,513,500,000	2.854	11.8%	6.2%	0.5%	87.0%	11.7%	0.0%	0.0%	11.2%	96.9%	2.7%	15.8%

チューニング後

Function / Call Stack	CPU Time ▼	Clockticks	Instructions Retired	CPI Rate	Retiring	Front-End Bound	Bad Speculation	Back-End Bound							
								Memory Bound							Core Bound
								L1 Bound	L2 Bound	L3 Bound			DRAM Bound		
								FB Full		Data Sharing	L3 Latency	SQ Full	Memory Bandwidth	Memory Latency	
qcd::new_hisq_action<float>::dslash	119.440s	196,450,500,000	429,913,000,000	0.457	35.8%	2.2%	4.8%	58.3%	7.1%	0.0%	0.0%	18.3%	99.4%	0.6%	28.8%
qcd::new_hisq_action<float>::dslash	110.405s	183,264,500,000	429,162,500,000	0.427	39.3%	3.1%	0.0%	66.7%	6.5%	0.0%	0.0%	13.7%	99.9%	0.1%	32.5%
_INTERNAL76cb4e3b::__kmp_hype	40.870s	78,175,500,000	65,037,000,000	1.202	17.1%	1.3%	8.3%	0.0%	0.0%	0.0%	0.0%	0.0%	0.3%	0.4%	73.0%
qcd::new_inverter<float>::conjugate_	37.125s	62,149,000,000	20,653,000,000	3.009	5.4%	0.4%	2.2%	95.9%	12.9%	0.0%	0.0%	1.9%	45.0%	17.9%	12.0%

図 9. ベースラインとパックドレイアウトのインテル ® VTune ™ プロファイラーでの比較

パックド・メモリー・レイアウトを採用することにより、HotQCD のパフォーマンスはベースラインから 1.38 倍向上します（**図 10**）。`dslash` カーネルでは 1.54 倍向上しますが、パックドレイアウトがキャッシュ汚染を引き起こし、`dslash` に続く後続のベクトル演算が遅くなるため、ベンチマークの数値をそのまま受け取ることはできません。この最適化を HBM を搭載していないインテル® Xeon® システムに適用すると、パフォーマンスはベースラインから 1.21 倍向上しました（**図 11**）。

コンポーネント	ベースライン	ベースライン + プリフェッチ	パックド形式 + プリフェッチ
dslash	1	1.14x	1.54x
HotQCD	1	1.13x	1.38x

図 10. インテル® Xeon® CPU マックス・シリーズにおける HotQCD の相対的なパフォーマンス向上

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

	インテル® Xeon® プロセッサー (HBM なし)	インテル® Xeon® マックス・シリーズ (HBM 搭載) による スピードアップ
ベースライン	1.00	1.66
ベースライン + プリフェッチ	1.14	1.64
パックド形式 + プリフェッチ	1.21	1.88

図 11. インテル® Xeon® プロセッサーとインテル® Xeon® CPU マックス・シリーズにおける HotQCD のパフォーマンス

まとめ

最後に、要点をまとめます。

- HBM を搭載したインテル® Xeon® CPU マックス・シリーズは、同等のインテル® Xeon® プロセッサー（DDR5）と比べてパフォーマンスが大幅に（ベースラインで 1.66 倍）向上します。
- プリフェッチにより、HBM のパフォーマンスはさらに向上します（HBM ベースラインから 1.13 倍、最適化 + HBM なしから 1.88 倍）。
- プリフェッチのパフォーマンスを適切にチューニングするには、アプリケーションのメモリー・アクセス・パターンを理解することが不可欠です。プリフェッチ命令は低コストですが、使用には注意してください。
- 4KB ページ境界をまたぐ大きなメモリー・アクセス・ストライドは、HBM のパフォーマンスにとって理想的ではありません。ストライドアクセスよりもメモリーの連続したチャンクを読み取ることを優先することで、ベースラインの 1.38 倍から 1.54 倍の最適な HBM パフォーマンスを達成できます。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、Intel Corporation またはその子会社の商標です。
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。

intel software

THE PARALLEL UNIVERSE

インテルのテクノロジーを使用するには、対応したハードウェア、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。詳細については、OEM または販売店にお問い合わせいただくか、http://www.intel.co.jp/ を参照してください。

実際の費用と結果は異なる場合があります。

インテルは、サードパーティーのデータについて管理や監査を行っていません。ほかの情報も参考にして、正確かどうかを評価してください。

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。注意事項の改訂 #20110804。
https://software.intel.com/en-us/articles/optimization-notice#ja-jp

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。構成の詳細は、補足資料を参照してください。性能やベンチマーク結果について、さらに詳しい情報をお知りになりたい場合は、http://www.intel.com/benchmarks/（英語）を参照してください。

性能の測定結果はシステム構成の日付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。詳細は、システム構成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またはコンポーネントはありません。

本資料は、(明示されているか否かにかかわらず､また禁反言によるとよらずにかかわらず) 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。

インテルは、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、および非侵害性の黙示の保証、ならびに履行の過程、取引の過程、または取引での使用から生じるあらゆる保証を含みますが、これらに限定されるわけではありません。

© Intel Corporation. Intel、インテル、Intel ロゴ、その他のインテルの名称やロゴは、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。