intel software

THE PARALLEL UNIVERSE

エンタープライズ規模での サプライチェーン最適化

インテル® プロセッサーにおける Transformer モデル推論の最適化

電柱の保守予測を最適化してサービス品質を向上

Issue 52 2023

目次

編集者からのメッセージ 3

注目記事

エンタープライズ規模でのサプライチェーン最適化 6
Red Hat* OpenShift* Data Science とインテル® アーキテクチャーを使用してオープンソースの AI テクノロジーを活用

インテル® プロセッサーにおける Transformer モデル推論の最適化 20
oneAPI ディープ・ニューラル・ネットワーク・ライブラリー（oneDNN）の利点を生かした融合操作

電柱の保守予測を最適化してサービス品質を向上 28
インテル® AI アナリティクス・ツールキットでマシンラーニングを高速化

インテル® End-to-End AI Optimization Kit による人工知能の高速化 35
インテルにより最適化されたソフトウェアで AI を大衆化

Fortran と OpenMP* でヘテロジニアス・プログラミングの課題を解決 43
オープンな標準プログラミング言語でヘテロジニアス並列処理を表現

oneMKL と OpenMP* ターゲットオフロードで線形システムを解く 51
標準ベースのオープンなアプローチで線形代数計算を高速化

SYCL* によるデバイス検出 59
システムに搭載されているアクセラレーターを知る

レベルゼロとは 66
オープンなバックエンド・アプローチでどこでも計算を実現

© 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。

クロスベンダーのヘテロジニアス・コンピューティング

私の同僚は最近、oneAPI（英語）のさまざまな側面を試しており、そのすべてが有望性を示しています。Tony Mongkolsmai は、「NVIDIA*、AMD*、およびインテル® アクセラレーター向けにプログラムするため、CUDA*、ROCm*、および SYCL* を学ぶ必要があるのか ?」（英語）と疑問を投げかけ、3 つのベンダーすべての GPU で同じ SYCL* コードをコンパイルして実行する方法を示し、さらにインテル® GPU と NVIDIA* GPU で同時に実行する方法も示しました。本誌の名誉編集者である James Reinders は、CUDA* から C++ with SYCL* へ移行する（英語）ためのリソースと、ベンダー固有のツールやアクセラレーターの制約からコードを解放するのに役立つコード例とセルフガイドのチュートリアルを紹介しています。そして、Guy Tamir の oneAPI 基本トレーニング・シリーズ（英語）のビデオでは、SYCL* を使用したダイレクト・プログラミングから、oneAPI ディープ・ニューラル・ネットワーク・ライブラリー（oneDNN）などのコンポーネント・ライブラリーを使用した API ベースのプログラミングまで、oneAPI のすべての側面をカバーしています。Guy は、oneAPI の観点からデータ・サイエンス・アプリケーションについても説明しています。

データサイエンスといえば、本号では注目記事「**エンタープライズ規模でのサプライチェーン最適化**」を皮切りに、AI 関連のコンテンツを満載しています。この記事は、Red Hat の Ted Jones 氏と Karl Eklund 氏、インテルの Karol Brejna と Piotr Grabuszynski の共著です。Red Hat* OpenShift* Data Science プラットフォームとインテルにより最適化されたソフトウェアを使用して、オープンソースの AI テクノロジーを活用する方法を説明します。続く「**インテル® プロセッサーにおける Transformer モデル推論の最適化**」では、自然言語処理を行う Google の一般的な Transformer モデルに対するいくつかの最適化について説明します。次に、「**電柱の保守予測を最適化してサービス品質を向上**」では、Accenture と共同で開発した多くの実用的な AI リファレンス・キットの 1 つについて説明します。「**インテル® End-to-End AI Optimization Kit による人工知能の高速化**」では、インテルにより最適化されたソフトウェアで AI を大衆化する方法を示します。

データサイエンスから、Fortran と OpenMP* ターゲットオフロード API を使用したヘテロジニアス並列プログラミングに移って、「**Fortran と OpenMP* でヘテロジニアス・プログラミングの課題を解決**」では、並列ループをアクセラレーターにオフロードし、ホストとデバイスのメモリー間でデータを移動するいくつかのコンパイラー・ディレクティブについて説明し、Fortran 標準の DO CONCURRENT ループの概要を提供します。「**oneMKL と OpenMP* ターゲットオフロードで線形システムを解く**」では、oneMKL 関数をアクセラレーターにディスパッチする方法を紹介します。後者は、The Parallel Universe 51 号の記事「Fortran、oneMKL、OpenMP* を使用して LU 因数分解を高速化」の続編で、今回はホスト - デバイス間のデータ転送の概念分析から直接測定とデータ移動を最小限に抑えることによるパフォーマンス上の利点の分析を行います。

© 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。

最後に、2 つの記事「**SYCL* によるデバイス検出**」と「**レベルゼロとは**」で本号を締めくくります。前者では、John Pennycook と私が、SYCL* デバイス検出 API を使用して、システムで利用可能なアクセラレーターを特定し、それらの特性を照会する方法を示します。後者は、非常に強力な oneAPI を支えるレベルゼロのハードウェア抽象化レイヤーについて説明します。

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 4 月

© 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 →

エンタープライズ規模でのサプライチェーン最適化

Red Hat* OpenShift* Data Science とインテル® アーキテクチャーを使用してオープンソースの AI テクノロジーを活用

Ted Jones Red Hat シニア・ソリューション・アーキテクト
Karl Eklund Red Hat 主席アーキテクト
Karol Brejna インテル コーポレーション クラウド・ソリューション・アーキテクト
Piotr Grabuszynski インテル コーポレーション クラウド・ソリューション・エンジニア

サプライチェーンの崩壊、新型コロナウイルス感染症の影響、停滞する経済状況は、小売業者が直面している課題のほんの一部です。ビジネスプロセスの各段階で、起業家は会社の収益、競争力、または将来の方向性を決定する決断を下す必要があります。最適なソリューションの開発には、時間がかかり、要求が厳しく、誤りが発生しやすくなりがちです。

意思決定には、過去のデータによる裏付けが必要ですが、数千もの注文と配送から得られる膨大な量のデータを人間が分析することは困難なため、意思決定プロセスは、マシンラーニング（ML）を使用した自動メカニズムによってサポートするか、完全に置き換えるべきです。ML は、マシンがより多くのデータを分析して学習し、精度を向上するプロセスです。数百万ものエントリーでも問題なく処理できます。データ間で見つかった関係に基づいて数学的モデルが作成され、後で動作、結果、価格、配送時間などを予測するために使用されます。

© 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。

ML を使用して問題を解決するには、別の角度から見ることも重要です。企業のプロセスを人工知能（AI）アルゴリズムでサポートするには、まず適切なモデルを準備しなければなりません。しかし、これには適切なツールを使用する必要があります。そのため、インテルと Red Hat は、ML 関連のプロセスを促進する新しいソリューションに常に取り組んでいます。Red Hat* OpenShift* Data Science などの製品は、データ転送、データマイニング、トレーニング、プロセスの自動化、モデルのデプロイを改善し、インテルは、計算効率を向上させるため、scikit-learn* 向けインテル® エクステンション（英語）（インテル® AI アナリティクス・ツールキットの一部）などの最適化されたハードウェアとソフトウェアを提供しています。

問題ステートメント

増え続ける問題の 1 つは、配送日の見積もりです。物資の不足は、利益を減少させ、コストを増加させる可能性があります。そのため、プロセスを制御し、関連するリスクを管理することが重要です。以下の例では、過去のデータを使用して遅配を予測する方法を示しています。遅配の可能性が高い場合、例えば、倉庫から追加の在庫を出したり、代替部品を使用したり、あるいはサプライヤーを変更するという決定が下されます。

アプローチの説明

配送遅延予測は分類問題の一種です。この場合、目標は、過去の納入履歴や交通データなどの一連の入力に基づき、配送が遅れるか、時間どおりかを予測します。これらの特徴は、モデルのトレーニングに使用されます。トレーニング済みのモデルは、新しい未知のデータに対して予測を行うために使用できます。このアプローチでは、1 つのシンプルで高速なモデルを使用して、すべての注文を「スキャン」し、遅延の可能性を調べます。リスクのある注文は、さらに詳しく分析できます（例えば、遅延の潜在的な理由を示すほかのアルゴリズムを使用するなど）。

テクノロジー

Red Hat* OpenShift* Data Science

Red Hat* OpenShift* Data Science は、インテリジェント・アプリケーションのデータ・サイエンティストやプログラマー向けのサービスであり、セルフマネージドまたはマネージド・クラウド・プラットフォームとして利用できます。ML モデルを実際の環境にデプロイする前に、迅速に開発、トレーニング、テストできる、完全にサポートされた環境を提供します。オンプレミス、パブリッククラウド、データセンター、エッジのいずれであっても、コンテナ上の本番環境で ML モデルをデプロイし、Red Hat* OpenShift* Data Science からほかのプラットフォームに簡単にエクスポートできます。

ML に Red Hat* OpenShift* Data Science を使用すると、多くの利点があります。このプラットフォームには、Jupyter* Notebook、TensorFlow*、PyTorch*、scikit-learn* など、データ・サイエンティストがワークフローで使用するさまざまな商用パートナーとオープンソースのツールおよびフレームワークが含まれています。Red Hat* OpenShift* Data Science は、安全でスケーラブルな環境を提供します。

© 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。

インテル® AI アナリティクス・ツールキット

インテル® AI アナリティクス・ツールキットは、XGBoost、scikit-learn*、TensorFlow*、PyTorch* など、インテルにより最適化された AI ソフトウェア、ライブラリー、フレームワークのセットを提供します。インテル® アーキテクチャー上のエンドツーエンドのデータサイエンスとアナリティクスのパイプラインを高速化するのに役立ちます。このツールキットは、AI アプリケーションの作成をスピードアップして簡素化することを目的としています。また、複数のプラットフォームへのモデルのデプロイもサポートします。

シナジー

以下に示す例は、インテル® AI アナリティクス・ツールキットと、Red Hat* OpenShift* Data Science に組み込まれている Jupyter* Notebook カーネルを使用して作成しました。これは、データサイエンスとアナリティクスに安定した開発環境を提供するシンプルでエレガントな方法です。

リファレンス実装

ML ソリューションの作成プロセスには、通常いくつかのステップが含まれます。

- 問題の定義
- データの収集
- 環境の準備とツールの選択
- データの前処理と特徴量エンジニアリング
- トレーニング・データの準備
- 最適な ML アルゴリズムの選択
- トレーニング
- モデルの検証
- モデルの配布とデプロイ

問題の定義

まず、解決する問題の種類を決定します。問題が複雑になるほど、ソリューションが見つかる可能性は低くなります。これは一般的なルールではありませんが、問題を単純化することから始める必要があります。この例では、配送が開始される前に潜在的な配送の問題を示すシンプルなソリューションが必要です。

データの収集

必要な答えが分かったら、その答えが含まれている可能性のあるデータを探します。場合によっては、複数のデータベースやファイルを調べ、それらを 1 つの統一されたセットに結合する必要があります（この記事のすべての例は、Kaggle*（英語）プラットフォームで公開されているオープンソースのデータに基づいています）。以下の例では、次の特徴を持つ過去のレコードを含む CSV ファイルを保有していると仮定します。

© 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。

- 注文 ID
- 注文日
- 配送日
- 配送モード
- 注文者 (ID、名前)
- 顧客区分 (コンシューマー、法人 ...)
- 配送先 (市町村、都道府県、国)
- 市場コード (US、APAC...)
- 地域
- 製品 (ID、カテゴリー)
- 注文内容 (価格、数量、割引、配送料)
- 注文優先順位

図 1 はデータの一部を示しています。

Order Date	Ship Date	Ship Mode	Segment	City	Region
1430 unique values	1464 unique values	Standard Class 60% Second Class 20% Other (10206) 20%	Consumer 52% Corporate 30% Other (9343) 18%	New York City 2% Los Angeles 1% Other (49628) 97%	Central South Other (33528)
15-10-2012	15-10-2012	Same Day	Consumer	Amarillo	Central
03-11-2011	05-11-2011	Second Class	Corporate	Gold Coast	Oceania
12-12-2011	14-12-2011	Second Class	Consumer	Fresno	West
14-09-2011	15-09-2011	First Class	Corporate	Kamina	Africa
14-09-2011	14-09-2011	Same Day	Consumer	New York City	East
05-11-2014	05-11-2014	Same Day	Corporate	Burlington	South
14-01-2014	18-01-2014	Standard Class	Consumer	Stockton-on-Tees	North
10-01-2011	11-01-2011	First Class	Consumer	Brisbane	Oceania
22-08-2013	26-08-2013	Standard Class	Consumer	Mataram	Southeast Asia

図 1. データセットを視覚化

また、運送会社と交わしたサービス・レベル・アグリーメント（SLA）があります。配送時間は SLA の規定に沿っている必要があります。

- 配送モードが即日（Same Day）の場合は即日配送
- 第 1 種郵便（First Class）の場合は翌日配送
- 第 2 種郵便（Second Class）の場合は 3 日以内に配送
- 普通郵便（Standard Class）の場合は 5 日以内に配送

© 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。

環境の準備とツールの選択

Jupyter* Notebook などの開発プロセスをサポートするツールを使用して、プロジェクトの作業を円滑に進めることができます。Red Hat* OpenShift* Data Science UI は、このような環境を素早く立ち上げることができます（**図 2**）。インテルにより最適化されたソフトウェアがすでに統合されているため、環境の作成と管理が容易です。

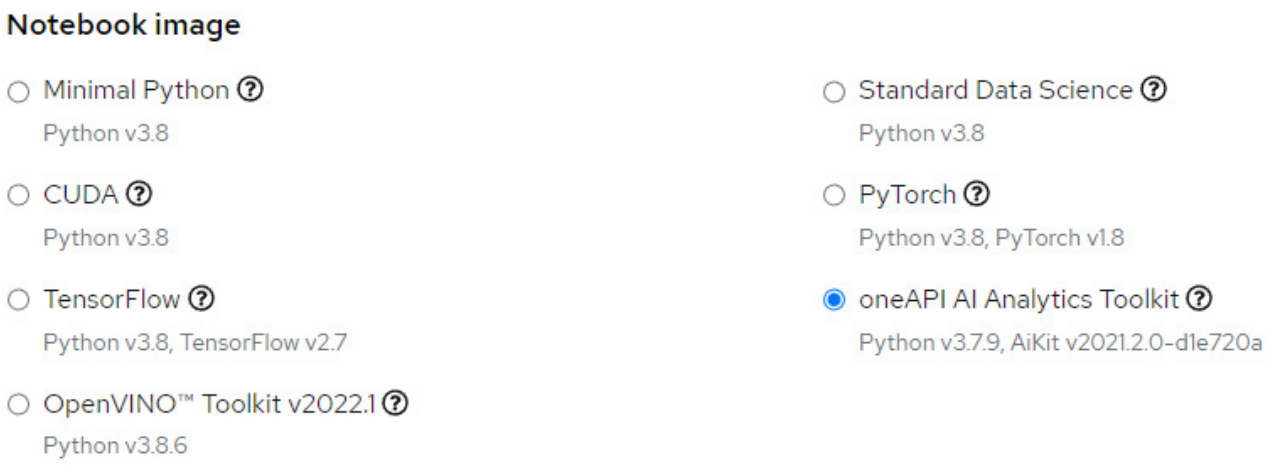

図 2. Red Hat* OpenShift* Data Science インターフェイスで Jupyter* Notebook を作成

ソリューションに適した AI フレームワークを選択します。オープンソースの ML ライブラリーとして人気の高い scikit-learn* は、教師あり / 教師なしトレーニングの幅広いアルゴリズムを提供しています。scikit-learn* 向けインテル® エクステンションは、インテル® ハードウェア上でパフォーマンスを向上します（「データ・サイエンティストが必要とする scikit-learn* のパフォーマンスをインテルが大幅に向上」(英語)を参照)。インテル® AI アナリティクス・ツールキットのコンテナ内で有効にするには、2 行のコードを追加するだけです。

```
from sklearnex import patch_sklearn
patch_sklearn()
```

この例では、scikit-learn* 向けインテル ® エクステンション・ライブラリーがインストールされた Red Hat* OpenShift* Data Science をベースにしています。提案するソリューションのアーキテクチャーを**図 3** に示します。

© 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。

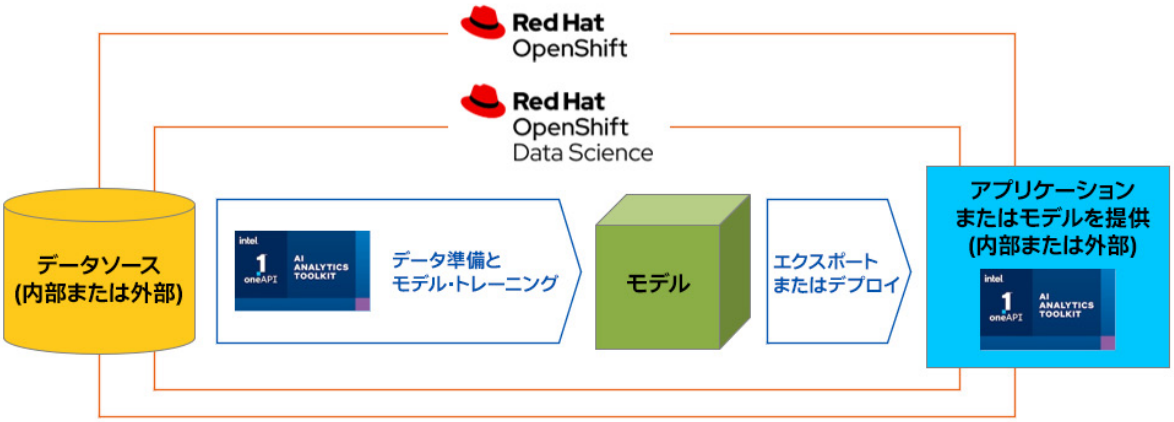

図 3. この例のソリューション・アーキテクチャー

データの前処理と特徴量エンジニアリング

すべてのデータを収集したら、それらをどのように処理するかを決定します。前処理では、データセットのクリーニングとフォーマット調整を行います。これには、欠損値や矛盾の除去、外れ値の処理、データセットのスケーリングなどが含まれます。次に、トレーニングに使用する特徴を決定します。特徴量エンジニアリングでは、データを選択して新しい特徴に変換し、データ内のパターンや関係をより適切に表現します。この例では、問題に対する明快な答えはありませんが、注文日、配送日、配送モード、SLA 条件が分かれば、遅延を計算することができます。

まず、データを読み込んで、pandas の DataFrame 形式で保存します。

```
df = pd.read_csv(DATA_PATH, encoding='latin-1', parse_dates=True)
```

日付をより便利な形式に変換し、配送時間を日数で計算する必要があります。

```
df['Order Date'] = pd.to_datetime(df['Order Date'], format="%d-%m-%Y")
df['Ship Date'] = pd.to_datetime(df['Ship Date'], format="%d-%m-%Y")

df['Delivery Time'] = (df['Ship Date'] - df['Order Date']).dt.days
```

このデータと SLA から遅延を計算できます。新しいカラムを作成し、ラベルを付けます:遅延(1)または時間どおり(0)。

© 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。

```
df['SLA'] = df['Ship Mode'].map({
    'Same Day': 0,
    'First Class': 1,
    'Second Class': 3,
    'Standard Class': 5,
})

df['Delay in Days'] = (df['Delivery Time'] - df['SLA']).map(lambda d: d if d > 0 else 0)

df['Delay'] = df['Delay in Days'].map(lambda d: 1 if d > 0 else 0)

df['Delay'].value_counts()
```

```
0 31741
1 19549
Name: Delay, dtype: int64
```

値「Delay」が答えです。この例では、31,741 件が時間どおりで、19,549 件が遅延です。「Delay」カラムは、トレーニング中にターゲット（予測する値）として使用します。

トレーニング・データの準備

これまでのプロセスは、必要な特徴を含むコレクションの構築に重点を置いていました。有効なモデルを作成するには、データのバランス、分類、シャッフルなど、さらに多くの操作が必要になるかもしれません。その後、データをトレーニング・セットとテストセットに分割します。この例では、遅延と時間どおりの比率は約 2：3 であり、これは許容範囲ですが、データを分類する必要があります。

```
# 分類するカラム
CATEGORIZE = [
    'Ship Mode',
    'Customer ID',
    'Segment',
    'City',
    'State',
    'Country',
    'Postal Code',
    'Market',
    'Region',
    'Product ID',
    'Category',
    'Sub-Category',
    'Order Priority',
]

pd_categories = {}

for c in CATEGORIZE:
    pd_categories[c] = pd.Categorical(df[c])
    df[c] = pd_categories[c]
    df[f'{c}-cat'] = df[c].cat.codes

df.head()
```

分類後、データをトレーニング・セットとテストセットにランダムに分割します。

© 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。

```
m = np.random.rand(len(pds)) < 0.8

ds_train = pds[m]
ds_test = pds[~m]
```

最適なアルゴリズムの選択

問題に応じて、教師あり / 教師なしアルゴリズムを使用できます。この決定を行う際には、データの種類、目標、パフォーマンス、および精度を考慮することが重要です。アルゴリズムの評価と比較には、予測精度や曲線下面積（AUC）などの既知のメトリックを使用できます。AUC は、真陽性と偽陽性の予測プロットの下の面積として計算されます。したがって、常に 0 から 1 の間の数値になります（大きいほど良い）。

アプリケーションに最適なアルゴリズムを確認します。ガウス・ナイーブ・ベイズ、ランダムフォレスト、LightGBM、決定木、XGBoost、K 近傍法の 6 つの異なる分類器からモデルをトレーニングして評価します。アルゴリズムごとに、平均 AUC と 20 反復の実行時間を計算します（**図 4**）。結果から、このデータに最適なモデルはランダムフォレストであることが分かります。ここでは、ランダムフォレストを使用します。

トレーニング

```
from statistics import mean

NUMBER_OF_ITERATIONS = 20

time_results = {}
roc_auc_results = {}

for _ in range(NUMBER_OF_ITERATIONS):
    models = [
        ("NB",      GaussianNB()),
        ("RF",      RandomForestClassifier()),
        ("lgbm",    lgb.LGBMClassifier()),
        ("Dtree",   DecisionTreeClassifier()),
        ("XGB",     xgb.XGBClassifier()),
        ("KN",      KNeighborsClassifier()),
    ]

    for name, model in models:
        start_time = time()
        kf = KFold(shuffle=True, random_state=7919)
        cv_result = cross_val_score(model, ds_train_features, ds_train_target, cv=kf, scoring="roc_auc")
        stop_time = time()

        mean_cv_results = mean(cv_result)
        print(name, cv_result, mean_cv_results)

        total_t = stop_time - start_time
        try:
            time_results[name].append(total_t)
        except KeyError:
            time_results[name] = [total_t]

        try:
            roc_auc_results[name].append(mean_cv_results)
        except KeyError:
            roc_auc_results[name] = [mean_cv_results]
```

© 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。

```
------------------------- RESULTS FOR OPTIMIZED SKLEARN -------------------------
| name   | iters | avg time      | min time      | max time      | avg auc       |
| NB     | 20    | 0.0972        | 0.0884        | 0.1522        | 0.6808        |
| RF     | 20    | 2.4618        | 2.3030        | 2.8867        | 0.9237        |
| lgbm   | 20    | 1.4740        | 1.0682        | 3.2824        | 0.8647        |
| Dtree  | 20    | 1.0654        | 1.0400        | 1.1016        | 0.8174        |
| XGB    | 20    | 9.0131        | 5.8058        | 26.3675       | 0.8811        |
| KNN    | 20    | 0.2834        | 0.2675        | 0.3175        | 0.5271        |
```

図 4. 簡単な評価の結果

ML トレーニングは、入力特徴をターゲットにマップするデータ内のパターンを見つけます。トレーニング済みのモデルにはこれらの依存関係が含まれるため、将来の値を予測できます。ソリューションの開発中に、選択したアルゴリズムとライブラリーに応じて、トレーニング・プロセスの設定を変更できます。これらの設定を調整すると、モデルが改善または悪化する可能性があるため、さまざまな値を試してみる価値があります。この例では、`class_weight` パラメーターを設定して、データのわずかな不均衡を補正します。それ以外は、デフォルトのパラメーターを使用したトレーニングで最良の結果が得られました。

```
rfc = RandomForestClassifier(
    class_weight={0:2, 1:3}
)

rfc.fit(ds_train_features, ds_train_target)
```

最終モデルの検証

MLモデルの検証は、未知のデータに対してその精度を評価するプロセスです。モデルを実世界にデプロイする前に、データ・サイエンティストが問題を発見して修正できるため、このステップは非常に重要です。精度が低すぎる、または非常に高い場合は、データが少なすぎる、重要な特徴がない、データ漏洩、学習不足または過学習などの潜在的な問題を示している可能性があります。基本的な評価には、前述の精度と AUC メトリックを使用できます。検証プロセスを開始し、データを収集するコードを以下に示します。

```
ds_test_features = ds_test[FEATURES]
ds_test_target = ds_test[TARGET]

preds = rfc.predict(ds_test_features)
preds_prob = rfc.predict_proba(ds_test_features)[:, 1]

acc_score = accuracy_score(ds_test_target, preds)
auc_score = roc_auc_score(ds_test_target, preds_prob)

print(f'acc: {acc_score} | auc: {auc_score}')
```

© 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。

```
acc: 0.8664118396550035 | auc: 0.9381309986948686
```

約 86% の精度を達成し、AUC は 0.93 を上回りました。データ準備段階でのランダムな要因（シャッフルなど）により、結果がわずかに異なる場合があります。

ソリューションのベースラインを計算して、健全性チェックを行います。毎回「0」と答えるモデルをシミュレートし、その精度を計算します。

```
accuracy_score(ds_test_target, np.zeros_like(ds_test_target)), roc_auc_score(ds_test_target, np.zeros_like(ds_test_target))
```

```
(0.6153169873986519, 0.5)
```

ベースラインの精度は約 62% なので、ベースラインと比較して大きな改善が見られます。

結果を視覚化するため、グラフを使用できます。その 1 つが、受信者動作特性 (ROC) 曲線です。**図 5** は、今回の実験における ROC 曲線を示したものです。この曲線の下の面積が AUC 値です。

```
%matplotlib inline

fpr, tpr, thresholds = roc_curve(ds_test_target, preds_prob)

plt.plot(fpr, tpr)
plt.show()
```

© 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。

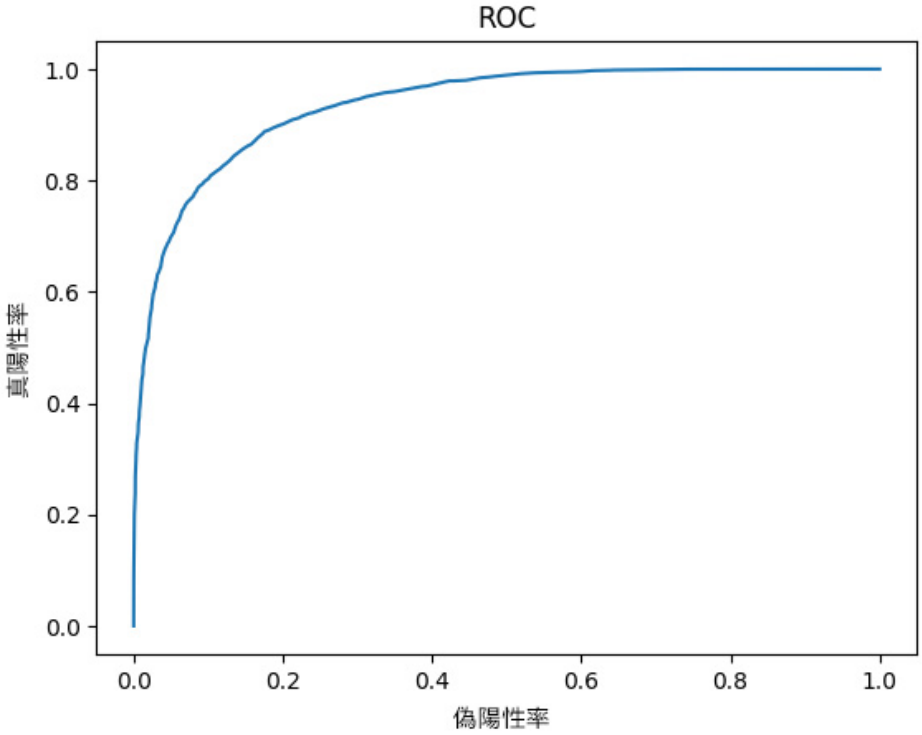

図 5. この実験における ROC 曲線

また、どの特徴が出力に最も大きな変化をもたらすか視覚化することができます（**図 6**）。

```
importances = rfc.feature_importances_
std = np.std([tree.feature_importances_ for tree in rfc.estimators_], axis=0)

forest_importances = pd.Series(importances, index=FEATURES)

fig, ax = plt.subplots()
forest_importances.plot.bar(yerr=std, ax=ax)
fig.tight_layout()
```

© 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。

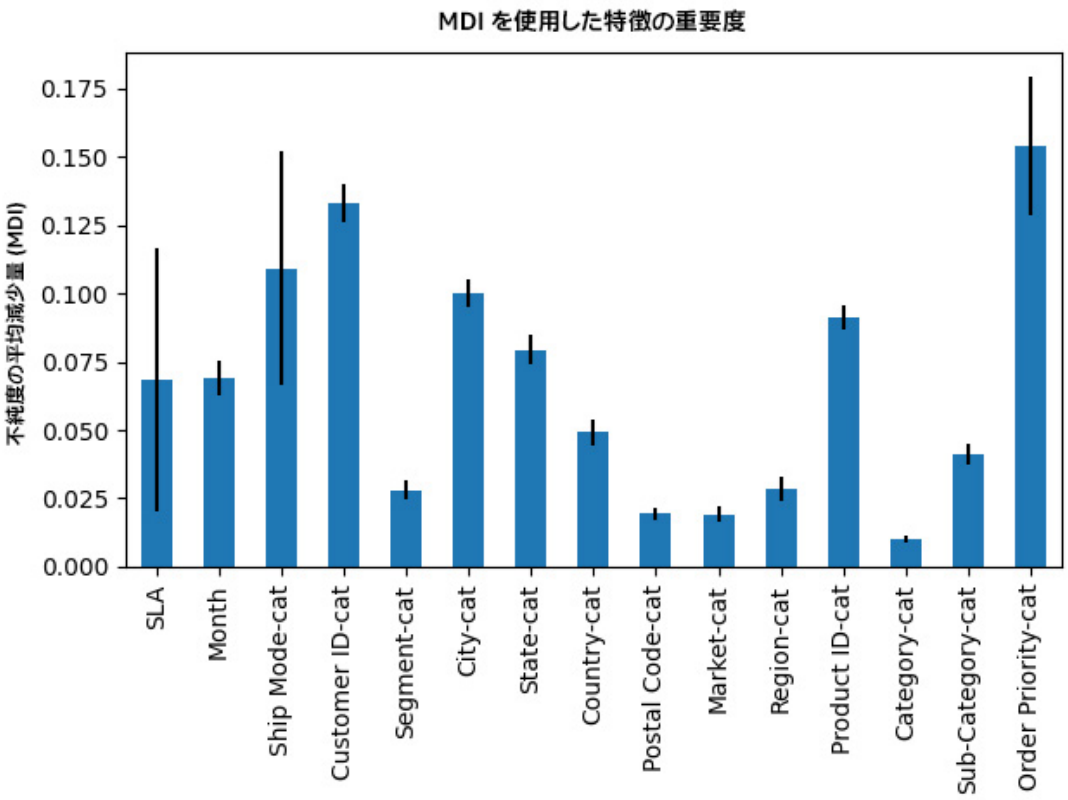

図 6. この例の特徴の重要度を示す図

推論の実行：簡単な方法

健全性チェックのため、同じ Jupyter* Notebook の中でモデルを使用してみます。検証データから 1 行を選択し、サンプルを用意します。

```
INPUT_DATA = {
  'Customer ID':     'AA-00000',
  'Order Date':      '12-07-2022',
  # …
  'Quantity':        '12',
  'Order Priority':  'Medium',
}
```

このモデルを使用するには、トレーニング時に行ったように、入力データを変換する必要があります。トレーニング時に使用したのと同じカテゴリーを使用することが重要です。例えば、「New York」が 129 にマッピングされていた場合、ここでも同じ数字を使用する必要があります。

© 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。

```
INPUT_DATA['Postal Code'] = float(INPUT_DATA['Postal Code'])

input_d = pd.DataFrame(data={k: [v] for k, v in INPUT_DATA.items()})

udf = pd.DataFrame()

# SLA
udf['SLA'] = input_d['Ship Mode'].map({
    'Same Day': 0,
    'First Class': 1,
    'Second Class': 3,
    'Standard Class': 5,
})

# 月
udf['Month'] = pd.to_datetime(input_d['Order Date'], format="%d-%m-%Y").dt.month

# 配送モード - cat
for c in CATEGORIZE:
    try:
        udf[f'{c}-cat'] = pd_categories[c].categories.get_loc(INPUT_DATA[c])
    except KeyError:
        udf[f'{c}-cat'] = -1

udf.head()
```

	SLA	Month	Ship Mode-cat	Customer ID-cat	Segment-cat	City-cat	State-cat	Country-cat	Postal Code-cat	Market-cat	Region-cat	Product ID-cat	Category-cat	Sub-Category-cat	Order Priority-cat
0	0	7	1	-1	0	375	145	47	-1	4	3	8246	2	0	3

推論を実行します。

```
if_delay = not rfc.predict(udf)[0]
proba_of_delay = rfc.predict_proba(udf)[:, 1][0]
proba_of_delay = (1 - proba_of_delay if if_delay else proba_of_delay) * 100

print('We expect' + (' no' if if_delay else '') + f' delay (with probability ~{proba_of_delay:.0f}%)')
```

```
We expect no delay (with probability ~93%)
```

© 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

今日の起業家は、サプライチェーンの崩壊、新型コロナウイルス感染症の影響、停滞する経済状況などの困難に直面しています。小売業者は、収益、競争力、または会社の将来の方向性に影響を与える選択を、ビジネスプロセスのあらゆるレベルで決断する必要があります。ML は、意思決定プロセスを大幅に簡素化し、その精度を向上できます。この記事では、例を使って小売配送の遅延を検出する方法を示しました。

企業は、過去のデータに基づいた手法とアルゴリズムを使用して、潜在的な配送遅延を警告するソリューションを作成できます。このアプローチにより、すべての配送を迅速かつ高精度にスキャンできます。問題のある注文は、さらに詳しく分析することが可能です。例えば、遅延時間、遅延に最も影響を与える要因、部品やサービスを時間どおりに提供できるサプライヤーなどの情報を得られます。詳しい分析では、サプライヤーが提供する製品、カテゴリー、またはデータに応じて異なるアルゴリズムを使用し、複数のモデルを生成して予測精度を高めることができます。

Red Hat* OpenShift* Data Science と scikit-learn* 向けインテル® エクステンションは、データサイエンスと分析プロセスを改善する強力な組み合わせです。インテルにより最適化されたライブラリーを含む Red Hat* OpenShift* Data Science に組み込まれた Jupyter* Notebook のおかげで、環境を迅速かつ容易に準備できます。

© 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。

インテル® プロセッサーにおける Transformer モデル推論の最適化

oneAPI ディープ・ニューラル・ネットワーク・ライブラリー（oneDNN）の利点を生かした融合操作

Xiaoming（Jason）Cui インテル コーポレーション AI フレームワーク・エンジニア
Ashraf Bhuiyan インテル コーポレーション AI ソフトウェア・エンジニアリング・マネージャー

Transformer モデルは、自然言語処理（NLP）で最も人気のあるモデルの 1 つです（**図 1**）。2017 年に Google によって公開（英語）されて以来、ほかの多くの NLP モデルで採用されており、非 NLP モデルにも拡張されています。言語翻訳は Transformer の一般的な用途であり、その優れた精度と並列処理により、従来の LSTM モデルに大きく取って代わるものです。BERT（英語）モデルは Transformer に基づいているため、Transformer を最適化すると、多くの NLP モデルおよび非 NLP モデルを改善できます。

© 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。

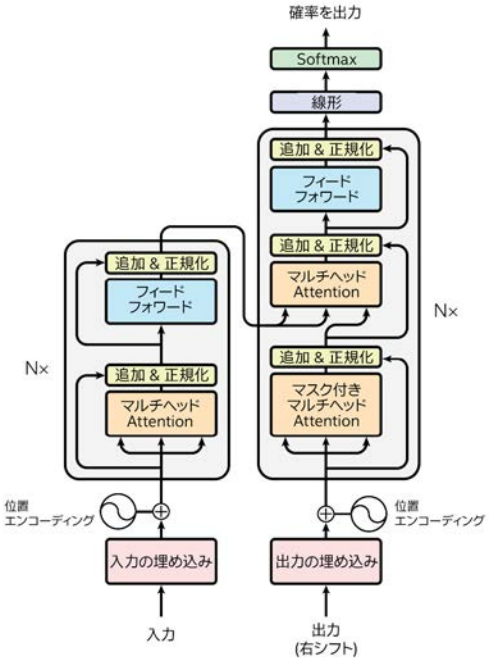

図 1. Transformer モデル・アーキテクチャー（出典：Vaswani ほか、Attention Is All You Need（英語））

インテルは、MLCommons v0.5（英語）の Transformer のトレーニングおよび評価コードを始め、インテル® Xeon® スケーラブル・プロセッサーでの推論向けにモデルを最適化しています。これらの最適化により、推論のスループットとレイテンシーのどちらもパフォーマンスが大幅に向上しました。最適化されたモデルは、インテル® アーキテクチャー向け Model Zoo（英語）に追加されています。

モデルのトレーニング

推論の最適化を試みる前に、トレーニング済みモデルを入手する必要があります。MLPerf* Transformer モデル（v0.5）を使用し、MLPerf* が推奨する精度でトレーニングしました。このモデルにはチェックポイントがあり、必要に応じて、さらにトレーニングすることで精度を少し向上できます。トレーニング済みのチェックポイントから推論を行うこともできますが、チェックポイントには推論に必要のないトレーニング・ノードが含まれているためサイズが大きく、通常、モデルをデプロイする方法としては最適ではありません。また、チェックポイントから推論を実行すると、重みをロードするオーバーヘッドが発生します。

推論パフォーマンスを最適化するため、トレーニング済みチェックポイントを使用する代わりに、トレーニング済みモデルのチェックポイントを、推論グラフとモデルの重みだけを含むグラフに固定しました。その結果、MLPerf* Transformer モデルのサイズは、チェックポイントを使用した場合の 1GB 以上から約 800MB に減りました。固定したグラフは、モデルを int8 に量子化してさらにパフォーマンスを向上する場合にも必要になります。グラフを固定するスクリプトは、インテル® アーキテクチャー向け Model Zoo にあります。

© 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。

グラフを固定する過程で、定数の畳み込み、単位元ノードの削除など、いくつかの最適化が適用されます。グラフを固定したら、既知の入力ノードと出力ノードを持つ protobuf（`.pb`）ファイルを取得します。固定したグラフから推論を実行するのは容易です。入力データをモデルの入力ノードに供給するだけです。実行後、モデルの出力ノードから結果を得ることができます。後でモデルをさらに最適化するアイデアを思いついた場合、モデルの重みが変わらない限り、再度トレーニングすることなく、最適化されたコードでモデルを再固定できます。

Transformer モデルには、エンコーダーとデコーダーの 2 つの主要なモジュールがあります。各モジュールには、フィード･フォワード･ネットワーク（FFN）、エンコーダー･デコーダーの Attention、ビーム検索、融合操作、埋め込み、Self-Attention など、複数のレイヤーがあります。ここでは、これらのレイヤーのいくつかに適用した最適化を紹介します。

FFN レイヤーのパディング / アンパディングの最適化

ここで最も重要な最適化は、FFN レイヤーからパディング / アンパディング・アルゴリズムを削除することです。このレイヤーへの入力は、形状 `[batch_size, input_length, Hidden_size]` の 3 次元パディングテンソルです。**図 2** に示すように、このモデルには 2 番目の次元にパディングがあり、空白ブロックはパディング値ゼロを表します。モデルは、テンソルを密レイヤーに送る前にパディングを検出して削除し、**図 3** に示すように、形状 `[batch_size, number_tokens*Hidden_size]` の 2 次元テンソルに再整形します。パディングの値はゼロのままなので、計算する必要はありません。しかし、密レイヤーの後、出力テンソルにパディングを追加して、元の形状に戻す必要があります（次のレイヤーの入力テンソルになるため）。

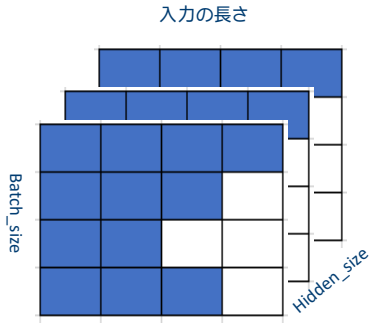

図 2. FFN 入力テンソルのデータレイアウト

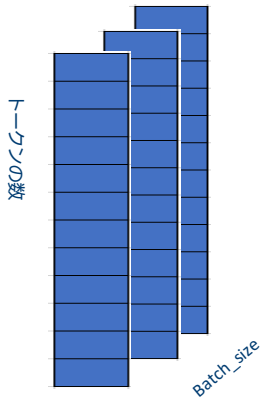

図 3. FFN 入力テンソルの最適化されたデータレイアウト

© 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。

パディング / アンパディング・アルゴリズムは、より多くのメモリー操作を使用することで、計算量を減らします。計算がメモリー操作より遅い場合、パフォーマンスは向上します。しかし、最近のインテル® Xeon® スケーラブル・プロセッサーでは、このアルゴリズムの計算はメモリー操作よりも高速です。したがって、FFN レイヤーからパディング / アンパディングを削除することで、パフォーマンスが向上します。

エンコーダー・デコーダーの Attention キャッシュの最適化

Transformer モデルでは、入力テンソルはエンコーダー・モジュールを通過して、エンコーダー・メモリー・テンソルを出力します。これは、デコーダーモジュールでエンコーダー・デコーダーの Attention の計算に使用されます（**図 4**）。

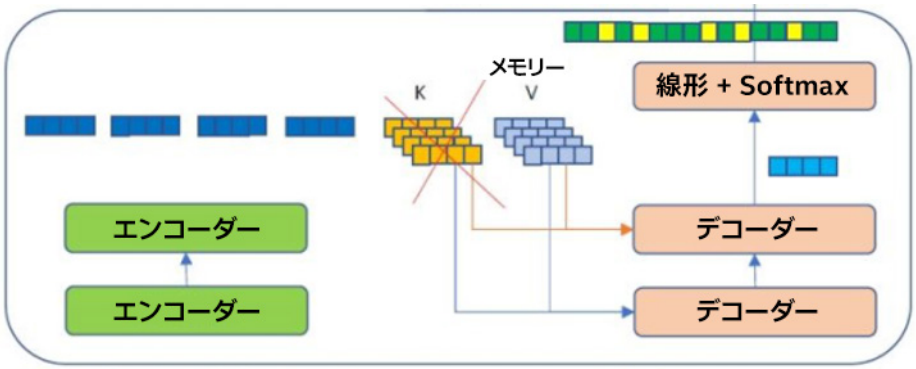

図 4. エンコーダー・デコーダーの Attention

Attention は、Transformer モデルの最も重要な部分です。次の式で計算されます。ここで、Q、K、および V は密レイヤーの後のテンソルです。

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Q は、デコーダーの入力を使用した密レイヤーの結果です。K と V は、エンコーダー・メモリー・テンソルの入力を使用した密レイヤーの結果です。デコーダーの反復のすべてのステップで、デコーダーの入力テンソルの値は変化しますが、エンコーダーからのメモリーテンソルの値は同じです。つまり、K テンソルと V テンソルはデコーダーのすべてのステップで同じですが、元の Transformer モデルはこれらのテンソルを再計算します。

この冗長な計算を排除する必要があることは明らかです。そのため、K と V をキャッシュするメカニズムを追加しました。エンコーダー・デコーダーの Attention の開始時にのみ K と V テンソルを計算し、残りのデコーダーの反復で再利用するため値をキャッシュします。

© 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。

ビーム検索キャッシュの最適化

ビーム検索も時間のかかる Transformer モジュールです。プロファイル結果は、このモジュールの最大のボトルネックが、特定の条件を満たすテンソル値を収集する `Gather_ND` 操作であることを示しています（**図 5**）。メモリーアクセスがランダムであるため、この操作のベクトル化と並列化は困難です。

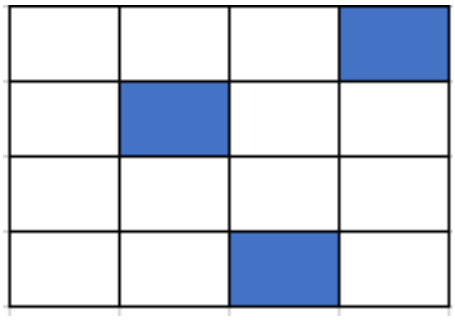

図 5. Gather_ND パターン

しかし、モデルで収集する必要がある値は、多次元テンソルのベクトルであることに気付きました。TensorFlow* フレームワークには、テンソルからベクトルを収集する `Gather_V2` 操作が含まれています（**図 6**）。この操作は容易にベクトル化および並列化でき、結果は同じになります。ビーム検索アルゴリズムで `Gather_ND` を `Gather_V2` に置き換えると、パフォーマンスが向上します。

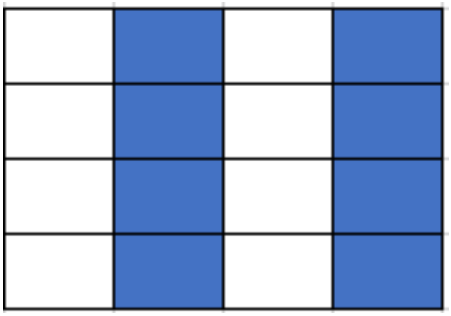

図 6. Gather_V2 パターン

融合操作の最適化

融合操作は、ディープラーニング・モデルにおける一般的な最適化です。Transformer では、`MatMul`、`Reshape`、`BiasAdd`、および要素ごとの操作が FFN レイヤーのホットスポットであることに気付きました。インテルによる TensorFlow*（英語）の最適化は、これらの操作の特定のパターンを新しい操作に融合します。しかし、Transformer モデルの操作パターン（**図 7**）は、必要な融合パターン（**図 8**）と一致しません。モデルのソースコードを調査した結果、エラーを発生させたり、計算やメモリー使用量を増加させることなく、`Reshape` 操作を `MatMul` の前に移動できることが分かりました。これは、ソースコードを数行変更するだけで済みます。

© 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。

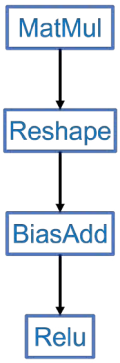

図 7. 最適化前の操作

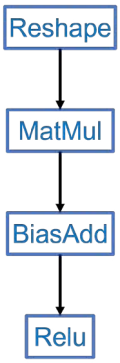

図 8. 変更後のパターン

再生成されたモデルで推論を実行すると、TensorFlow* 向けインテル® オプティマイゼーション・ランタイムは、高度に最適化された oneAPI ディープ・ニューラル・ネットワーク・ライブラリー (oneDNN) (英語) を呼び出す **MklFusedMatMul** という名前の新しい操作に操作を自動的に融合します (**図 9**)。これにより、メモリー操作が減り、元のモデルよりもはるかに優れたパフォーマンスを実現します。

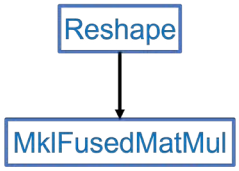

図 9. 融合および最適化された最終パターン

Transformer の Attention レイヤーにおける推論パフォーマンスのホットスポットは、`Mul`、`BatchMatMul`、`AddV2`、および `Softmax` であることが分かりました (**図 10**)。`Mul`、`AddV2`、および `Softmax` は、要素単位のメモリー依存の操作です。

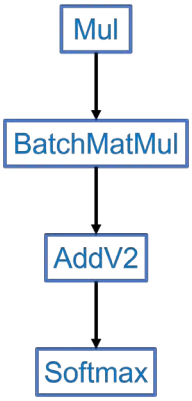

図 10. オリジナルの操作シーケンス

© 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。

TensorFlow* 向けのインテルによる最適化は、シーケンシャルな `BatchMatMul`、`Mul`、および `AddV2` 操作を融合します。ここでも、パターンを再配置することで（**図 11**）、最適化が可能になります（**図 12**）。これらの融合は、効率良く計算するだけでなく、メモリー・トラフィックを軽減するため、モデルにとって重要です。メモリー・トラフィックの軽減は、メモリー依存の操作で特に重要です。

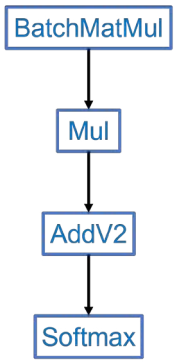

図 11. 変更後の操作シーケンス

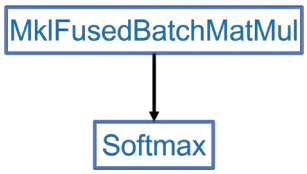

図 12. 融合および最適化された最終パターン

次の最適化は、Transformer モデルでレイヤーの正規化を融合します。レイヤーの正規化は、エンコーダーおよびデコーダーモジュールのすべてのレイヤーで使用され、メモリー移動または要素単位の操作のいずれかを行います。これらの操作は、メモリー依存のパフォーマンス・ボトルネックであるため、メモリー・トラフィックを軽減するレイヤー正規化の融合を実装しました（**図 13** と**図 14**）。

図 13. 最適化前の操作

図 14: 最適化後のフレームワーク

© 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

Transformer は強力ですが、複雑なディープラーニング・モデルです。MLPerf* リポジトリーの Transformer 推論モデルは、oneDNN ライブラリーを利用するインテルの最適化バージョンほどハイパフォーマンスではありません。ここで紹介した最適化は、ベースラインの推論パフォーマンスを取得したデフォルトの FP32 モデルに基づいています。これらの最適化により、モデルの精度を損なうことなく、パフォーマンスが大幅に向上しました。サポートしているハードウェアでは、bfloat16 および int8 に変換することで、モデルをさらに最適化できます。例えば、第 4 世代インテル® Xeon® スケーラブル・プロセッサーは、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）命令で、bfloat16（BF16）（英語）と 8 ビット量子化（int8）の両方をサポートしています。

- **TensorFlow* と Bfloat16 により第 3 世代インテル® Xeon® スケーラブル・プロセッサー上で AI パフォーマンスを高速化（英語）**
- **第 4 世代インテル® Xeon® スケーラブル・プロセッサー上で TensorFlow* を最適化（英語）**

これらの低精度のサポートを使用するように変換されたモデルでは、さらに優れたパフォーマンスを実現できます。詳細は、インテル® アーキテクチャー向け Model Zoo（英語）を参照してください。この記事で使用したすべてのトレーニング済みチェックポイント、グラフの固定、およびソースコードは、皆さんが参照したり、使用できるようにリポジトリーで公開しています。

© 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。

電柱の保守予測を最適化してサービス品質を向上

インテル® AI アナリティクス・ツールキットでマシンラーニングを高速化

Kelli Belcher インテル コーポレーション AI ソリューション・エンジニア

世界的に電力需要が増加する中、信頼性の高いサービスを提供し、コストのかかる予期しないダウンタイムを回避するには、インフラ設備を正確かつ効率的に監視する必要があります。現在の手動による問題の特定は 50% 未満の精度であり、電柱の保守と交換のコストは 100 億米ドルを超えています（出典 : Utility Poles : Maintenance or Replacement（英語）、Utility Partners of America、2020 年 8 月 8 日）。

インテルは Accenture と協力して、Predictive Asset Analytics Reference Kit（英語）を開発しました。これは、インフラ設備を予防的に保守し、システムの安定性を向上し、機能停止やダウンタイムを回避し、運用コストを抑えるため、インフラ設備の健全性と障害の可能性を予測するように設計されています。このリファレンス・キットには、トレーニング・データ、オープンソースのトレーニング済みの資産予測分析モデル、ライブラリー、ユーザーガイド、およびトレーニング・サイクル、予測スループット、精度を最適化する oneAPI コンポーネントが含まれています。

© 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。

このリファレンス・キットの実装方法を示すため、このチュートリアルで使用するデータセットは、ユーザーガイド（英語）の手順に従って生成されました。ワークフローを**図 1** に示します。これは、電柱の使用年数、保守履歴、機能停止記録、地理空間データなど、インフラ設備の全体的な健全性に関する 34 の特徴で構成されています。ターゲット変数は、電柱が故障しているかどうかを示すバイナリー・インジケーターです。

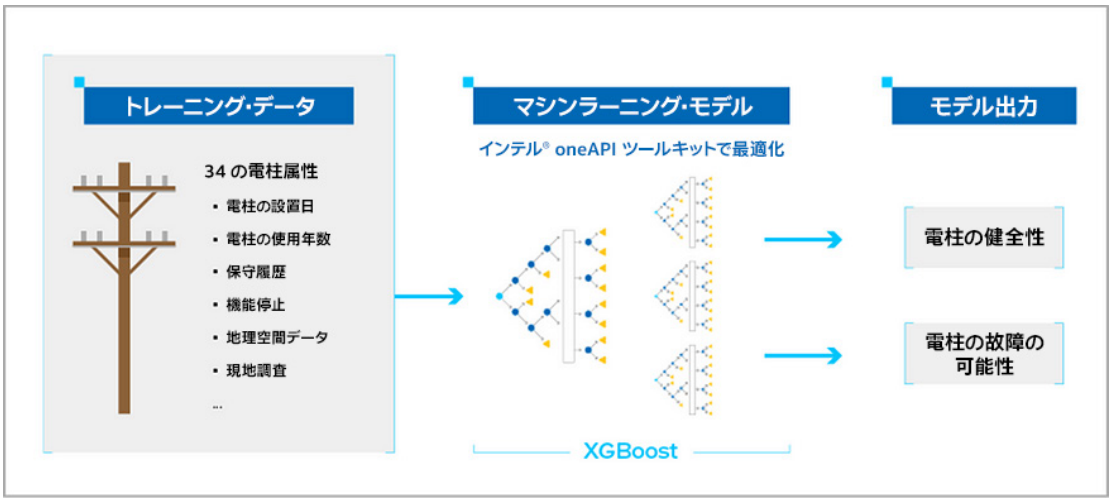

図 1. Predictive Asset Analytics Reference Kit のワークフロー

このリファレンス・キットの主な利点は、コードをほとんどまたは全く変更することなく、ヘテロジニアスな XPU アーキテクチャー全体でモデルのトレーニングと推論を最適化できることです。インテル® AI アナリティクス・ツールキット（AI キット）とインテル® oneAPI データ・アナリティクス・ライブラリー（インテル® oneDAL）がこれを可能にします。このガイドで使用する AI キットの主なライブラリーは、Modin 向けインテル® ディストリビューション（英語）、scikit-learn* 向けインテル® エクステンション（英語）、XGBoost 向けインテル® オプティマイゼーション（英語）です。これらはすべて、AI キットの一部またはスタンドアロン・ライブラリーとしてダウンロードできます。さらに、インテル® oneDAL の daal4py（英語）を使用して、XGBoost モデルの推論を高速化します。

データ処理

まず、Modin を使用してデータを処理および調査します。Modin は、pandas ワークフローをデータセットのサイズに合わせてスケーリングするように設計された分散型 DataFrame ライブラリーであり、1MB ~ 1TB 以上のデータセットをサポートします。pandas では、一度に 1 つのコアのみが使用されます。Modin の Dask エンジンは、利用可能なすべての CPU コアを利用できるため、非常に大きなデータセットをはるかに高速に処理できます。

図 2 は、2GB のデータセットに対して同じ pandas 操作を実行する Modin（左）と pandas（右）の例です。2 つの例の唯一の違いは、インポート文です。

© 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。

図 2. Modin (左) と pandas (右) のパフォーマンスの比較

Dask エンジンで Modin を使用するには、次のコードセルに示すように、最初に pandas のドロップイン置換をインポートしてから、エンジン呼び出し文で Dask 実行環境を初期化します。

```
import modin.pandas as pd
from modin.config import Engine
Engine.put("dask")
```

探索的データ分析

Modin の Dask エンジンが初期化されると、利用可能な CPU コア間で並列に処理される同じ pandas 関数を使用して、データの処理と調査を続行できます。**図 3** の State 1 に示すように、データ内の電柱の約 40% に不具合があると識別されました。ターゲット分布のわずかな不均衡を考慮して、交差検証中に層化抽出法を使用します。データをさらに詳しく調べるには、こちらのノートブック（英語）を参照してください。

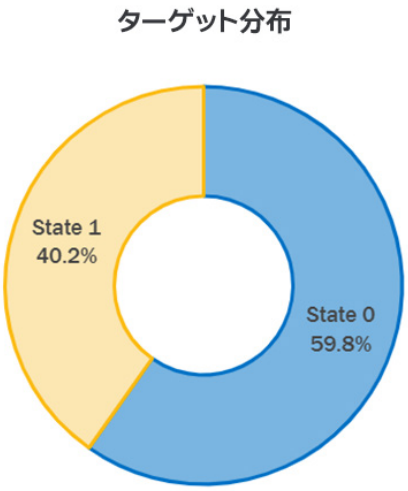

図 3. ターゲット変数 Asset Label の分布

© 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。

ソリューションの構築

電柱の故障の確率を予測するため、いくつかのマシンラーニング・アルゴリズムを検討しました。データの非線形性のため、サポートベクトル分類器（SVC）と XGBoost の 2 つのノンパラメトリック・モデルを選択しました。どちらも、特徴とターゲット変数の間の非線形関係を捉えることができる高性能の分類器です。しかし、ハイパーパラメーターをチューニングし、大規模な産業用データセットのモデルをトレーニングするには、長い時間がかかります。ここで、インテル® AI アナリティクス・ツールキットの出番です。

サポートベクトル分類器による分析

SVC モデルのトレーニングと推論を最適化するため、scikit-learn* 向けインテル® エクステンションを利用して、インテルの CPU と GPU の両方で、ベクトル命令、スレッド化、メモリー最適化を使用することで、アルゴリズムの実行時間を短縮します。scikit-learn* 向けインテル® エクステンションは、サポート・ベクトル・マシン、K 近傍法、ランダムフォレストなど、scikit-learn* ライブラリー内の多くの推定器と関数をサポートしています。

これを実装するには、いくつかの方法があります。サポートされているすべてのアルゴリズムをグローバルに動的にパッチするには、以下のコードセルに示す関数を呼び出すだけです。

```
from sklearnex import patch_sklearn
patch_sklearn()
```

このパッチは、サポートされている stock scikit-learn* アルゴリズムを最適化されたバージョンに置き換えます。そして、コードにほかの変更を加えることなく、同じ AI パッケージと scikit-learn* ライブラリーを引き続きインポートして使用できます。scikit-learn* 向けインテル® エクステンションの適用に関する詳細は、『デベロッパー・ガイド』（英語）を参照してください。

層化 3 分割交差検証を使用して SVC をチューニングし、見つかった最良のハイパーパラメーターは、正則化パラメーターが 10 の放射基底関数カーネルでした。サンプル外のテストセットでは、SVC は 0.919 の ROC 曲線下面積（AUC）と 0.902 の平均精度（AUPRC）スコアを達成しました（**図 4**）。

© 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。

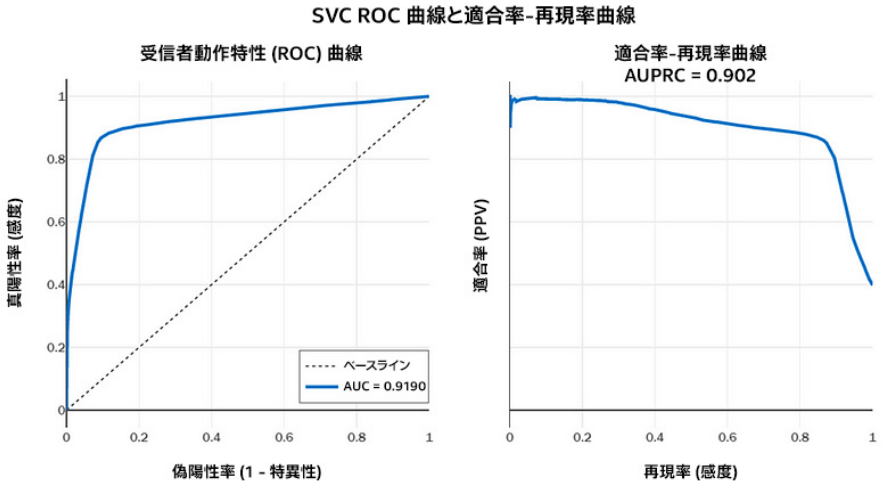

図 4. テストケースの SVC パフォーマンス

XGBoost 向けインテル® オプティマイゼーションによる分析

SVC のパフォーマンスをさらに向上できるかどうか確認するため、層化 3 分割交差検証を使用して、上記と同じ手順で XGBoost モデルをチューニングしました。リファレンス・キットの開発には、XGBoost v1.4.3 が使用されました。XGBoost v0.81 以降には、インテルがパッケージに直接アップストリームしている最適化が含まれています。次のグラフは、このデータセットのデフォルトのハイパーパラメーターを使用した stock XGBoost v0.81 と比較した、XGBoost v1.4.3 のトレーニング（**図 5**）と予測（**図 6**）時間の向上を示しています。

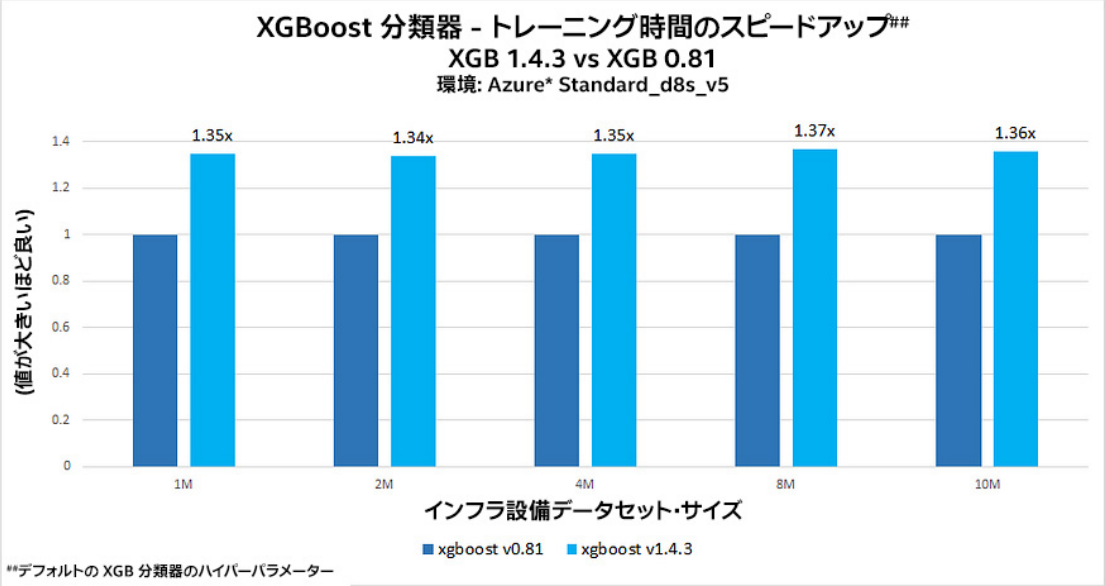

図 5. XGBoost v1.4.3 で利用可能なインテルの最適化は
stock XGBoost v0.81 と比較してトレーニングを約 1.37 倍高速化

© 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。

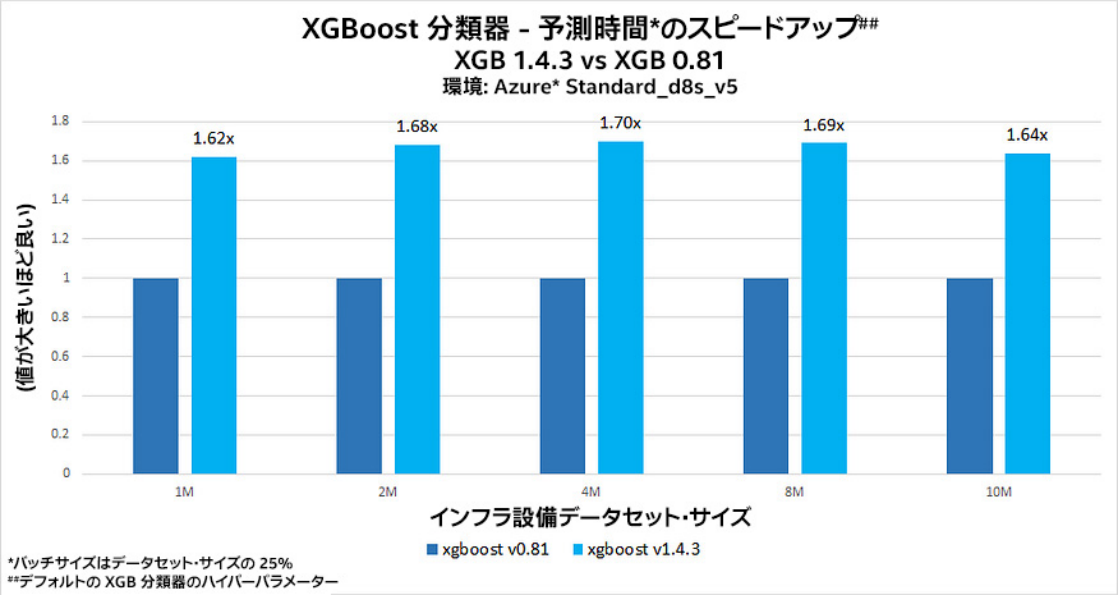

図 6. XGBoost v1.4.3 で利用可能なインテルの最適化は stock XGBoost v0.81 と比較して予測時間を 1.62 倍～ 1.70 倍向上

予測時間をさらに向上するため、チューニングした XGBoost モデルを daal4py モデルに変換できます。daal4py は、oneDAL（英語）向けの Python* API です。インテル® AVX-512 のベクトル化を使用して、インテル® Xeon® プロセッサーで勾配ブースティングのパフォーマンスを最大化します（**図 7**）。

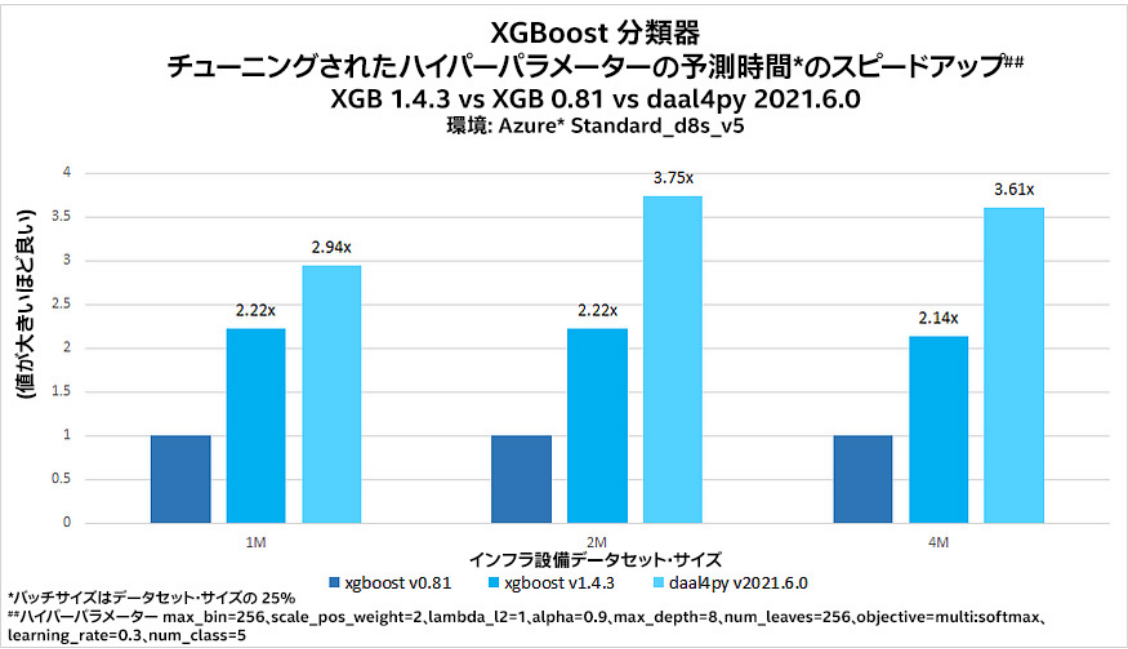

図 7. stock XGBoost v0.81 と比較した XGBoost v1.4.3 と daal4py v2021.6.0 で利用可能なインテルの最適化による予測時間のスピードアップ

© 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。

チューニングしたハイパーパラメーターにより、XGBoost 向けインテル® オプティマイゼーションは、2.14 倍~ 2.22 倍のスピードアップを実現し、予測時間を短縮しました。daal4py は予測時間をさらに向上し、XGBoost v0.81 と比較して 2.94 倍~ 3.75 倍の高速化をもたらしました。チューニングした XGBoost モデルから daal4py への変換は、以下のコード行で簡単に行えます。

```
import daal4py as d4p
daal_model = d4p.get_gbt_model_from_xgboost(xgb.get_booster())
```

そして、トレーニング済みモデルを入力データとともに daal4py の予測関数に送り、テストセットでの確率を計算できます。

```
daal_prob = d4p.gbt_classification_prediction(nClasses = 2, resultsToEvaluate = “comp
uteClassLabels|computeClassProbabilities” ,
fptype = ‘float’ ).compute(X_test, daal_model).probabilities
```

図 8 は、サンプル外のテストセットにおいて、daal4py により高速化した XGBoost モデルが達成した予測パフォーマンスを示しています。XGBoost は、AUC が 0.9401、平均精度 が 0.937 と、SVC を凌駕する結果を示しました。

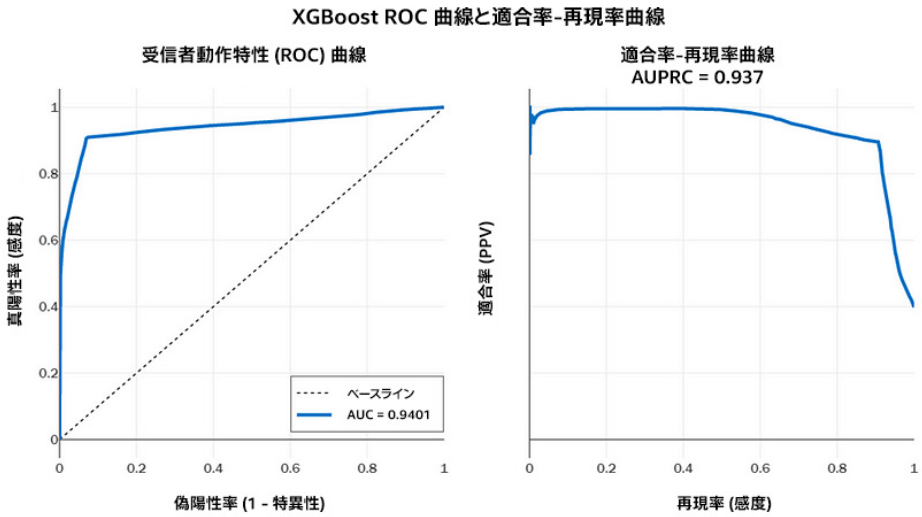

図 8. daal4py 最適化を使用したテストセットでの XGBoost のパフォーマンス

まとめ

大規模な保守予測ソリューションでは、通常、トレーニングと推論の高速化が必要です。インテル® AI アナリティクス・ツールキットで利用可能な最適化を使用して、電柱の故障の可能性を予測する効率良いエンドツーエンドの確率的分類ツールを開発しました。トレーニングと推論を高速化した結果、数十万件のインフラ設備の健全性予測を作成する計算時間の短縮、生産性の向上、コスト軽減を実現しました。このリファレンス・キットは、同様の業界やユースケースに簡単にスケーリングできる、公益事業者向けの設備保守予測のパフォーマンス最適化ガイドを提供します。

© 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。

インテル® End-to-End AI Optimization Kit による人工知能の高速化

インテルにより最適化されたソフトウェアで AI を大衆化

Jian Zhang インテル コーポレーション AI ソフトウェア・エンジニアリング・マネージャー

人工知能（AI）は、ヘルスケア、小売、製造業など、事実上すべての業界に革命をもたらしました。しかし、今日の AI ソリューションのほとんどは高コストであり、その利用は少数のデータ・サイエンティストに限定されています。これは複数の要因によるものです。第一に、最新のエンドツーエンドの AI パイプラインは複雑です。データ処理、特徴量エンジニアリング、モデルの開発、モデルのデプロイ、保守など、複数の段階が必要です。これらの段階の反復的な性質により、プロセスには時間がかかります。第二に、AI ソリューションの開発には、多くの場合、深い専門知識が必要です。これは、初心者や一般のデータ・サイエンティストにとって参入の障壁となります。第三に、人々は精度を高めるため、より大きく、より深いモデルを開発する傾向があります。これらの「過度にパラメーター化された」モデルは多大な計算量を必要とし、リソースに制約のある環境でのデプロイを妨げます。

© 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。

インテル® End-to-End AI Optimization Kit（英語）は、エンドツーエンドの AI パイプラインをより高速に、シンプルに、アクセスしやすくし、どこでも、誰でも AI にアクセスできるようにするために開発されました。一般的なハードウェア上でハイパフォーマンスかつ軽量なモデルを効率良く提供する、エンドツーエンドの AI 最適化を実現する構成可能なツールキットです。このツールキットは、PyTorch* 向けインテル® エクステンション（IPEX）（英語）、TensorFlow* 向けインテル® エクステンション（ITEX）（英語）、インテル® AI アナリティクス・ツールキット（AI キット）など、インテルが最適化した一連のフレームワークに基づいて構築されています。また、ハイパーパラメーター最適化のため、SigOpt（英語）を統合します。インテル® End-to-End AI Optimization Kit は、データ準備、モデル最適化、およびモデル構築向けの独自のコンポーネントと機能も提供します。

エンドツーエンドの AI パイプラインのスケールアップとスケールアウトの効率を向上し、複雑なディープラーニング（DL）モデルの「夜間トレーニング」を可能にします。推論スループットが高く、リソース要件が低い軽量の DL モデルを提供します。また、エンドツーエンドの AI をよりシンプルにします。Click-to-Run ワークフローと SigOpt AutoML を使用してパイプラインを自動化し、データ処理と特徴量エンジニアリングの複雑な API を抽象化し、分散トレーニングを簡素化し、既存またはサードパーティーのマシンラーニング（ML）ソリューションやプラットフォームと簡単に統合できます。複雑な計算集約型の DL モデルを一般的なハードウェアで使用できるようにし、Smart Democratization Advisor（SDA）によって生成されたパラメーター化されたモデルを通じてビルトインの最適化されたモデルと、ニューラル・アーキテクチャー検索（NAS）テクノロジーで構築されたドメイン固有のコンパクト・ニューラル・ネットワークを提供します。これらすべてにより、データ・サイエンティストが AI にアクセスしやすくなります。

アーキテクチャー

さまざまなドメインの一般的なモデル (RecSys、CV、NLP、ASR、RL など) が、インテル® End-to-End AI Optimization Kit（**図 1**）の入力となります。これらのストックモデルは、重く、トレーニングに時間がかかり、チューニングや最適化が複雑です。モデルの種類に応じて、インテル® End-to-End AI Optimization Kit は、モデル・アドバイザーまたはニューラル・ネットワーク・コンストラクターのいずれかを使用してモデルを最適化します。最適化されたモデルは、FLOPS（1 秒あたりの浮動小数点演算数）とトレーニング時間がストックモデルの 1/10 で済み、精度の低下も同等か最小限に抑えられると期待されます。

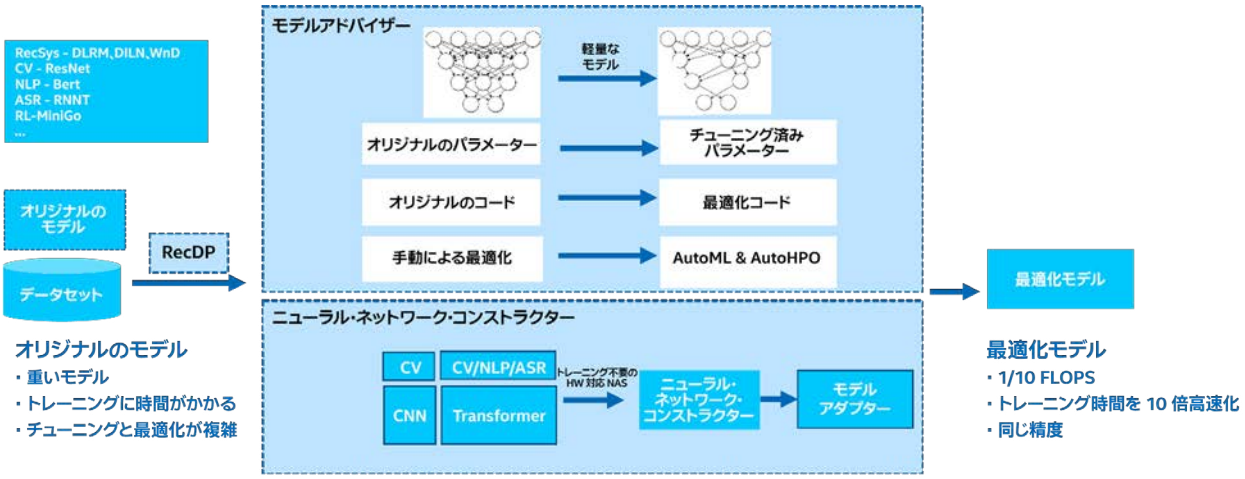

図 1. インテル® End-to-End AI Optimization Kit のアーキテクチャーとワークフロー

© 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。

RecDP

RecDP は、PySpark* 上に構築された並列データ処理および特徴量エンジニアリング・ライブラリーであり、Modin（英語）などのデータ処理ツールに拡張可能です。主な特徴と機能は次のとおりです。

- 表形式のデータセット処理ツールキット
- Spark* プログラミングの複雑さを隠す抽象 API
- アダプティブ・クエリー・プランと戦略による最適化されたパフォーマンス
- ターゲット・エンコーディングやカウント・エンコーディングなどの一般的な特徴量エンジニアリング機能のサポート
- サードパーティー・ソリューションへの簡単な統合

RecDP は、パフォーマンスを向上するため、「遅延実行」を使用します。操作を融合し、データ統計のコレクションを活用して、データセットの不要なパススルーを回避します。これは、大規模なデータセットを処理する場合に重要です。RecDP は、Optimized Analytics Package for Spark* Platform（英語）によって提供される、ネイティブの単票形式の SQL エンジン機能を活用して、パフォーマンスを向上することもできます。

Smart Democratization Advisor（SDA）

SDA は、自動化を促進するユーザーガイド付きツールです。パラメーター化されたモデルによりビルトインのインテリジェンスを提供し、ハイパーパラメーター最適化（HPO）およびビルトインの最適化モデル（RecSys、CV、NLP、ASR、および RL など）に SigOpt を活用します。また、手動によるモデルのチューニングと最適化を変換して、AutoML と AutoHPO を支援します。

ニューラル・ネットワーク・コンストラクター

ニューラル・ネットワーク・コンストラクターは、ニューラル・アーキテクチャー検索テクノロジーに基づいています。定義済みのスーパーネットを使用して、特定のドメインのニューラル・ネットワーク構造を直接構築します。主な特徴と機能は次のとおりです。

- CV、NLP、ASR ドメインのモデルなど、マルチモデルのサポート
- 統合された Transformer ベースのスーパーネットを使用
- ハードウェア対応 NAS は、FLOPS やレイテンシーなどのメトリックをしきい値として使用してモデル・アーキテクチャーとモデルサイズを決定
- トレーニング不要の NAS は、候補を評価する際にトレーニング精度ではなく、ゼロコストのプロキシーメトリックを使用。トレーニング可能性、表現力、多様性、顕著性など、複数のネットワークの特徴が考慮されます
- モデルアダプターを活用して、ユーザーの運用環境にモデルをデプロイ。モデルアダプターは、微調整、知識蒸留、およびドメイン適応機能を提供する転移学習ベースのコンポーネントです

例

ここでは、ツールキットが DL 推奨モデル（DLRM）（英語）でどのように機能するか例を紹介します。これには、環境設定、データ処理、ビルトイン・モデル・アドバイザーとパッチコード、および最適化プロセス全体を開始する 1 行のコマンドが含まれます。

© 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。

ステップ 1 : 環境設定

インテル® End-to-End AI Optimization Kit は AI キット上に構築されているため、パイプラインの実行に必要なソフトウェアはすでに利用可能な状態です。

```
# DockerFile
FROM docker.io/intel/oneapi-aikit
ENV http_proxy=http://proxy-prc.intel.com:912
ENV https_proxy=http://proxy-prc.intel.com:912

# SigOpt
RUN python -m pip install sigopt==7.5.0 --ignore-installed

# PyTorch* conda
RUN conda activate pytorch
RUN python -m pip install prefetch_generator tensorboardX onnx tqdm lark-parser

# PyTorch* 向けインテル(R) エクステンション
RUN python -m pip install intel_extension_for_pytorch==1.10.0 -f https://software.intel.com/
ipex-whl-stable psutil
RUN mkdir -p /home/vmagent/app
WORKDIR /home/vmagent/app
```

ステップ 2 : RecDP による並列データ処理

次のステップでは、RecDP を使用してデータ処理を簡素化します。この例では、`Categorify()` と `FillNA()` の 2 つがチェーン操作であり、Spark* の遅延実行を使用してデータの不要なパススルーを減らしています。

```
from pyspark.sql import *
from pyspark import *
from pyspark.sql.types import *
from pyrecdp.data_processor import *
from pyrecdp.encoder import *
from pyrecdp.utils import *
import numpy as np

path_prefix = "file://"
current_path = "/home/vmagent/app/dataset/demo/processed/"
csv_file = "/home/vmagent/app/dataset/demo/criteo_mini.txt"

# 1. Spark* を起動して、データ・プロセッサーを初期化
t0 = timer()
spark = SparkSession.builder.master('local[*]') \
                            .config('spark.driver.memory','100G') \
                            .appName("DLRM").getOrCreate()
schema = StructType([StructField(f'_i{i}', IntegerType()) for i in range(0, 14)] )
df = spark.read.option('sep', '\t').option("mode", "DROPMALFORMED") \
               .schema(schema).csv(path_prefix + csv_file)
proc = DataProcessor(spark, path_prefix, current_path=current_path, spark_mode='local')

# 2. RecDP でデータを処理
CAT_COLS = list(range(14, 40))
to_categorify_cols = ['_c%d' % i for i in CAT_COLS]
op_categorify = Categorify(to_categorify_cols)
op_fillna_for_categorified = FillNA(to_categorify_cols, 0)
proc.append_ops([op_categorify, op_fillna_for_categorified])
df = proc.transform(df, name='dlrm')
t1 = timer()
print(f"Total process time is {(t1 - t0)} secs")
```

© 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。

ステップ 3 : Smart Democratization Advisor（SDA）と AutoML

SDA は、手動による最適化を変換して AutoML を支援します。ビルトインモデルの定義済みパラメーターを提供することで、AutoML の作業時間を大幅に短縮できます。

```
model_info = dict()

# モデルの設定
model_info["score_metrics"] = [("accuracy", "maximize"), ("training_time", "minimize")]
model_info["execute_cmd_base"] = "python launch.py"
model_info["result_file_name"] = "best_auc.txt"

# SigOpt の設定
model_info["experiment_name"] = "dlrm"
model_info["sigopt_config"] = [
    {'name':'learning_rate','bounds':{'min':5,'max':50},'type':'int'},
    {'name':'lamb_lr','bounds':{'min':5,'max':50},'type':'int'},
    {'name':'warmup_steps','bounds':{'min':2000,'max':4500},'type':'int'},
    {'name':'decay_start_steps','bounds':{'min':4501,'max':9000},'type':'int'},
    {'name':'num_decay_steps','bounds':{'min':5000,'max':15000},'type':'int'},
    {'name':'sparse_feature_size','grid': [128,64,16],'type':'int'},
    {'name':'mlp_top_size','bounds':{'min':0,'max':7},'type':'int'},
    {'name':'mlp_bot_size','bounds':{'min':0,'max':3},'type':'int'}]
model_info["observation_budget"] = 1
```

いくつかの設定可能なパラメーターのほかに、コードレベルの最適化が必要なケースがあります。すべてのビルトインモードに対して、キットは最適化されたモデルとパッチコードを提供します。この例では、IPEX と BF16 に加え、複数 CPU ノードでのモデルの収束を改善するオプティマイザーを使用します。

```
# フレームワークの最適化： IPEX & BF16 を使用

# 最適化を埋め込む
-       m_curr = (m[i] if self.max_emb_dim > 0 else m)
-       EE = nn.EmbeddingBag(n, m_curr, mode="sum", sparse=True)
-       W = np.random.uniform(
-           low=-np.sqrt(1 / n), high=np.sqrt(1 / n), size=(n, m_curr)
-       ).astype(np.float32)
        # 大衆化、2 次元の疎行列/密行列を使用
+       W = np.random.uniform(
+           low=-np.sqrt(1 / n), high=np.sqrt(1 / n), size=(n, m)
+       ).astype(np.float32)

# オプティマイザーの最適化
- optimizer = torch.optim.SGD(dlrm.parameters(), lr=args.learning_rate) …)
```

ステップ 4 : エンドツーエンドのモデル最適化

最後に、数行のコードでエンドツーエンドのモデル最適化を開始します。

```
import sys
from e2eAIOK import SDA
sda = SDA(model ="dlrm", settings=setting)
sda.launch()
```

© 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® End-to-End AI Optimization Kit は、次のような一般的なモデル向けの多くの Click-to-Run を提供しています。

- RecSys : **DLRM**（英語）、**DIEN**（英語）、**WnD**（英語）
- 自動音声認識（ASR）: **RNNT**（英語）
- コンピューター・ビジョン（CV）: **RESNET**（英語）
- 自然言語処理（NLP）: **BERT**（英語）
- 強化学習（RL）: **MiniGO**（英語）

パフォーマンス

テストは、4 ノードのクラスターで実施されました（**表 1**）。各ノードには、2 基のインテル® Xeon® Gold 6240 プロセッサーと 384GB のメモリーが搭載されており、ノードは 25GB イーサネットを介して接続されました。また、1TB HDD 1 台をデータドライブとして使用しました。

設定	説明
テスト実施日	2021 年 12 月
プラットフォーム	S2600WFT
CPU	インテル® Xeon® Gold 6240
ノード数	4
ノードごとの CPU 数	2 ソケット、ソケットごとに 18 コア、コアごとに 2 スレッド
メモリー	DDR4 デュアルランク 384G、12 スロット /32GB/2666MHz
ストレージ	1x 400GB インテル® SSD（SSDSC2BA400G3）OS ドライブ 1TB HDD（データストレージ）
ネットワーク	1x インテル® イーサネット・ネットワーク・アダプター X722、 1x インテル® イーサネット・ネットワーク・アダプター XXV710
マイクロコード	0x500002C
BIOS バージョン	SE5C620.86B.0X.02.0094.102720191711
OS/ ハイパーバイザー /SW	Fedora* 29 5.3.11-100.fc29.x86_64

表 1. システム構成

3 つの一般的な RecSys モデルで、ストックモデルと最適化されたモデルのパフォーマンスを比較しました（**表 2**）。最適化されたモデルは、3 つのワークロードすべてで、ETL、トレーニング、および推論を大幅に高速化しました（**図 2**）。

© 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。

ワークロード	DLRM	WnD	DIEN
フレームワーク	PyTorch*	TensorFlow*	TensorFlow*
ライブラリー	oneMKL、oneCCL	oneMKL、oneCCL	oneMKL、oneCCL
データセット（サイズ、形状）	Criteo（英語）	outbrain（英語）	categoryFiles（英語）
精度（FP32、INT8、BF16）	BF16	-	-
KMP_AFFINITY	granularity=fine,compact,1,0	granularity=fine,compact,1,0	granularity=fine,compact,1,0
numactl	ソケット・バインディング	ソケット・バインディング	ソケット・バインディング
OMP_NUM_THREADS	20	16	4

表 2. テスト設定

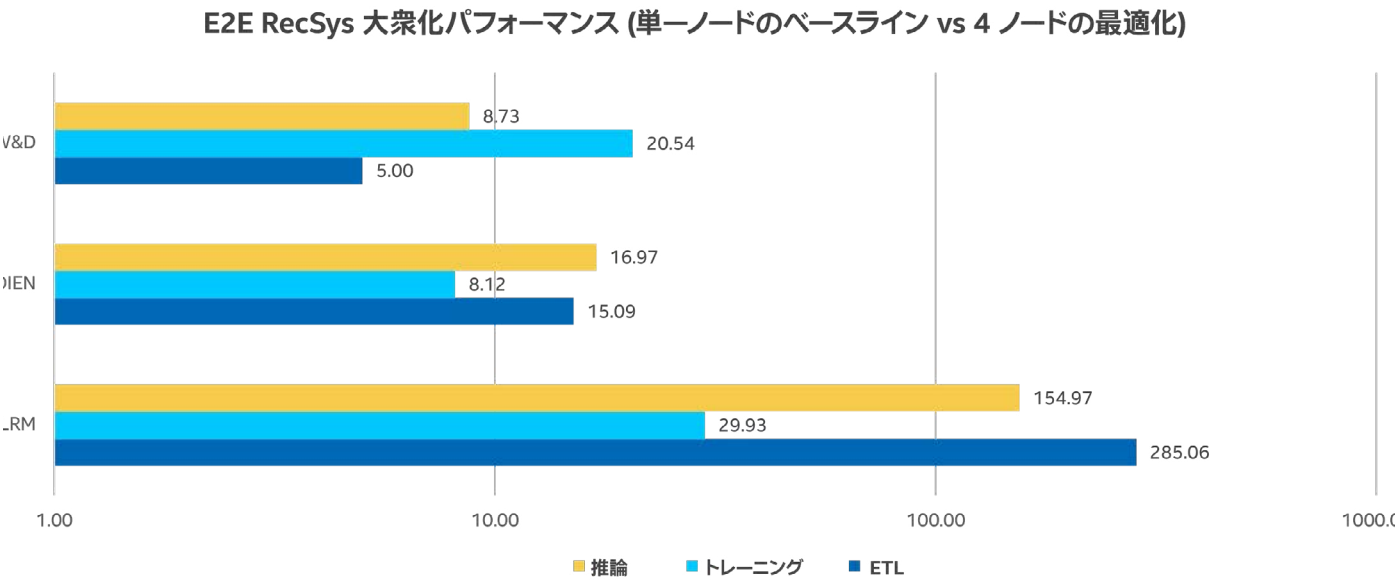

図 2. インテル® End-to-End AI Optimization Kit による RecSYS のスピードアップ。ベースライン構成：1 ノード、2x インテル® Xeon® Gold 6240 プロセッサー（18 コア）、インテル® ハイパースレッディング・テクノロジー有効、インテル® ターボ・ブースト・テクノロジー有効、384GB（12 スロット /32GB/2666MHz）メモリー、BIOS SE5C620 .86B.0X.02.0094.102720191711（ucode:0x500002C）、Fedora* 29（5.3.11-100.fc29.x86_64）、PyTorch*、TensorFlow*、Spark*。最適化構成：4 ノード、PyTorch* 向けインテル® エクステンション、TensorFlow* 向けインテル® エクステンション、Horovod*、変更した DLRM、WnD、および DIEN ワークロードを除いて同じハードウェアとソフトウェア構成。

© 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® End-to-End AI Optimization Kit は、一般的なハードウェア向けのハイパフォーマンスかつ軽量なモデルを提供する構成可能なツールキットであり、AI の大衆化に役立ちます。いくつかの重要な最適化を活用しています。

- RecDP による並列データ処理
- インテルにより最適化されたトレーニング・フレームワーク
- レイヤー数が少なく、分散メモリー型並列コンピューターで通信オーバーヘッドを軽減する軽量モデル
- 大きなバッチサイズで高速に収束するオプティマイザー・チューニング (DLRM)
- 特徴量エンジニアリング（埋め込みテーブルとエンコード）の最適化

このツールキットは、一般的なモデルで良好な結果をもたらします。プロジェクトで使用する場合は、https://github.com/intel/e2eAIOK/（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。

Fortran と OpenMP* でヘテロジニアス・プログラミングの課題を解決

オープンな標準プログラミング言語でヘテロジニアス並列処理を表現

Shiquan Su 博士 インテル コーポレーション ソフトウェア・テクニカル・コンサルティング・エンジニア
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
Barbara Perz インテル コーポレーション ソフトウェア・テクニカル・コンサルティング・エンジニア
Pamela Harrison インテル コーポレーション ソフトウェア・テクニカル・コンサルティング・エンジニア

急速な技術革新は、ヘテロジニアス・コンピューティングの新しい時代をもたらしています。ハードウェアの多様性と増大する計算能力の要求により、ヘテロジニアス並列処理を利用できるプログラミング・モデルが求められています。これらのモデルは、さまざまなベンダーのハードウェアでプログラムを実行できるように、オープンで、移植性に優れていなければなりません。Fortran は古い言語ですが、今でも科学や工学の分野で活発に利用されている重要なプログラミング言語です。同様に、1997 年にリリースされたコンパイラー主導の並列処理のオープンな標準である OpenMP* も、ヘテロジニアス・コンピューティングをサポートするように進化し、計算をアクセラレーター・デバイスにオフロードして不連続メモリー間でデータを移動するディレクティブが含まれるようになりました。開発者は、ホストとデバイスメモリーの概念、テクスチャー / サーフェスや定数メモリーなどのより繊細なメモリータイプを、OpenMP* ディレクティブを通じて利用できます。

© 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。

タスクをアクセラレーターにオフロードすると、一部の計算はより効率的になります。例えば、高度なデータ並列計算では、GPU の多くの処理要素を活用できます。この記事では、Fortran + OpenMP* がヘテロジニアス・コンピューティングの 3 つの主な課題である、アクセラレーターへの計算のオフロード、不連続メモリーの管理、ターゲットデバイス上での既存の API の呼び出し、をどのように解決するかを示します。

アクセラレーターへの計算のオフロード

最初に、簡単な例から始めましょう。**リスト 1** は、OpenMP* の `TARGET`、`TEAMS`, および `DISTRIBUTE PARALLEL DO` 構文を使用して入れ子のループを実行する方法を示しています。`TARGET` 構文はターゲットデバイス上の並列領域を作成します。`TEAMS` 構文はチームのリーグ（つまり、スレッドのグループ）を作成します。この例では、チームの数は `num_blocks` パラメーター以下です。各チームには、`block_threads` 変数以下の数のスレッドがあります。各チームのプライマリー･スレッドは､`teams` 領域でコードを実行します。外部ループの反復は、各チームのプライマリー・スレッドに分散されます。チームのプライマリー・スレッドが `DISTRIBUTE PARALLEL DO` 構文を検出すると、チーム内のほかのスレッドがアクティブになります。チームは並列領域を実行した後、内部ループの実行をワークシェアします。この一連の処理を図式化したのが**図 1** 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。

```
PROGRAM target_teams_distribute
   EXTERNAL saxpy

   INTEGER, PARAMETER :: n = 2048, num_blocks = 64
   REAL,  ALLOCATABLE :: A(:), B(:), C(:)
   REAL               :: d_sum = 0.0
   INTEGER            :: i, block_size = n / num_blocks
   INTEGER            :: block_threads = 128

   ALLOCATE(A(n), B(n), C(n))
   A = 1.0
   B = 2.0
   C = 0.0

   CALL saxpy(A, B, C, n, block_size, num_blocks, block_threads)

   DO i = 1, n
      d_sum = d_sum + C(i)
   ENDDO

   PRINT '("sum = 2048 x 2 saxpy sum:"(f))', d_sum

   DEALLOCATE(A, B, C)
END PROGRAM target_teams_distribute

SUBROUTINE saxpy(B, C, D, n, block_size, num_teams, block_threads)
   REAL    :: B(n), C(n), D(n)
   INTEGER :: n, block_size, num_teams, block_threads, i, i0

   !$OMP TARGET MAP(TO: B, C) MAP(TOFROM: D)
   !$OMP TEAMS num_teams(num_teams) thread_limit(block_threads)
   DO i0 = 1, n, block_size
      !$OMP DISTRIBUTE PARALLEL DO
      DO i = i0, min(i0 + block_size - 1, n)
         D(i) = D(i) + B(i) * C(i)
      enddo
   ENDDO
   !$OMP END TEAMS
   !$OMP END TARGET
END SUBROUTINE
```

リスト 1. OpenMP* ディレクティブ（青で表示）を使用して入れ子のループをアクセラレーターにオフロード

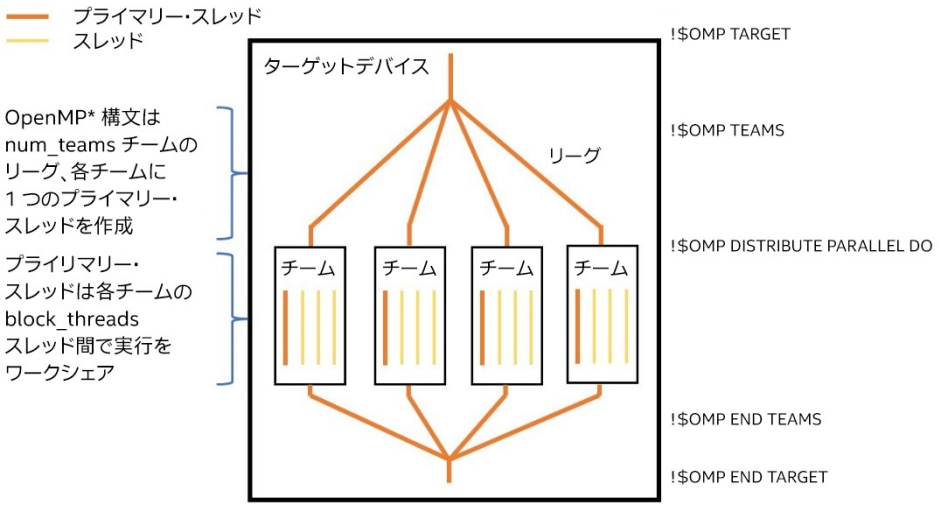

図 1. OpenMP* TARGET、TEAMS、および DISTRIBUTE PARALLEL DO 領域の概念図

© 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。

ホストとデバイス間のデータ転送

次に、ホストとデバイス間のメモリー管理とデータ移動について説明します。OpenMP* には 2 つのアプローチが用意されています。1 つ目のアプローチは、DATA 構文を使用して、不連続メモリー間のデータをマップします。例えば、**リスト 1** は、TARGET ディレクティブの MAP(TO: B, C) 節と MAP(TOFROM: D) 節で配列 B、C、および D をデバイスにコピーし、デバイスから D の最終値を取得します。2 つ目のアプローチは、OpenMP* ランタイム・ライブラリー・ルーチンの 1 つ、デバイス・メモリー・アロケーターの呼び出しです。この記事では、2 つ目のアプローチは説明しません。

リスト 2 は、TARGET DATA 構文で新しいデバイスデータ環境（ターゲットデータ領域とも呼ばれる）を作成し、配列 A、B、および C をマップしています。TARGET DATA 領域は、2 つの TARGET 領域を囲んでいます。最初の領域は、MAP(TO: A, B) および MAP(FROM: C) データモーション節に従って囲んでいるデバイスデータ環境から A、B、および C を継承する新しいデバイスデータ環境を作成します。ホストは、1 つ目のターゲット領域が完了するのを待ってから、データ環境の A と B に新しい値を割り当てます。TARGET UPDATE 構文は、デバイスデータ環境で A と B を更新します。2 つ目のターゲット領域が終了すると、デバイスデータ環境の終了時に C の結果がデバイスからホストメモリーにコピーされます。この一連の処理を図式化したのが**図 2** です。

ポッドキャスト
災害管理の使命と科学的発見
聴く（英語）

アニメーター、デジタルコンテンツ制作者、建築エンジニア、熟練ゲーマーの皆さん、インテル® oneAPI ベース & レンダリング・ツールキットでビジュアライゼーションの限界に挑戦してみてください。詳細 >
intel 1 oneAPI RENDERING TOOLKIT

© 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。

```
PROGRAM target_data_update
   INTEGER           :: i, n = 2048
   REAL, ALLOCATABLE :: A(:), B(:) ,C(:)
   REAL              :: d_sum = 0.0

   ALLOCATE(A(n), B(n), C(n))
   A = 1.0
   B = 2.0
   C = 0.0

   !$OMP TARGET DATA MAP(TO: A, B) MAP(FROM: C)
   !$OMP TARGET
   !$OMP PARALLEL DO
   DO i = 1, n
      C(i) = A(i) * B(i)
   ENDDO
   !$OMP END TARGET

   A = 2.0
   B = 4.0

   !$OMP TARGET UPDATE TO(A, B)

   !$OMP TARGET
   !$OMP PARALLEL DO
   DO i = 1, n
      C(i) = C(i) + A(i) * B(i)
   ENDDO
   !$OMP END TARGET
   !$OMP END TARGET DATA

   DO i = 1, n
      d_sum = d_sum + C(i)
   ENDDO

   PRINT '("sum = 2048 x (2 + 8) sum:"(f))', d_sum

   DEALLOCATE(A, B, C)
END PROGRAM target_data_update
```

リスト 2. デバイスデータ環境の作成

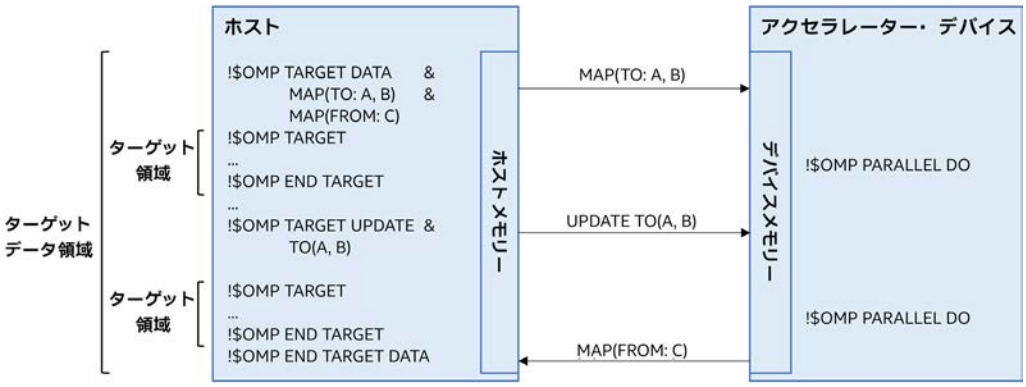

図 2. リスト 2 の OpenMP* プログラムにおけるホストとデバイス間のデータ転送。
各矢印は、ホストとデバイスのメモリー間のデータ転送を示します。

© 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。

Linux* でインテル® Fortran コンパイラーと OpenMP* ターゲットオフロードを使用してサンプルプログラムをコンパイルするコマンドを次に示します。

```
$ ifx -xhost -qopenmp -fopenmp-targets=spir64 source_file.f90
```

OpenMP* ターゲット領域から既存の API を使用する

OpenMP* ターゲット領域から外部関数を呼び出す方法は、前号の記事、「Fortran、oneMKL、OpenMP* を使用して LU 因数分解を高速化」で説明しました。DISPATCH ディレクティブで、関連するサブルーチンまたは関数呼び出しの周囲で条件付きディスパッチ・コードを出力するようにコンパイラーに指示します。

```
!$OMP TARGET DATA
!$OMP DISPATCH
CALL external_subroutine_on_device
!$OMP END TARGET DATA
```

ターゲットデバイスが利用可能な場合、構造化ブロックのバリアントバージョンが呼び出されます。

インテル® Fortran のサポート

インテル® Fortran コンパイラー（ifx）は、インテル® Fortran コンパイラー・クラシック（ifort）のフロントエンドとランタイム・ライブラリーをベースに、LLVM バックエンドを使用した新しいコンパイラーです。詳細は、『インテル® Fortran コンパイラー・クラシックおよびインテル® Fortran コンパイラー・デベロッパー・ガイドおよびリファレンス』（英語）を参照してください。バイナリー（.o/.obj）およびモジュール（.mod）は ifort と互換性があり、さまざまな Fortran 標準規格（95、2003、2018）と OpenMP* を利用したヘテロジニアス・コンピューティングをサポートしています。Fortran を利用したヘテロジニアス並列処理の別のアプローチ（英語）は、DO CONCURRENT ループです。

```
PROGRAM test_auto_offload
   INTEGER, PARAMETER :: N = 100000
   REAL               :: a(N), b(N), c(N), sumc

   a = 1.0
   b = 2.0
   c = 0.0
   sumc = 0.0

   CALL add_vec

   DO i = 1, N
      sumc = sumc + c(i)
   ENDDO

   PRINT *,' sumc = 300,000 =', sumc

   CONTAINS
      SUBROUTINE add_vec
         DO CONCURRENT (i = 1:N)
            c(i) = a(i) + b(i)
         ENDDO
      END SUBROUTINE add_vec

END PROGRAM test_auto_offload
```

© 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。

このコードを次のようにコンパイルすると、OpenMP* ランタイム・ライブラリーがデバイス・カーネル・コードを生成します。

```
$ ifx -xhost -qopenmp -fopenmp-targets=spir64 -fopenmp-target-do-concurrent source_file.f90
```

`-fopenmp-target-do-concurrent` オプションは、`DO CONCURRENT` のデバイスカーネルを自動的に生成するようにコンパイラーに指示します［**翻訳者注：**このコードには OpenMP* のディレクティブはありませんが、インテル® Fortran コンパイラーは `DO CONCURRENT` ループの並列処理に OpenMP* ランタイムを使用するため、`-qopenmp` と `-fopenmp-targets=spir64` オプションが必要です］。

次の環境変数を設定することにより、OpenMP* ランタイムは、カーネル・アクティビティーのプロファイルを提供できます。

```
$ export LIBOMPTARGET_PLUGIN_PROFILE=T
```

実行ファイルを実行すると、出力が得られます。

プログラムを実行したときの出力で、サブルーチン名 `add vec` を探します。

```
Kernel 0 : __omp_offloading_3b_dd004710_test_auto_offload_IP_add_vec__l10
```

Fortran 言語委員会は、Fortran 2023 標準規格に `DO CONCURRENT` のリダクションを追加する提案に取り組んでいます。

```
DO CONCURRENT(i = 1:N) REDUCE(+:sum)
```

最後に

Fortran と OpenMP* を使用したヘテロジニアス並列プログラミングの概要を説明しました。上記のコード例から分かるように、OpenMP* は、一般にほかへの影響がない並列処理を表現するための記述的なアプローチです。つまり、OpenMP* ディレクティブが有効でない場合でも、シーケンシャル・プログラムはそのまま残ります。このコードは、アクセラレーター・デバイスがない、ホモジニアス・プラットフォームでも動作します。Fortran + OpenMP* は、ヘテロジニアス並列処理のための、強力で、オープンな、標準ベースのアプローチです。

© 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。

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

oneMKL と OpenMP* ターゲットオフロードで線形システムを解く

標準ベースのオープンなアプローチで線形代数計算を高速化

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
Nawal Copty インテル コーポレーション シニア・ソフトウェア・エンジニア

前号の記事、「Fortran、oneMKL、OpenMP* を使用して LU 因数分解を高速化」では、LU 因数分解と後続の反転をアクセラレーターにオフロードする方法を紹介しました。OpenMP* オフロード領域を統合して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でオーバーヘッドを削減するヒントをいくつか示しましたが、連立線型方程式を解くには至りませんでした。これは読者の演習として残しておいたものですが、『oneAPI GPU 最適化ガイド』の「データ転送とメモリー割り当てを最小化する」のいくつかのテクニックを説明するためにここで実行します。

OpenMP* を使用してアクセラレーターにオフロードされた oneMKL の LU 因数分解とソルバー関数を使用してバッチ処理された線形システムのグループを解く方法を示します。逆行列の因数は実際にはほとんど使用しないため、このデモではそのステップを省略し、因数分解から解法に進みます。前のサンプルコードは Fortran で記述されていました。ここでも引き続き Fortran を使用します。

© 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。

まず、いくつかの小さな線形システムを読み込んで正しく解いていることを確認します。古い線形代数の教科書に適切な例がいくつか見つかりました（**図 1**）。行列は正方で、LU 因数分解を使用して解くことができます。

$$\begin{bmatrix} 2 & 4 & -4 \\ 1 & -4 & 3 \\ -6 & -9 & 10 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 12 \\ -21 \\ -24 \end{bmatrix} \qquad \begin{bmatrix} 0 & 0 & 3 \\ 2 & 4 & -1 \\ 6 & 5 & 5 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ -10 \\ -7 \end{bmatrix}$$

$$(x_1, x_2, x_3) = (-4, 2, -3) \qquad (x_1, x_2, x_3) = (-2, -1, 2)$$

図 1. oneMKL と OpenMP* ターゲットオフロードを使用して、これらの 2 つの線形システムをバッチで解きます。これらの小さな問題はテストには適していますが、高速化の恩恵を受けるには小さすぎます。

このプログラムは、前号の記事のプログラムと似ています。まず OpenMP* オフロードをサポートする Fortran インターフェイスをインクルードし、次に 32 ビット整数型と 64 ビット整数型のどちらを使用するかを決定します（**リスト 1**）（詳細は、「ILP64 インターフェイスと LP64 インターフェイスの使用」（英語）を参照）。この判断はコンパイル時に行われます（次のコンパイラー・コマンドを参照）。**図 1** のテスト問題は、それぞれ 1 つの右辺（RHS）がある 2 つの 3x3 線形システムで構成されています。`batch_size`、`n`、`nrhs`、および `stride` 変数がそれぞれ設定され、`a`、`b`、および `ipiv` 配列が割り当てられて、バッチ処理されたシステムが保持されます。最後の 4 つの文は、2 つの行列とその右辺をそれぞれ `a` と `b` にロードします。

インテル® DPC++ 互換性ツール
CUDA* アプリケーションを標準ベースの SYCL* コードに移行
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。

```
INCLUDE "mkl_omp_offload.f90"

PROGRAM solve_batched_linear_systems

! 32 ビット整数型と 64 ビット整数型のどちらを使用するか決定
#IF DEFINED(MKL_ILP64)
    USE onemkl_lapack_omp_offload_ilp64   ! 64 ビット
#ELSE
    USE onemkl_lapack_omp_offload_lp64    ! 32 ビット
#ENDIF

    INTEGER,         PARAMETER :: n = 3, batch_size = 2, nrhs = 1
    INTEGER                    :: lda, stride_a, stride_ipiv
    INTEGER                    :: ldb, stride_b
    REAL (KIND=8), ALLOCATABLE :: a(:,:), b(:,:)
    INTEGER,       ALLOCATABLE :: ipiv(:,:), info(:)

    INTEGER allocstat
    CHARACTER (LEN = 132) :: allocmsg

    lda         = n
    stride_a    = n * lda
    stride_ipiv = n
    ldb         = n
    stride_b    = n * nrhs

    ! 必要なメモリーを割り当て
    ALLOCATE (a(stride_a, batch_size), b(n, batch_size*nrhs), &
              ipiv(stride_ipiv, batch_size),                  &
              info(batch_size), STAT = allocstat, ERRMSG = allocmsg)
    if (allocstat > 0) STOP TRIM(allocmsg)

    ! 行列を初期化。Fortran は列優先言語であることに注意。
    a(:,1) = (/2.0, 1.0, -6.0, 4.0, -4.0, -9.0, -4.0,  3.0, 10.0/)
    a(:,2) = (/0.0, 2.0,  6.0, 0.0,  4.0,  5.0,  3.0, -1.0,  5.0/)
    b(:,1) = (/12.0, -21.0, -24.0/)        ! x = (-4,  2, -3)
    b(:,2) = (/ 6.0, -10.0,  -7.0/)        ! x = (-2, -1,  2)
```

リスト 1. サンプルプログラムのセットアップ。OpenMP* ターゲットオフロードをサポートする oneMKL ヘッダーとモジュールは、青色で表示しています。

計算を開始する準備が整いました。2 つの OpenMP* ターゲット領域を使用して LU 因数分解と解を線形システムにディスパッチする基本的な実装から始めましょう (**リスト 2**)。サンプルを次のようにコンパイルして実行し、正しい結果が得られることを確認します。

© 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。

```
$ ifx -i8 -DMKL_ILP64 -qopenmp -fopenmp-targets=spir64 -fsycl -free \
> lu_solve_omp_offload_ex1_small.F90 -o lu_solve_ex1_small \
> -L${MKLROOT}/lib/intel64 -lmkl_sycl -lmkl_intel_ilp64 -lmkl_intel_thread \
> -lmkl_core -liomp5 -lpthread -ldl

$ ./lu_solve_ex1_small

==================================
 Solutions to the linear systems
==================================
   -4.0000    2.0000   -3.0000
   -2.0000   -1.0000    2.0000
```

```
    ! OpenMP* オフロードを使用して LU 因数分解を計算
    ! 開始時に a は入力行列を格納
    ! 終了時に a は LU 因数分解された行列を格納
    !$OMP TARGET DATA MAP(TOFROM:a) MAP(FROM:ipiv) MAP(FROM:info)

        !$OMP DISPATCH
        CALL dgetrf_batch_strided(n, n, a, lda, stride_a, &
                                  ipiv, stride_ipiv, batch_size, info)
    !$OMP END TARGET DATA

    IF (ANY(info .ne. 0)) THEN
        PRINT *, 'Error: getrf_batch_strided returned with errors'
    ELSE
        ! 線形システムを解く。終了時に解を b に格納。
        !$OMP TARGET DATA MAP(TO:a) MAP(TO:ipiv) MAP(TOFROM: b) MAP(FROM:info)

            !$OMP DISPATCH
            CALL dgetrs_batch_strided('N', n, nrhs, a, lda, stride_a, &
                                      ipiv, stride_ipiv, &
                                      b, ldb, stride_b, batch_size, info)
        !$OMP END TARGET DATA

        IF (ANY(info .ne. 0)) THEN
            PRINT *, 'Error: getrs_batch_strided returned with errors'
        ELSE
            PRINT *, 'Computation executed successfully'
        ENDIF
    ENDIF
```

リスト 2. oneMKL と OpenMP* ターゲットオフロードを使用してバッチ処理された線形システムの基本的なソリューション。OpenMP* ディレクティブは青で表示しています。LAPACK の呼び出しは緑色で表示しています。

TARGET 構造は、アクセラレーター・デバイスに制御を転送します。最初の領域は、入力行列をデバイスメモリーに転送し [MAP(TOFROM:a)]、インプレース LU 因数分解をデバイスにディスパッチし (dgetrf_batch_strided)、LU 因数分解された行列 (a に格納)、ピボット・インデックス [MAP(FROM:ipiv)]、ステータス [MAP(FROM:info)] をデバイスメモリーから取得します。因数分解が正常に実行されると、プログラムは次の OpenMP* 領域に進みます。LU 因数分解された行列とピボット・インデックスをデバイスメモリーに戻し [MAP(TO:a) および MAP(TO:ipiv)]、線形システムをデバイスで解きます (dgetrs_batch_strided)。RHS は解ベクトルで上書きされ、計算 [MAP(FROM:info)] のステータスとともにデバイスメモリー [MAP(TOFROM:b)] から取得されます。ホストとデバイス間のデータ転送を図式化したのが**図 2** 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。

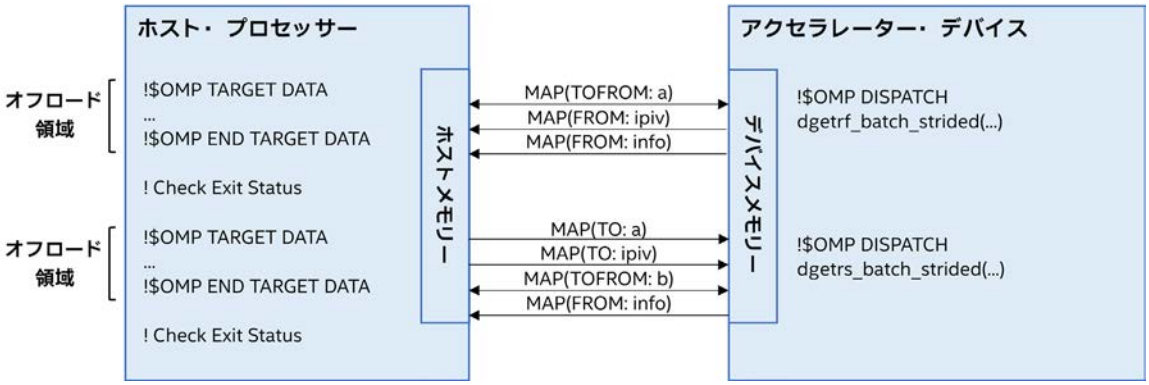

図 2. リスト 2 に示した 2 つの OpenMP* ターゲットオフロード領域のホストとデバイス間のデータ転送。各矢印は、ホストとデバイスのメモリー間のデータ転送を示します。

ただし、ホストとデバイス間のデータ転送は、『oneAPI GPU 最適化ガイド』の「データ転送とメモリー割り当てを最小化する」で説明しているように、ランタイム・ライブラリーからデバッグ情報を要求することで、データの移動を直接確認できます。テスト問題は小さすぎて高速化を証明できないため、問題のサイズを増やします。それぞれ 1 つの RHS がある 8 つの 8,000x8,000 線形システムのバッチを（倍精度で）解いてみましょう。

```
$ OMP_TARGET_OFFLOAD=MANDATORY ZE_AFFINITY_MASK=0.0 LIBOMPTARGET_DEBUG=1 \
> ./lu_solve_ex1 8000 8 1 1 >& lu_solve_ex1.out

$ grep Moving lu_solve_ex1.out

Libomptarget --> Moving 64 bytes (hst:0x00007fffa0074c28) -> (tgt:0x0000000003871008)
Libomptarget --> Moving 88 bytes (hst:0x00007fffa0074b48) -> (tgt:0x0000000003871088)
Libomptarget --> Moving 4096000000 bytes (hst:0x00007f1d07fff200) -> (tgt:0x00007f1c0dedd000)
Libomptarget --> Moving 88 bytes (hst:0x00007fffa0074cd8) -> (tgt:0x0000000003871108)
Libomptarget --> Moving 4096000000 bytes (tgt:0x00007f1c0dedd000) -> (hst:0x00007f1d07fff200)
Libomptarget --> Moving 512000 bytes (tgt:0x00007f1d8ce05000) -> (hst:0x00007f1da079b280)
Libomptarget --> Moving 64 bytes (tgt:0x0000000003882000) -> (hst:0x00007f1da0a33dc0)
Libomptarget --> Moving 64 bytes (hst:0x00007fffa0074c28) -> (tgt:0x0000000003871008)

Libomptarget --> Moving 512000 bytes (hst:0x00007f1da075b240) -> (tgt:0x000000000576b000)
Libomptarget --> Moving 88 bytes (hst:0x00007fffa0074c78) -> (tgt:0x0000000003871088)
Libomptarget --> Moving 512000 bytes (hst:0x00007f1da079b280) -> (tgt:0x00007f1d8ce05000)
Libomptarget --> Moving 88 bytes (hst:0x00007fffa0074b48) -> (tgt:0x0000000003871108)
Libomptarget --> Moving 4096000000 bytes (hst:0x00007f1d07fff200) -> (tgt:0x00007f1c0dedd000)
Libomptarget --> Moving 88 bytes (hst:0x00007fffa0074cd8) -> (tgt:0x0000000003871188)
Libomptarget --> Moving 512000 bytes (tgt:0x000000000576b000) -> (hst:0x00007f1da075b240)
Libomptarget --> Moving 64 bytes (tgt:0x0000000003882000) -> (hst:0x00007f1da0a33dc0)
```

OpenMP* `MAP` 構文を使用して明示的に転送される配列をハイライトし、2 つの OpenMP* ターゲット領域を区切るスペースを追加しました。ハイライトされていないデータ移動は、ターゲットデバイスにマップされている配列のドープベクトル（英語）です。一般に小さいため、無視できます。

1 つ目のターゲット領域で、配列 `a`（4,096,000,000 バイト）がホストからターゲットへ（`hst → tgt`）、およびターゲットからホストへ（`tgt → hst`）転送され［`MAP(TOFROM:a)`］、2 つ目のターゲット領域でターゲットデバイスに転送されます［`MAP(TO:a)`］。ピボット（`ipiv`、512,000 バイト）は、1 つ目のターゲット領域のデバイスで計算されて取得され［`MAP(FROM:ipiv)`、`tgt → hst`］、2 つ目のターゲット領域のデバイスに転送されて［`MAP(TO:ipiv)`、`hst → tgt`］、解が計算されます。配列 `b`（512,000 バイト）に格納されている RHS は、デバイスに転送され、ソリューション・ベクトルで上書きされて、ホストに転送されます［`MAP(TOFROM:b)`］。ステータス配列 `info`（64 バイト）は、両方のターゲット領域の最後にデバイスから取得されます［`MAP(FROM:info)`、`tgt → hst`］。

© 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。

不連続メモリー間のデータの移動には時間とエネルギーがかかるため、ハイライトされている移動（合計 12,290,048,128 バイト）に細心の注意を払う必要があります。これを念頭に、初期の実装を改善する方法を見てみましょう。

前号の LU 因数分解の記事では、1 つの OpenMP* ターゲット領域で十分な場合に 2 つのターゲット領域を使用することの短所を説明しました。まず、制御フローをターゲットデバイスに転送すると、オーバーヘッドが発生します。初期の実装では 2 回発生します（**リスト 2**）。次に、冗長なホストとデバイス間のデータ転送が必要です（**図 2**）。線形システムを解くときに必要なのは、入力配列と RHS をデバイスにコピーして、解をデバイスから取得することです。ステータス配列も取得する必要がありますが、これらは比較的小さいものです。ピボットはデバイスでのみ使用されるため、`ipiv` 配列をデバイスメモリーに直接割り当てることにします [`MAP(ALLOC:ipiv(1:stride_ipiv, 1:batch_size))`]。2 つの OpenMP* ターゲット領域を融合すると、クリーンで簡潔なコードになり（**リスト 3**）、データの移動が少なくて済みます（**図 3**）。

```
!$OMP TARGET DATA MAP(TO:a) MAP(TOFROM: b)                    &
                  MAP(ALLOC:ipiv(1:stride_ipiv, 1:batch_size)) &
                  MAP(FROM:info_rf, info_rs)

    !$OMP DISPATCH
    CALL dgetrf_batch_strided(n, n, a, lda, stride_a, &
                              ipiv, stride_ipiv, batch_size, info_rf)

    !$OMP DISPATCH
    CALL dgetrs_batch_strided('N', n, nrhs, a, lda, stride_a, &
                               ipiv, stride_ipiv, &
                               b, ldb, stride_b, batch_size, info_rs)
!$OMP END TARGET DATA

IF (ANY(info_rf .ne. 0)) THEN
    PRINT *, 'Error: getrf_batch_strided returned with errors.'
ELSEIF (ANY(info_rs .ne. 0)) THEN
    PRINT *, 'Error: getrs_batch_strided returned with errors.'
ELSE
    PRINT *, 'Computation executed successfully'
ENDIF
```

リスト 3. 1 つの OpenMP* ターゲット領域を使用して oneMKL でバッチ処理された線形システムを解く。OpenMP* ディレクティブは青で表示しています。LAPACK の呼び出しは緑色で表示しています。

© 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。

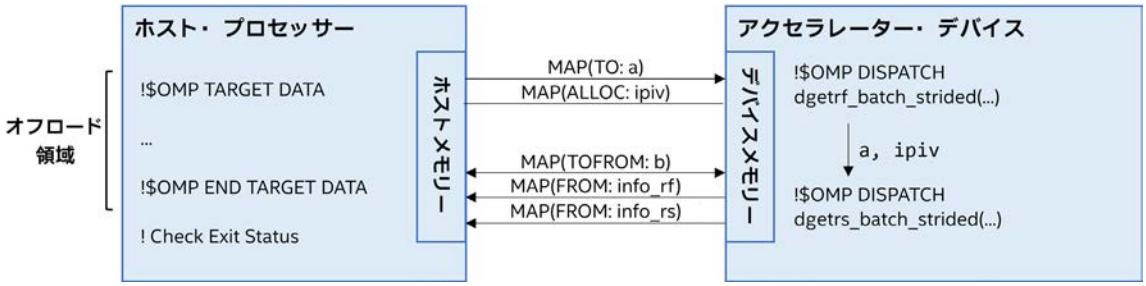

図 3. リスト 3 の OpenMP* ターゲットオフロード領域におけるホストとデバイス間のデータ転送。各矢印（アクセラレーター・デバイス内を除く）は、ホストとデバイスのメモリー間のデータ転送を示します。

この効果はデバッグ情報で確認できます。再度、OpenMP* MAP 構文を使用して明示的に転送される配列をハイライトして示します。

```
Libomptarget --> Moving 88 bytes (hst:0x00007ffe443ba9c8) -> (tgt:0x0000000001e55008)
Libomptarget --> Moving 64 bytes (hst:0x00007ffe443bad68) -> (tgt:0x0000000001e55088)
Libomptarget --> Moving 64 bytes (hst:0x00007ffe443bad18) -> (tgt:0x0000000001e55108)
Libomptarget --> Moving 512000 bytes (hst:0x00007f39ada86240) -> (tgt:0x00007f399a06d000)
Libomptarget --> Moving 88 bytes (hst:0x00007ffe443bae38) -> (tgt:0x0000000001e55188)
Libomptarget --> Moving 4096000000 bytes (hst:0x00007f3913fff200) -> (tgt:0x00007f3819edd000)
Libomptarget --> Moving 88 bytes (hst:0x00007ffe443bae98) -> (tgt:0x0000000001e55208)
Libomptarget --> Moving 512000 bytes (tgt:0x00007f399a06d000) -> (hst:0x00007f39ada86240)
Libomptarget --> Moving 64 bytes (tgt:0x0000000001e67040) -> (hst:0x00007f39add5fd80)
Libomptarget --> Moving 64 bytes (tgt:0x0000000001e67000) -> (hst:0x00007f39add5fdc0)
```

ドープベクトルを無視すると、**リスト 3** の 1 つの OpenMP* ターゲット領域では、ホストとデバイス間のデータ転送（4,097,024,128 バイト）が初期の実装の 2 つのターゲット領域の転送（12,290,048,128 バイト）（**リスト 2**）よりも大幅に少なくなります。

ホストとデバイス間のデータ転送を最小化することは、ヘテロジニアス・コンピューティングにおける最優先事項です。行列サイズが大きくなるほど、2 つ目のサンプルのコード（**リスト 3**）は、オリジナルのコード（**リスト 2**）よりもパフォーマンスが大幅に向上します（**表 1**）。

© 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。

行列サイズ	CPU 時間	GPU 時間 (サンプル 1)	GPU 時間 (サンプル 2)
1,000x1,000	0.13	1.72	1.70
2,000x2,000	0.60	0.84	0.72
4,000x4,000	3.90	2.51	2.10
8,000x8,000	19.15	5.98	4.55
12,000x12,000	63.11	12.55	9.51
16,000x16,000	139.45	21.33	15.71

表 1. 2 x 第 4 世代インテル® Xeon® Platinum 8480+ プロセッサー（2.0GHz）（CPU）、インテル® データセンター GPU マックス・シリーズ 1550（GPU）、528GB メモリーを搭載した Linux*（Ubuntu* 20.04 x64、5.15.47 カーネル）システムでさまざまな行列サイズのバッチ処理された線形システムを解くのにかかった時間（秒）。GPU テストは 1 つのタイルのみ使用。各線形システムの右辺は 1 つ。各実験は 5 回実行。oneMKL 関数の JIT コンパイルのオーバーヘッドが含まれるため最初の実行は破棄。レポートされている時間は残りの 4 つの実行の合計。

この記事で使用したサンプルプログラムは、このリポジトリー（英語）から入手できます。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）では、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。

OpenMP* ターゲット構文には、アクセラレーターのオフロードをプログラマーが細かく制御できるようにする、多くのオプションが用意されています。OpenMP* 仕様（英語）とサンプルコード（英語）、および 2 つのチュートリアル「OpenMP* API を使用した GPU プログラミングの概要」（英語）と「データ転送とメモリー割り当てを最小化する」（英語）を確認してみてください。

© 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。

SYCL* によるデバイス検出

システムに搭載されているアクセラレーターを知る

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
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

デバイス検出は、SYCL* やクロスアーキテクチャーのヘテロジニアス並列プログラミング・アプローチの重要な一面です。以前の oneAPI の記事では、SYCL* と oneMKL および oneDPL ライブラリーを使用して計算をアクセラレーター・デバイスにオフロードすること（つまり、コードを実行する場所の制御）に焦点を当てました。ヘテロジニアス・システム向けの移植可能なコードを記述するには利用可能なハードウェアに関する情報をシステムに照会する機能が必要になるため、この記事ではデバイス検出に焦点を当てています。例えば、GPU を使用するように SYCL* デバイスセレクターを指定しても、システムに GPU がない場合、次の文は失敗します。

```
...
    sycl::queue Q(sycl::gpu_selector_v);
...
terminate called after throwing an instance of 'sycl::_V1::runtime_error'
  what():  No device of requested type available. Please check https://software.intel.com/
content/www/us/en/develop/articles/intel-oneapi-dpcpp-system-requirements.html -1 (PI_ERROR_
DEVICE_NOT_FOUND)
Aborted
```

© 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。

このコードは、GPU がないシステムには移植できません。GPU セレクターの代わりにデフォルトセレクターを使用して SYCL* キューをインスタンス化すれば動作することは保証されますが、キューがどこにワークを送信するか制御できなくなります。SYCL* ランタイムがデバイスを選択します。

```
...
    sycl::queue Q(sycl::default_selector_v);

    std::cout << "Running on: "
              << Q.get_device().get_info<sycl::info::device::name>()
              << std::endl;
...
Running on: Intel(R) Xeon(R) Gold 6128 CPU @ 3.40GHz
```

確実なヘテロジニアス並列プログラムを作成するため、SYCL* デバイス検出を詳しく見て、次の質問に答えてみましょう。

- どのアクセラレーター・デバイスを使用できますか？
- SYCL* キューはどのデバイスを使用していますか？
- oneDPL 実行ポリシーはどのデバイスを使用していますか？

堅牢なデバイス検出

さまざまなインテル® ハードウェアのオプションがあり、最新のインテル® oneAPI ソフトウェアがすでにインストールされているインテル® DevCloud for oneAPI（英語）で、いくつかの例を実行してみましょう。ハードウェアは定期的に更新されますが、この記事の執筆時点（2022 年 12 月 2 日）では次の計算ノードが利用可能でした。

```
$ pbsnodes | grep properties | sort | uniq -c | sort -nr

     79      properties = xeon,skl,gold6128,ram192gb,net1gbe,jupyter,batch
     78      properties = xeon,cfl,e-2176g,ram64gb,net1gbe,gpu,gen9
     26      properties = xeon,skl,gold6128,ram192gb,net1gbe,jupyter,batch,fpga_compile
     25      properties = core,tgl,i9-11900kb,ram32gb,netgbe,gpu,gen11
     12      properties = xeon,skl,ram384gb,net1gbe,renderkit
     12      properties = xeon,skl,gold6128,ram192gb,net1gbe,fpga_runtime,fpga,arria10
      6      properties = xeon,icx,gold6348,ramgb,netgbe,jupyter,batch
      4      properties = xeon,icx,plat8380,ram2tb,net1gbe,batch
      4      properties = xeon,clx,ram192gb,net1gbe,batch,extended,fpga,stratix10,fpga_runtime
```

ご覧のように、さまざまな CPU、GPU、および FPGA オプションがあります（機密保持契約ユーザーは、インテル® DevCloud for oneAPI の NDA パーティション内のリリース前のハードウェアにアクセスできます）。ノードをリクエストして、利用可能なデバイスを確認します。

```
$ qsub -I -l nodes=1:gen11:ppn=2
```

このコマンドは、第 11 世代インテル® プロセッサー・グラフィックスを含む単一ノードへの対話型アクセスをリクエストします。SYCL* では、すでに説明した 2 つのセレクターに加えて、ビルトインセレクターとして `default_selector_v`、`gpu_selector_v`、`cpu_selector_v`、および `accelerator_selector_v` を提供しています。インテルは、FPGA 開発向けに、`fpga_selector` および `fpga_emulator_selector` の 2 つの拡張機能も提供しています。これらの拡張機能は `sycl/ext/intel/fpga_device_selector.hpp` ヘッダーに含まれています。FPGA 上での SYCL* の使用に関する詳細は、『インテル® oneAPI プログラミング・ガイド』の「FPGA フロー」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。

ビルトインセレクターは主に利便性を目的としていますが、例外処理と組み合わせると堅牢になります。

```
sycl::device d;
try {
    d = sycl::device(sycl::gpu_selector_v);
}
catch (sycl::exception const &e) {
    d = sycl::device(sycl::cpu_selector_v);
}
```

しかし、まだ SYCL* ランタイムがデバイスを選択しています。特に複数のデバイスが利用可能な場合は、さらなる制御が必要になります。

次のプログラムは、計算ノードで利用可能なプラットフォームとデバイスをリストします（この情報は、`sycl-ls` コマンドライン・ユーティリティーを使用して取得することもできます）。

```
#include <sycl/sycl.hpp>

int main()
{
    for (auto platform : sycl::platform::get_platforms())
    {
        std::cout << "Platform: "
                  << platform.get_info<sycl::info::platform::name>()
                  << std::endl;

        for (auto device : platform.get_devices())
        {
            std::cout << "\tDevice: "
                      << device.get_info<sycl::info::device::name>()
                      << std::endl;
        }
    }
}

$ icpx -fsycl show_platforms.cpp -o show_platforms
$ ./show_platforms

Platform: Intel(R) FPGA Emulation Platform for OpenCL(TM)
        Device: Intel(R) FPGA Emulation Device
Platform: Intel(R) OpenCL
        Device: 11th Gen Intel(R) Core(TM) i9-11900KB @ 3.30GHz
Platform: Intel(R) OpenCL HD Graphics
        Device: Intel(R) UHD Graphics [0x9a60]
Platform: Intel(R) Level-Zero
        Device: Intel(R) UHD Graphics [0x9a60]
```

SYCL* プラットフォームは、ホストがアクセラレーター・デバイスに接続されている OpenCL* プラットフォーム・モデルに基づいています。これは、上記の出力例で明白です。このシステムには OpenCL* プラットフォームと oneAPI レベルゼロ・プラットフォームがあります。各プラットフォームには、SYCL* プログラムがワークを送信できるデバイスがあります。OpenCL* または oneAPI レベルゼロ・バックエンドを使用するかどうかに応じて、2 つの GPU プラットフォームがあります。CPU および FPGA エミュレーション・プラットフォームもあります。この情報を利用して、これらのデバイスのいずれかにワークを送信するキューを作成できます。

© 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。

```
#include <sycl/sycl.hpp>

int main()
{
    auto platforms = sycl::platform::get_platforms();

    sycl::queue Q1(platforms[1].get_devices()[0]);
    sycl::queue Q2(platforms[3].get_devices()[0]);

    std::cout << "Q1 mapped to "
              << Q1.get_device().get_info<sycl::info::device::name>()
              << std::endl;

    std::cout << "Q2 mapped to "
              << Q2.get_device().get_info<sycl::info::device::name>()
              << std::endl;
}
$ ipcx -fsycl map_queues.cpp -o map_queues
$ ./map_queues

Q1 mapped to 11th Gen Intel(R) Core(TM) i9-11900KB @ 3.30GHz
Q2 mapped to Intel(R) UHD Graphics [0x9a60]
```

前の例では明示的にデバイスが指定されているため、各自のシステムでこのプログラムを試す場合は、必ずプラットフォーム・インデックスを更新してください。

SYCL* キューとデバイスの照会

前のサンプルコードではキューの作成を確認できますが、常に確認できるとは限りません。例えば、SYCL* キューは通常、oneAPI ライブラリー関数に渡されます。そのため、場合によっては、次のような情報をキューに照会する必要があります。

- キューがマップされているターゲットデバイスは何か？
- デバイスのバックエンド API は何か？
- それはインオーダー・キューか（つまり、カーネルを送信した順序で実行する必要があるか）？
- ターゲットデバイスのベクトル幅または最大ワークアイテム数はいくつか？

ライブラリー開発者は、照会した情報を使用して最適なコードパスを選択できます。そのため、SYCL* `queue` クラスでは、`get_backend()`、`get_context()`、`get_device()`、`is_in_order()` などの情報を照会するメンバー関数を提供しています。同様に、SYCL* device クラスでは、`is_cpu()`、`is_gpu()`、`get_info()` などのデバイス特性を照会するメンバー関数を提供しています。特に、`get_info()` 関数は、ベンダー、ベクトル幅、最大ワークアイテム数とイメージサイズ、メモリー特性など、ターゲットデバイスに関する詳細な情報を収集するために使用できます。SYCL* 2020 仕様には、照会できるデバイス情報記述子とデバイスアスペクトの完全なリストが含まれています。

これらの情報を使用したコードの最適化はここでは取り上げませんが、将来の記事の候補の 1 つです。

© 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。

カスタムセレクター

これまで見てきたデバイスセレクターは、SYCL* 実装により提供されるビルトインセレクターでした。これらのデバイスセレクターは、デバイスを受け入れてスコアを返す C++ の呼び出し可能オブジェクトとして実装されます。SYCL* 実装は、デバイスセレクターを呼び出してシステム内で利用可能なすべてのデバイスにスコアを付け、最終的にスコアが最も高いデバイスを選択します。

同じ形式の呼び出し可能オブジェクトを作成することで、独自のカスタム・デバイス・セレクターを作成できます。簡単な最初の例として、すべての CPU デバイスに正のスコアを付け、他のすべてのデバイスに負のスコアを付けることで、ビルトイン CPU セレクターと同じ動作を行うデバイスセレクターを作成します。

```
...
    auto my_cpu_selector = [](const sycl::device& d)
    {
        if (d.is_cpu())
        {
            return 1;
        }
        else
        {
            return -1;
        }
    };
    sycl::queue Q(my_cpu_selector);
...
Running on: Intel(R) Xeon(R) Gold 6128 CPU @ 3.40GHz
```

デバイスセレクターは単なる関数であるため、デバイスの任意のプロパティー（アスペクトやデバイス情報記述子など）とプログラムのほかの変数（コマンドライン引数など）を組み合わせて、デバイスのスコアを付けることができます。デバイスのスコア付けと選択を完全に制御できるため、選択したデバイスがアプリケーションの要件を満たしていることを確認できます。以下の例は、倍精度浮動小数点演算をサポートしていないデバイスを無視し、ブール変数を使用して GPU を優先するように構成するデバイスセレクターを示しています。

```
...
    bool prefer_gpus = true;  // コマンドラインまたは設定ファイルから
    auto my_selector = [=](const sycl::device& d)
    {
        // 倍精度をサポートしないデバイスを無視
        if (not d.has(sycl::aspect::fp64))
        {
            return -1;
        }

        // オプションで GPU を選択
        if (prefer_gpus and d.is_gpu())
        {
            return 1;
        }
        else
        {
            return 0;
        }
    };
    sycl::queue Q(my_selector);
...
```

© 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。

プログラマーの要件を満たすデバイスを選択できるように支援する aspect_selector 関数が最近 SYCL* に追加されました。例えば、次の文は、エミュレートされた固定機能デバイスを除外して、半精度をサポートする GPU デバイスを選択します。

```
auto dev = sycl::device{sycl::aspect_selector(
  std::vector{sycl::aspect::fp16, sycl::aspect::gpu},        // 対象のアスペクト
  std::vector{sycl::aspect::custom, sycl::aspect::emulated}  // 対象でないアスペクト
)};
```

この記事の執筆時点では aspect_selector はまだインテル® oneAPI DPC++/C++ コンパイラーでサポートされていませんが、間もなく利用可能になるはずです [**翻訳者注：**バージョン 2023.1 では aspect_selector はサポートされていますが、sycl::aspect::emulated は未実装です]。

oneDPL の実行ポリシーの変更

The Parallel Universe 48 号の記事「oneAPI の maxloc リダクション」では、oneDPL の実行ポリシーを使用して関数をアクセラレーターにオフロードする方法を紹介しました。コード例では oneapi::dpl::execution::dpcpp_default ポリシーを単純に使用しています。SYCL* キューを使用して実行ポリシーを変更し、oneDPL 関数を実行する場所を明示的に制御する方法を見てみましょう。

```
#include <oneapi/dpl/execution>

int main()
{
    sycl::queue Q1(sycl::gpu_selector_v);
    auto gpu_policy = oneapi::dpl::execution::make_device_policy(Q1);

    std::cout << "GPU execution policy runs oneDPL functions on "
              << gpu_policy.queue().get_device().
                                    get_info<sycl::info::device::name>()
              << std::endl;

    sycl::queue Q2(sycl::cpu_selector_v);
    auto cpu_policy = oneapi::dpl::execution::make_device_policy(Q2);

    std::cout << "CPU execution policy runs oneDPL functions on "
              << cpu_policy.queue().get_device().
                                    get_info<sycl::info::device::name>()
              << std::endl;
}
$ icpx -fsycl onedpl_policy_example.cpp -o onedpl_example
$ ./onedpl_example

GPU policy runs oneDPL functions on Intel(R) UHD Graphics [0x9a60]
CPU policy runs oneDPL functions on 11th Gen Intel(R) Core(TM) i9-11900KB @ 3.30GHz
```

以前のプログラムは、ビルトイン CPU セレクターとビルトイン GPU セレクターを使用してキューを作成し、これらのキューを使用して oneDPL 実行ポリシーを設定しています。前に示したように、プラットフォームとデバイスを照会して、キューをインスタンス化することもできます。

© 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。

```
...
    auto platforms = sycl::platform::get_platforms();

    sycl::queue Q1(platforms[3].get_devices()[0]);
    auto gpu_policy = oneapi::dpl::execution::make_device_policy(Q1);

    sycl::queue Q2(platforms[1].get_devices()[0]);
    auto cpu_policy = oneapi::dpl::execution::make_device_policy(Q2);
...
```

繰り返しますが、前の例ではデバイスが直接指定されているため、必ずシステムのプラットフォーム・インデックスを更新してください。

この記事では、SYCL* が提供するデバイス検出向けの機能と、プログラムでプラットフォームとデバイス情報を使用する方法の概要を説明しました。マルチデバイス・システムの普及とともに、プログラマーが特定のデバイス向けのアルゴリズムをターゲットにする機会は増えています。この続きは、今後の The Parallel Universe で詳しく説明する予定です。

関連情報

- インテル® デベロッパー・クラウド（英語）– さまざまなインテルのハードウェアで oneAPI をテストするために必要なものがすべてまとめられており、無料で利用できます。
- SYCL* 2020 リファレンス・ガイド（PDF）– SYCL* 2020 仕様が要約されたリファレンスです。この記事では、デバイスの選択、プラットフォーム・クラス、コンテキスト・クラス、およびデバイスクラスの説明を参照しました。
- 『Data Parallel C++ : Mastering DPC++ for Programming Heterogeneous Systems using C++ and SYCL』（英語）の「第 12 章 : デバイス情報」– 一部のコード例の構文は古くなっていますが、デバイス検出の概要がよく分かります。
- oneAPI-samples（英語）リポジトリーには、SYCL* と oneAPI ライブラリーを使用したプログラミングを説明する数多くのサンプルコードがあります。
- 『インテル® oneAPI プログラミング・ガイド』は、インテルの oneAPI ツールの基本的な概要を示します。
- 『oneAPI GPU 最適化ガイド』は、oneAPI を使用して最高のパフォーマンスを達成するためのコーディングに関するアドバイスを含むドキュメントです。

© 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。

レベルゼロとは

オープンなバックエンド・アプローチでどこでも計算を実現

Robert Mueller-Albrecht インテル コーポレーション プロダクト・マーケティング・エンジニア

この記事では、レベルゼロ（英語）のアプリケーション・インターフェイス、目的、およびビジョンの概要を説明します。レベルゼロの基本的なアーキテクチャーと、計算ユニットリソースへの低レベルのアクセス制御の利点を紹介します。レベルゼロは、OpenMP*（英語）や SYCL*（英語）のような言語拡張とともに使用できます。レベルゼロと SYCL* C++ 言語拡張抽象化レイヤーの間の相互作用がアプリケーション・ソフトウェア開発者にどのように明らかにされるか簡単に説明します。

レベルゼロは、任意の数のオフロードデバイスへのアクセスを設定および管理する低レベル API として設計されています。プログラムフローに干渉することなく、C++ 標準規格に準拠したヘテロジニアス計算を行うことができる抽象化レイヤーも提供します。この機能により、異なるランタイム環境の間でコードを移植できます。レベルゼロ API バックエンドを考慮することにより、SYCL* や OpenMP* 言語拡張の抽象化を超えて、制御のレベルを向上できます。

© 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。

レベルゼロ・インターフェイス（英語）は、oneAPI 仕様（英語）の一部です。CPU、GPU、およびアクセラレーターへのベアメタルアクセスにより、oneAPI の API ベースのプログラミング・モデルとダイレクト・プログラミング・モデルを補完します。インテル® oneAPI ベース・ツールキットの一部としてインテル® GPU をターゲットにしたインテルのリファレンス実装（英語）とインテル® oneAPI DPC++/C++ コンパイラーでの使用法も文書化されています。この記事を読めば、レベルゼロランタイムの使用を開始したり、独自のランタイムを検討するために必要なことが分かるでしょう。

ヘテロジニアス・コンピューティングの能力を引き出す

レベルゼロのインテルの最初の実装はインテル® GPU をターゲットにしていますが、レベルゼロのビジョンと可能性は非常に広範であり、特定のデバイス要件に合わせて調整された抽象化を作成することが可能です。関数ポインター、メモリー、I/O など、幅広い言語機能をサポートするように適合させることができます。API は、CPU、GPU、フィールド・プログラマブル・ゲート・アレイ（FPGA）、その他のアクセラレーター・アーキテクチャーを含む、さまざまな計算デバイスで動作するように設計されています。

レベルゼロは、OpenCL* や Vulkan* などのほかの低レベル API と共存できます。ただし、高レベルの oneAPI および SYCL* 開発者の経験がハードウェアに依存せず、可能な限りアーキテクチャーに依存しないで柔軟になるように、独立して進化することを意図しています。SYCL* を利用する高レベルのランタイム API とライブラリーで必要となる明示的な制御も提供します。レベルゼロは完全なオープンソースで、その仕様（英語）、ソース・リポジトリー（英語）、およびインテル® GPU 計算ランタイムの実装（英語）は、GitHub* で公開されています。

つまり、レベルゼロは、ヘテロジニアス・コンピューティングを広く開放し、オフロード計算の選択を実現する、柔軟なオープン・バックエンドを提供します。システムレベルのインターフェイスを明示的に制御する、次のような機能を提供します。

- デバイス検出
- メモリー割り当て
- ピアツーピア通信
- プロセス間共有
- カーネルのサブミット
- 非同期実行とスケジューリング
- 同期プリミティブ
- メトリックレポート
- システム管理

レベルゼロの主要な概念から始めましょう。

© 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。

レベルゼロの基本

レベルゼロは、アプリケーション層の下に位置していて、C++ アプリケーションとターゲット・デバイス・プロパティーの間の抽象化インターフェイスとして採用できます（**図 1**）。CPU やその他の計算デバイスが含まれることがあります。開発者は共有デバイスのリソースとシームレスにやり取りして、レベルゼロドライバーがサポートする特定のデバイスにワークロードをディスパッチできます。ドライバーは、サポートされているデバイスを利用可能なデバイスリストに追加し、任意の SYCL* キューをそのデバイスにマップしてワークを送信できます。特定のデバイス・プロパティーや複数のレベルゼロデバイス間で共有されるリソースにアクセスする必要がない場合、それらのデバイスは異なる SYCL* バックエンド API を使用するほかのデバイスと同じように動作します。

図 1. oneAPI バックエンド・アーキテクチャーのコンテキストのレベルゼロ・インターフェイス

レベルゼロの真の強みは、デバイス固有のメモリー共有または同期コンテキスト・オブジェクトに対する低レベルの制御とサポートです。デバイスの透明性が向上するだけでなく、レベルゼロ API にヘテロジニアス・オフロード・ターゲット・デバイスの構成可能性が追加されます。

レベルゼロデバイスの検出と選択の一連の流れを次に示します。

レベルゼロローダー

オフロードデバイスまたはアクセラレーターへのアクセスは、レベルゼロローダーから開始します。システムのデバイス向けのレベルゼロドライバーを検出します。ローダー・プロジェクトには、レベルゼロ実装をビルドして操作できるようにするレベルゼロヘッダーとライブラリー（英語）も含まれています。ドライバーの初期化と検出のコードサンプルを次に示します。

```
// ドライバーを初期化
zeInit(0);

// ドライバー・インスタンスをすべて検出
uint32_t driverCount = 0;
zeDriverGet(&driverCount, nullptr);

ze_driver_handle_t* allDrivers = allocate(driverCount *
                                          sizeof(ze_driver_handle_t));
zeDriverGet(&driverCount, allDrivers);

// GPU デバイスを含むドライバー・インスタンスを検索
ze_driver_handle_t hDriver = nullptr;
ze_device_handle_t hDevice = nullptr;
for(i = 0; i < driverCount; ++i)
{
    uint32_t deviceCount = 0;
    zeDeviceGet(allDrivers[i], &deviceCount, nullptr);
```

© 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。

```
        ze_device_handle_t* allDevices = allocate(deviceCount *
                                                  sizeof(ze_device_handle_t));
        zeDeviceGet(allDrivers[i], &deviceCount, allDevices);

        for(d = 0; d < deviceCount; ++d)
        {
            ze_device_properties_t device_properties {};
            device_properties.stype = ZE_STRUCTURE_TYPE_DEVICE_PROPERTIES;
            zeDeviceGetProperties(allDevices[d], &device_properties);

            if(ZE_DEVICE_TYPE_GPU == device_properties.type)
            {
                hDriver = allDrivers[i];
                hDevice = allDevices[d];
                break;
            }
        }

        free(allDevices);
        if(nullptr != hDriver)
        {
            break;
        }
    }

    free(allDrivers);
    if(nullptr == hDevice)
        return; // GPU デバイスなし
```

続いて、メモリー、コマンドキュー、モジュール、同期オブジェクトなどを管理するコンテキスト・オブジェクトを作成します。複数のデバイスで共有できるシステムリソースを管理する場合、コンテキストの使用は特に重要になります。共有メモリーのシナリオの簡単な例を次に示します。

```
    // コンテキストを作成
    zeContextCreate(hDriver, &ctxtDesc, &hContextA);
    zeContextCreate(hDriver, &ctxtDesc, &hContextB);

    zeMemAllocHost(hContextA, &desc, 80, 0, &ptrA);
    zeMemAllocHost(hContextB, &desc, 88, 0, &ptrB);
```

© 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。

スケジューリング・モデル

図 2 に示すように、すべてのコマンドがスケジュールされ、レベルゼロデバイスにディスパッチされます。

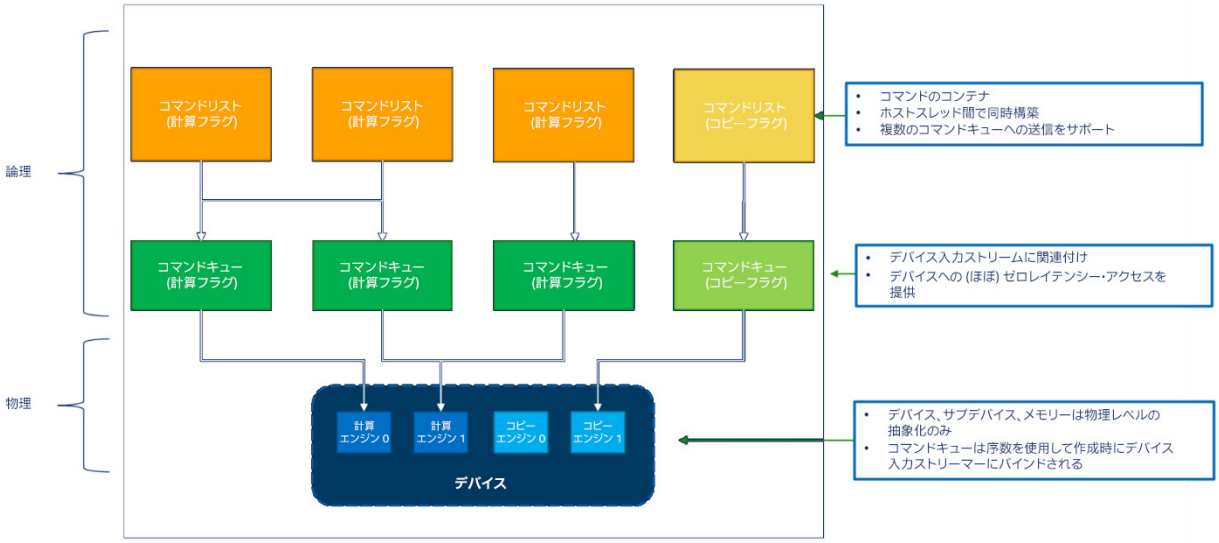

図 2. レベルゼロ・スケジューリング・モデル

コマンドは、オフロード計算ユニットまたはアクセラレーターで実行される一連のコマンドを表すコマンドリストに追加されます。コマンドリストはリセットして再利用できます。リストを再度作成する必要はありません。同じ一連のコマンドの送信に再利用できます。コマンドを再度追加する必要はありません。

コマンドリストは、実行のためコマンドキューに送信されます。キューはデバイスの物理入力ストリームに関連付けられた論理オブジェクトで、同期または非同期として設定でき、キューグループに構成できます。このスケジューリング・モデルは、**図 3** に示すソースコードのフローに変換されます。

図 3. レベルゼロ・スケジューリング実行フロー

© 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。

即時コマンドリスト

コマンドリストの処理を最適化して、レイテンシーの管理に役立てることができます。保証された応答時間が必要な優先度の高いタスクは、低レイテンシーの即時コマンドリストを使用して処理できます。これは、非常に低レイテンシーの送信使用モデル専用の、特別なタイプのコマンドリストです。

コマンドリストとその暗黙的なコマンドキューは、コマンドキュー記述子を使用して作成します。即時コマンドリストに追加したコマンドは、デバイス上でただちに実行されます。即時コマンドリストに追加したコマンドは、コマンドが完了するまでブロックされ同期的に実行される場合があります。即時コマンドリストは、完了後に閉じたりリセットする必要はありません。使用法は尊重され、想定された動作に従います。次の疑似コードは、即時コマンドリストの作成と使用の基本的な手順を示しています。

```
// 即時コマンドリストを作成
ze_command_queue_desc_t commandQueueDesc = {
    ZE_STRUCTURE_TYPE_COMMAND_QUEUE_DESC,
    nullptr,
    computeQueueGroupOrdinal,
    0, // インデックス
    0, // フラグ
    ZE_COMMAND_QUEUE_MODE_DEFAULT,
    ZE_COMMAND_QUEUE_PRIORITY_NORMAL
};
ze_command_list_handle_t hCommandList;
zeCommandListCreateImmediate(hContext, hDevice, &commandQueueDesc,
                                                &hCommandList);

// カーネルをデバイスにただちにサブミット
zeCommandListAppendLaunchKernel(hCommandList, hKernel, &launchArgs, nullptr, 0, nullptr);
…
```

産業用システムのエラー状態を判断して直ちに対処する必要がある閉フィードバック・ループのユースケースが豊富にあると想像してみてください。即時コマンドリストの概念により、保証された応答時間が必要なユースケースに GPU オフロード計算が追加されます。

アプリケーション開発者にとっての利点

レベルゼロの主要な設計原則の説明に続いて、アプリケーション開発者が SYCL* ベースのアプリケーションからレベルゼロと対話する方法を見てみましょう。

デバイスの選択

レベルゼロドライバーが実装されたデバイスはすべて、アプリケーション開発者が初期化して使用できます。インテルのレベルゼロ実装は、GitHub*（英語）で公開されており、ほかのデバイスのリファレンスとして使用して、oneAPI クロスアーキテクチャーのヘテロジニアス計算サポートを活用できます。

© 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。

特定のオフロードデバイスを選択するには、SYCL_DEVICE_FILTER 環境変数［**翻訳者注：**2023.1 ではこの環境変数は非推奨となり、ONEAPI_DEVICE_SELECTOR に置き換えられています］を使用します。この環境変数は、すべてのデバイスクエリー機能とデバイスセレクターに影響します。実行中のシステムで SYCL* を使用するデバイスが利用可能かどうかを確認するには、sycl-ls コマンドを使用します。

```
$ sycl-ls
[opencl:acc:0] Intel(R) FPGA Emulation Platform for OpenCL(TM), Intel(R) FPGA Emulation Device 1.2
[2022.13.3.0.16_160000]
[opencl:cpu:1] Intel(R) OpenCL, 11th Gen Intel(R) Core(TM) i7-1185G7 @ 3.00GHz 3.0
[2022.13.3.0.16_160000]
[opencl:gpu:2] Intel(R) OpenCL HD Graphics, Intel(R) Iris(R) Xe Graphics 3.0 [31.0.101.3358]
[ext_oneapi_level_zero:gpu:0] Intel(R) Level-Zero, Intel(R) Iris(R) Xe Graphics 1.3 [1.3.23828]
[host:host:0] SYCL host platform, SYCL host device 1.2 [1.2]
```

デバイスのプロパティーに関して、次のように考えることができます。ドライバー・オブジェクトは、システムの物理デバイスのコレクションを表します。複数のドライバーを利用できる場合があります。例えば、あるドライバーがあるベンダーのアクセラレーターをサポートし、別のドライバーが別のベンダーのアクセラレーターをサポートするようなケースです。コンテキスト・オブジェクトは、デバイスまたはシステムリソースを表します。その主な目的は、複数のデバイスで使用される可能性があるリソースの作成と管理です。デバイス・オブジェクトは、システムの物理デバイスを表します。デバイス検出 API は、システムのデバイスを列挙するために使用されます。zeDeviceGet() 関数は、ドライバーがサポートするレベルゼロデバイスの数を照会し、読み取り専用のグローバル構造であるデバイス・オブジェクトを取得するために使用されます。すべてのデバイスには、16 バイトの汎用一意識別子（UUID）が割り当てられます。デバイスハンドルは、デバイス固有のリソースの作成および管理中に使用されます。

レベルゼロの追加機能を利用するには、まず SYCL* 対応の C++ コードとレベルゼロ API 間の相互運用性を有効にする必要があります。C++ アプリケーション内からレベルゼロのデバイス固有のコンテキスト・オブジェクトを直接操作する場合は、ヘッダーファイルをソースコードに次の順序で含める必要があります。

```
#include “level_zero/ze_api.h”
#include “sycl/backend/level_zero.hpp”
```

レベルゼロ・バックエンドは、sycl::backend 列挙子に追加されます。

```
enum class backend {
  // ...
  ext_oneapi_level_zero,
  // ...
};
```

この方法で、SYCL* 名前空間から sycl:get-native API を使用して、SYCL* オブジェクトの基礎となるレベルゼロのデータ構造をリクエストできます。

```
template <backend BackendName, class SyclObjectT>
auto get_native(const SyclObjectT &Obj)
    -> backend_return_t<BackendName, SyclObjectT>
  ext_oneapi_level_zero,
```

© 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。

詳細は、インテル® oneAPI レベルゼロ・バックエンド仕様（英語）を参照してください。指定された SYCL* キューは、システムの利用可能なデバイスに接続されます。

```
try {
    vector<device> sub_devices = ...;
    for (auto &d : sub_devices)
    {
      // 各キューは自身のコンテキストにあり、データを共有しない
      auto q = queue(d);
      q.submit([&](handler& cgh) {...});
    }
}
```

その後、デバイスが使用され、ほかの SYCL* デバイスと同様に実行キューが割り当てられます。興味深いのは、デバイス固有のリソース、またはホストとオフロード実行デバイス間で共有されるリソースにアクセスする場合です。デバイス固有のリソースにアクセスする例として、統合共有メモリー（USM）を見てみましょう。以下の概要は、インテルの USM 実装の方法を反映していますが、ほかのレベルゼロ・ライブラリーにも同じ機能を実装できます。

統合共有メモリー

メモリーは、CPU と GPU の両方をカバーする単一の仮想アドレス空間を持つ統合メモリーとして上位レベルのソフトウェア・スタックに表示されます。リニアな、フォーマットされていないメモリー割り当ては、ホスト・アプリケーションでポインターとして表されます。ホストのポインターは、デバイスのポインターと同じサイズです。SYCL* 名前空間を使用してメモリーを割り当てる方法は 3 つあります。

1. `sycl::malloc_device`
   - 割り当ては、指定されたデバイスのみアクセスでき、ホストまたはコンテキストのほかのデバイスはアクセスできません。
   - データは常にデバイス上に保持され、カーネル実行に利用できる最速のデータです。
   - ホストまたはコンテキストのほかのデバイスにデータを転送するには、明示的なコピーが必要です。
2. `sycl::malloc_host`
   - 割り当ては、ホストとコンテキストのほかのデバイスからアクセスできます。
   - データは常にホスト上にあり、デバイスから PCI 経由でアクセスします。
   - データをホストまたはデバイスと同期するために明示的なコピーは必要ありません。
3. `sycl::malloc_shared`
   - 割り当ては、ホストと指定されたデバイスのみアクセスできます。
   - ホストとデバイス間でデータを移行（レベルゼロドライバーで操作）して、アクセスを高速化できます。
   - ホストとデバイス間の同期には明示的なコピーは必要ありませんが、コンテキストのほかのデバイスには必要です。

© 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。

この方法で呼び出される同等の低レベルのレベルゼロ呼び出しはそれぞれ、`zeMemAllocDevice`、`zeMemAllocHost`、および `zeMemAlloc_shared` です。実装の詳細は、レベルゼロコア API 仕様（英語）を参照してください。

USM は、レベルゼロが提供する高度なデバイス認識の一例です。詳細は、レベルゼロ・コア・プログラミング・ガイド（英語）を参照してください。

まとめと次のステップ

レベルゼロは、計算ユニットとアクセラレーターでワークをスケジュールし、メモリーを管理するための豊富なインターフェイスのセットを提供します。プログラムのロードと実行、メモリーの割り当て、およびヘテロジニアス・ワークロードの管理に必要な、すべてのサービスを提供します。特定のハードウェア構成に合わせてカスタマイズできるオープンなインターフェイスを使用しながら、それほど特殊ではないセットアップでワークロードを実行するために必要な抽象化と柔軟性を維持します。

コマンドキューやコマンドリストなど、レベルゼロで定義されたオブジェクトを使用すると、基盤となるハードウェアを低レベルで制御できます。これらの利用可能な最適化手法を使用すると、高レベルのプログラミング言語とアプリケーションは、ハードウェアに近いレイテンシーでワークロードを実行して、パフォーマンスを向上できます。

SYCL* と組み合わせると、C++ を使用してシームレスに使用およびアクセスできます。さらに、Python*、Julia*、および Java* での経験を活用して、さまざまな言語でレベルゼロの優れた言語ランタイムサポートを提供しています。オープンソース・コミュニティーと業界全体がレベルゼロへの貢献度を高めることにより、レベルゼロがマルチアーキテクチャーの選択における汎用的で強力なインターフェイスとなることを期待しています。

関連情報（英語）

- oneAPI レベルゼロおよび OpenCL* ドライバー向けインテル® グラフィックス計算ランタイム
- インテル® oneAPI DPC++/C++ コンパイラー・デベロッパー・ガイドおよびリファレンス : インテル® oneAPI レベルゼロ
- oneAPI レベルゼロ仕様
- oneAPI レベルゼロのヘッダー、ローダー、および検証レイヤー

© 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。