インテルの新しい組込み AI アクセラレーション・エンジンを使用する

インテル® AMX とインテル® XMX で AI の効率、スケーラビリティー、パフォーマンスを向上

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

人工知能（AI）が注目されるとともに、パフォーマンスを向上させる方法として、低精度データ型の導入と、これらのデータ型のハードウェア・サポートが求められています。低精度モデルは計算が速く、メモリー・フットプリントが小さくなることから、AI トレーニングと推論には、32 ビットのデータ型よりも低精度のデータ型が推奨されます。これらの低精度データ型を最適化およびサポートするには、ハードウェアに特別な機能と命令が必要です。インテルは、これらをインテル® CPU とインテル® GPU でそれぞれ、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）およびインテル® Xe マトリクス・エクステンション（インテル® XMX）として提供しています。最も使用されている 16 ビット形式は、16 ビット IEEE 浮動小数点数（fp16）、bfloat16、16 ビット整数（int16）であり、最も使用されている 8 ビット形式は、8 ビット整数 (int8) と 8 ビット Microsoft* 浮動小数点数 (ms-fp8) です。これらの形式の違いの一部を**図 1** に示します。

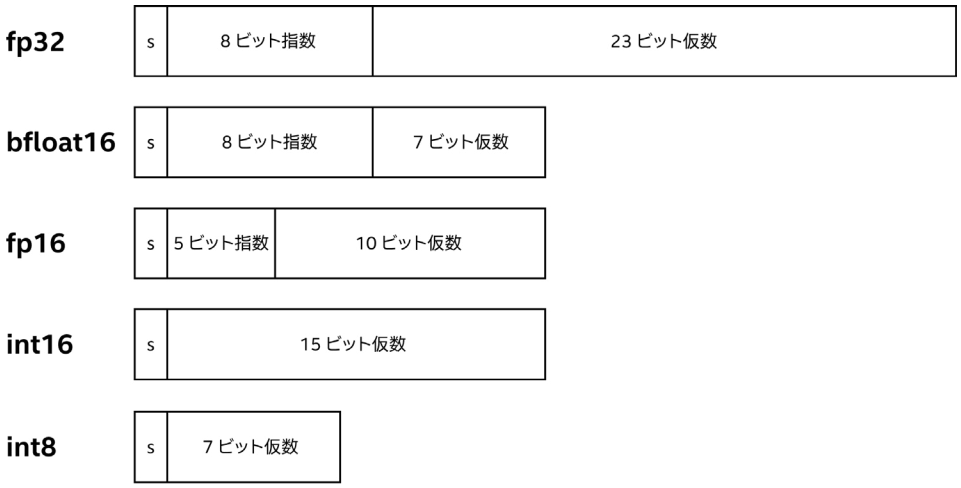


図 1. IEEE 標準データ型のさまざまな数値表現。s は符号付きビット（正の数は 0、負の数は 1）を表し、exp は指数を表します。

インテル® AMX とインテル® XMX を呼び出すプログラミング・パラダイムは、ハードウェアにより異なります。この記事では、それらをプログラムするさまざまな方法を紹介した後、これらの命令セットのパフォーマンスの利点を説明します。

インテル® AMX とインテル® XMX

インテル® AMX

インテル® AMX は、マイクロプロセッサー向けの x86 命令セット・アーキテクチャー（ISA）を拡張したものです。アクセラレーターが操作を実行できる、タイルと呼ばれる 2D レジスターを使用します。インテル® AMX の 1 つのユニットには、通常 8 つのタイルがあります。タイルは、ロード、ストア、クリア、または内積操作を実行できます。インテル® AMX は、int8 および bfloat16 データ型をサポートします。インテル® AMX は、第 4 世代インテル® Xeon® スケーラブル・プロセッサーでサポートされています（**図 2**）。

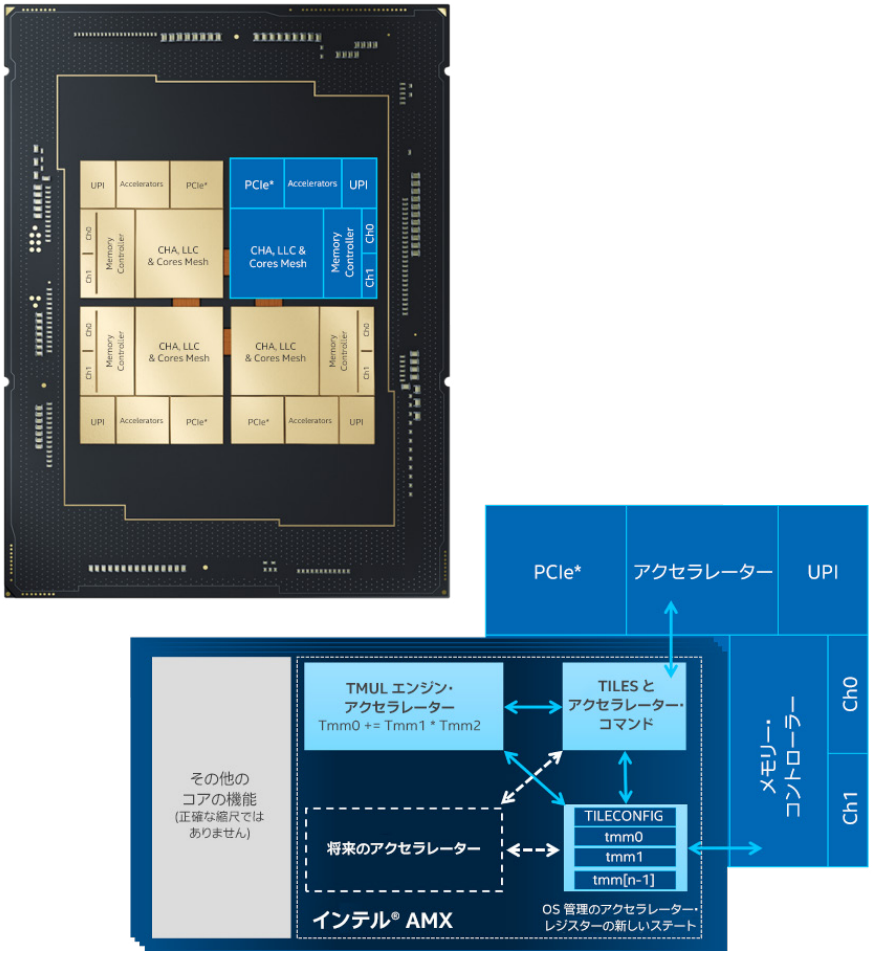


図 2. 第 4 世代インテル® Xeon® スケーラブル・プロセッサーのインテル® AMX

インテル® XMX

インテル® XMX（内積累算シストリック（Dot Product Accumulate Systolic、DPAS）とも呼ばれる）は、2D シストリック・アレイでの内積と累算命令の実行を専門としています。並列コンピューター・アーキテクチャーのシストリック・アレイは、密結合なデータ処理ユニットのホモジニアス・ネットワークです。各ユニットは、上流のユニットから受け取ったデータの関数として部分的な結果を計算し、その結果を保存して、下流のユニットに渡します。インテル® XMX は、ハードウェア世代に応じて、int8、fp16、bfloat16、tf32 などのさまざまなデータ型をサポートしています。インテル® XMX は、インテル® データセンター GPU マックス・シリーズまたはインテル® データセンター GPU フレックス・シリーズの一部です。インテル® データセンター GPU マックス・シリーズのインテル® X^e HPC 2 スタックは、**図 3** では X^e として省略されています。スタックとは、タイルの代わりに使用される用語です。インテル® データセンター GPU マックス・シリーズは 8 つのインテル® X^e スライスで構成されます。各スライスには 16 個のインテル® X^e コアが含まれています。各コアには 8 つのベクトルエンジンと 8 つの行列エンジンが含まれています。

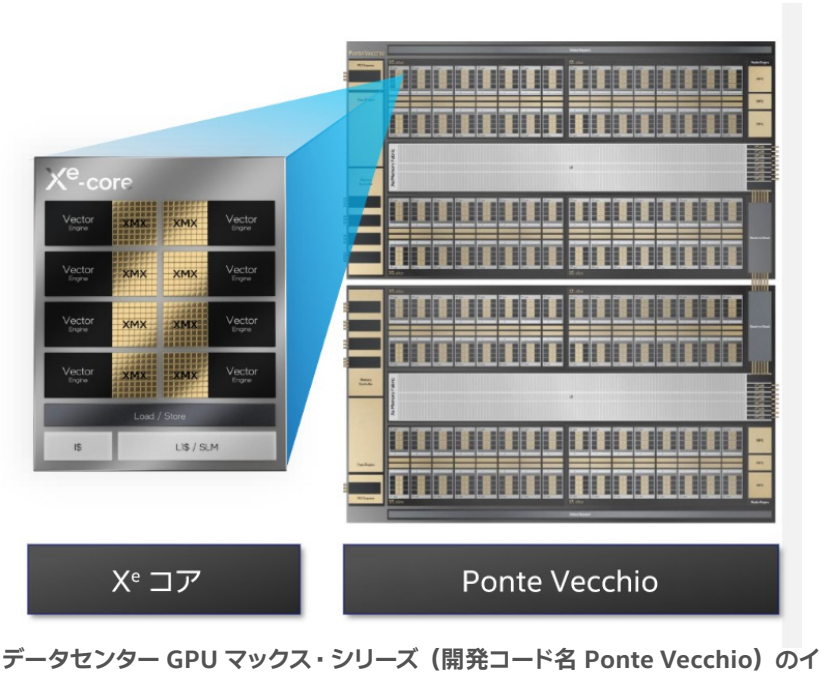


図 3. インテル® データセンター GPU マックス・シリーズ（開発コード名 Ponte Vecchio）のインテル® XMX

インテル® AMX とインテル® XMX を使用したプログラミング

ユーザーは、ディープラーニング・フレームワーク、専用ライブラリー、カスタム SYCL* カーネルから低水準の組込み関数まで、さまざまなレベルでインテル® XMX を操作できます。**図 4** は、さまざまなコーディング抽象化でインテル® AMX とインテル® XMX を呼び出す方法（後述）を示しています。これらの拡張機能を利用するには、インテル® oneAPI ベース・ツールキット 2024.0 が必要です

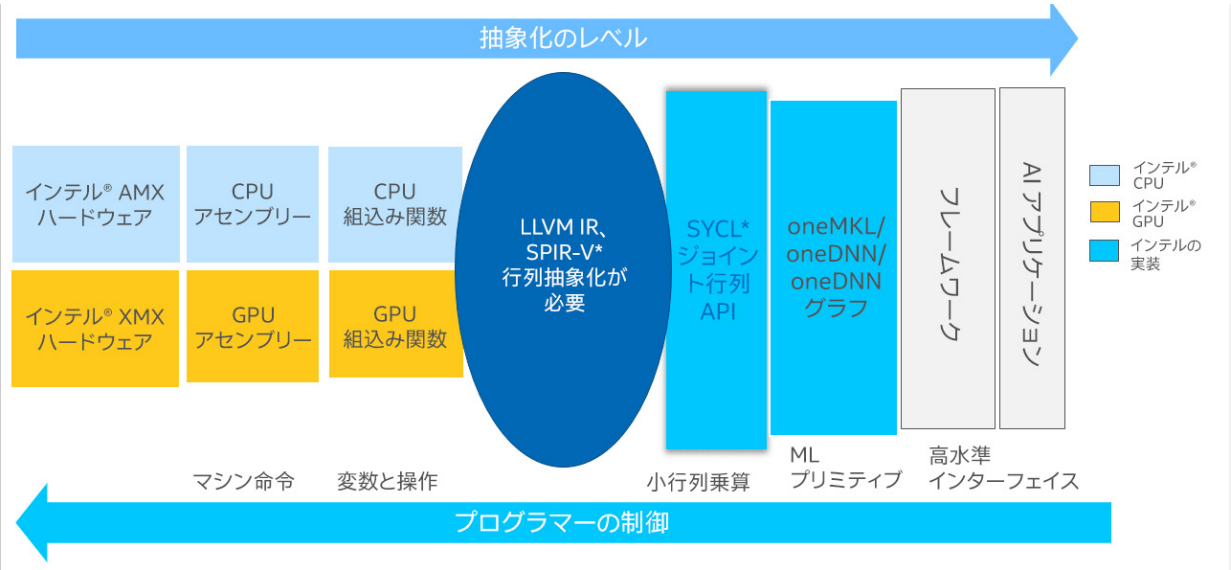


図 4. 行列計算のプログラミングの抽象化

SYCL* ジョイント行列拡張機能

ジョイント行列は、CPU 向けのインテル® AMX、GPU 向けのインテル® XMX、NVIDIA* Tensor コアのようなテンソル・ハードウェア・プログラミングのターゲットを統合する新しい SYCL* 拡張機能です。これらの SYCL* ジョイント行列の例（英語）は、この拡張機能の使用法を示しています。独自のニューラル・ネットワーク・アプリケーションを構築したいユーザーは、TensorFlow* などのフレームワークや、インテル® oneAPI ディープ・ニューラル・ネットワーク・ライブラリー（インテル® oneDNN）やインテル® oneAPI マス・カーネル・ライブラリー（インテル® oneMKL）などのライブラリーよりも抽象度が低いジョイント行列を使用できます。ジョイント行列は、ポータブルなだけでなく、異なるハードウェアで利用可能な最大のパフォーマンスを活用できる統合インターフェイスを提供します。ジョイント行列は、パフォーマンス、生産性、融合機能と、異なるテンソル・ハードウェアへの移植性を提供するため、ハードウェア・プラットフォームごとに異なるコードベースを維持する必要がなくなります。

「ジョイント行列」という用語は、行列がワーク項目のグループ内で共有され、各ワーク項目専用ではないことを強調しています（**図 5**）。グループスコープは SYCL* ジョイント行列構文の追加テンプレート・パラメーターとして追加されますが、現在の実装ではサブグループ・スコープのみサポートされています。行列に対する一般的な操作である、ロード、ストア、「乗算と加算」操作を実行するために必要な関数の一部は、それぞれ、`joint_matrix_load`、`joint_matrix_store`、`joint_matrix_mad` で表されます。それらは、プログラムを実行しているハードウェアに応じて、低水準のインテル® AMX またはインテル® XMX 組込み関数を呼び出します。`joint_matrix_load` 関数はメモリーから 2D タイル / レジスターにデータをロードし、`joint_matrix_store` は 2D タイルからアキュムレーター行列のデータをメモリーに格納し、`joint_matrix_mad` は 2 つの行列に乗算を実行し、結果を 3 番目の行列と累算して結果を返します。

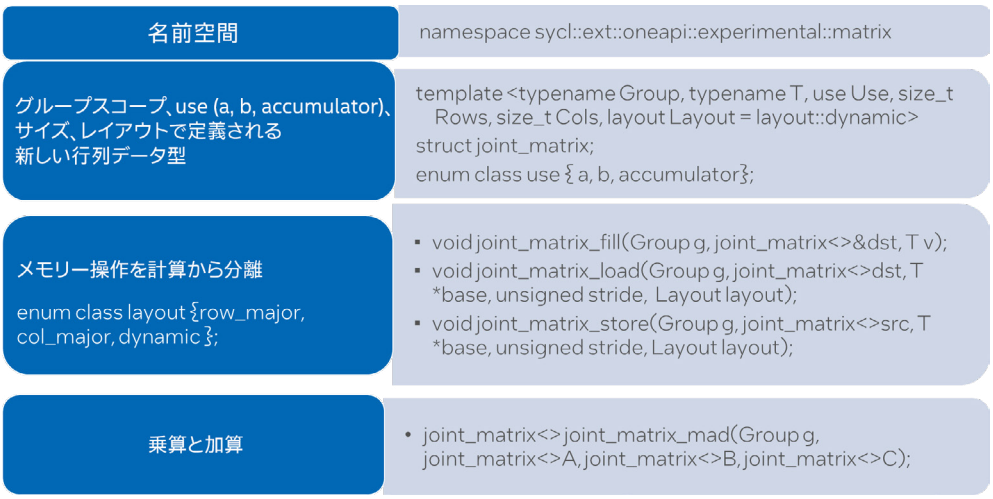


図 5. 一般的な SYCL* ジョイント行列 API

インテル® oneDNN とインテル® oneMKL

インテル® oneAPI ディープ・ニューラル・ネットワーク・ライブラリー（インテル® oneDNN）とインテル® oneAPI マス・カーネル・ライブラリー（インテル® oneMKL）は、インテル® CPU（第 4 世代インテル® Xeon® スケーラブル・プロセッサー以降）とインテル® データセンター GPU マックス・シリーズでそれぞれ、デフォルトでインテル® AMX とインテル® XMX を使用します。インテル® oneDNN ユーザーは、コードでインテル® XMX でサポートされているデータ型を使用していることを確認し、GPU サポートを有効にしてインテル® oneDNN ライブラリーをビルドする必要があります。

インテル® oneDNN にバンドルされている matmul_perf（英語）サンプルは、適切にコンパイルされるとインテル® AMX およびインテル® XMX を呼び出します。追加のインテル® oneDNN のサンプルは、こちら（英語）から入手できます。単純なインテル® oneMKL のサンプルは、こちら（英語）から入手できます。

インテル® AMX のパフォーマンス解析

標準の fp32 と bfloat16 で実行した行列乗算を比較してみましょう。使用するベンチマーク（英語）は、本質的には、GEMM を繰り返し呼び出す単純なラッパーで、次のオプションを渡すことができます。

- GEMM カーネルのインテル® マス・カーネル・ライブラリー（インテル® MKL）バージョン
- 単精度または倍精度の GEMM（SGEMM/DGEMM）
- 使用するスレッドの数（NUM_THREADS）
- 乗算する行列のサイズ（SIZE_N）

ベンチマーク・コードは、問題のサイズを取得し、C = A x B 行列のスペースを割り当て、A 行列と B 行列をランダムデータで初期化し、GEMM を 1 回呼び出して初期化し、設定した回数 GEMM を連続して呼び出し、後者の実行時間を測定します。bfloat16 のほうが fp32 よりも優れていることが分かります（**図 6**）。

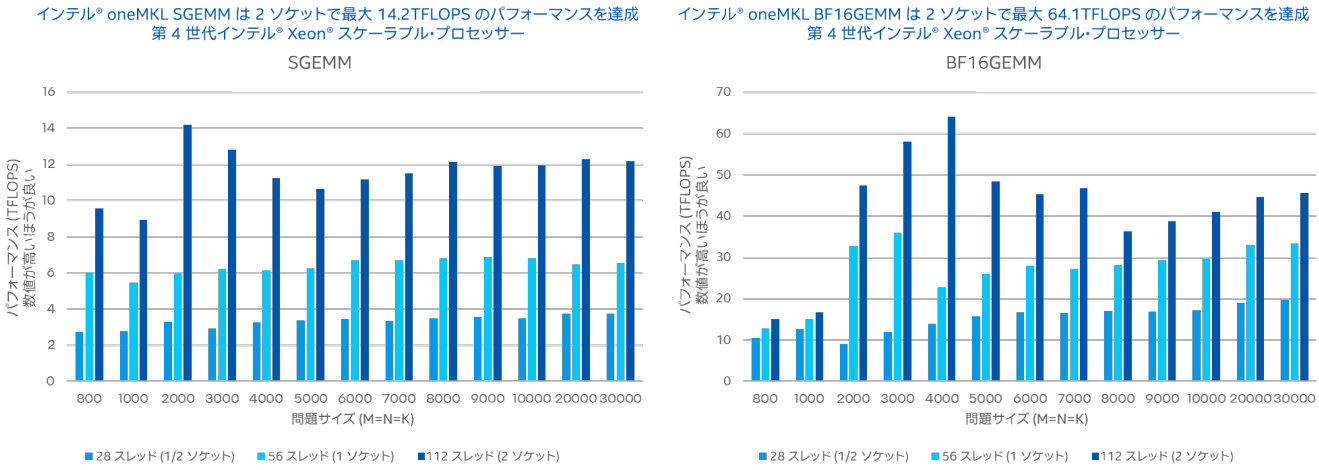


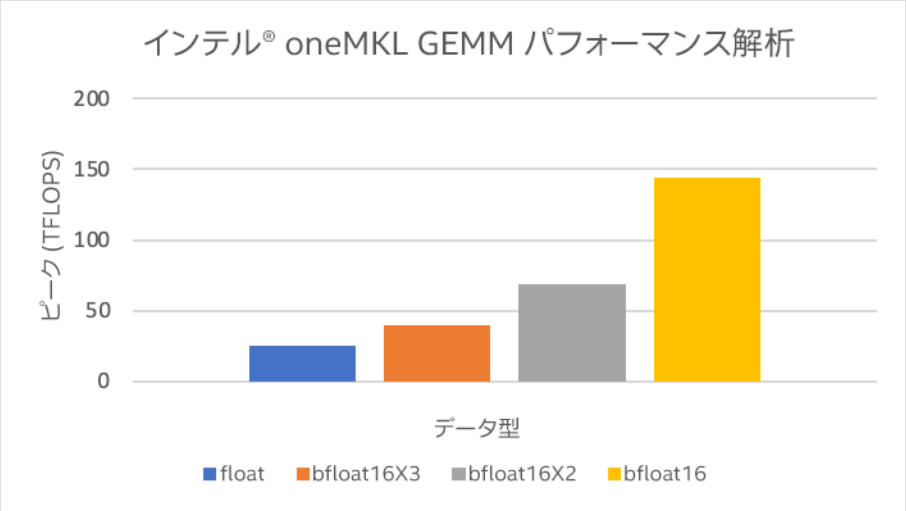
図 6. fp32 と bfloat16 データ型を使用した行列乗算パフォーマンスの比較

テスト：パフォーマンス結果は 2024.0 で検証しました。2023.x リリースと比較してパフォーマンスの低下は観察されませんでした。
システム構成とワークロードのセットアップ：1 ノード、2x インテル® Xeon® Platinum 8480+ プロセッサー、Denali Pass プラットフォーム、合計 DDR5 メモリー 1024GB（16 スロット/64GB/4800）、ucode 0x2b000161、インテル® ハイパースレッディング・テクノロジー無効、インテル® ターボ・ブースト・テクノロジー有効、Ubuntu* 22.04.1 LTS、5.17.0-051700-generic、1x インテル® SSD 3.5TB（OS ドライブ）、インテル® oneMKL 2023.0。正方行列（ 次元 800 から 30,000）の SGEMM および BFLOAT16GEMM のパフォーマンス。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。性能は、使用状況、構成、その他の要因によって異なります。詳細については、http://www.intel.com/PerformanceIndex/（英語）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。
法務上の注意書き
性能は、使用状況、構成、その他の要因によって異なります。詳細については、http://www.intel.com/PerformanceIndex/（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。構成の詳細は、補足資料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。実際の費用と結果は異なる場合があります。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。

システム構成：1x インテル® データセンター GPU マックス 1550、2 ソケットのインテル® Xeon® 8480+ プロセッサーでホスト、512GB DDR5-4800 メモリー、Ubuntu* 22.04、カーネル 5.15、IFWI 2023WW28、インテル® oneMKL 2023.1、ICX 2023.1。

インテル® XMX のパフォーマンス解析

次に、この GEMM ベンチマーク（英語）を使用して、インテル® GPU 上のインテル® XMX のパフォーマンスの利点を調査します。インテル® oneMKL は、インテル® XMX を使用して単精度 GEMM を高速化するいくつかのアルゴリズムをサポートしています。bfloat16x2 と bfloat16x3 は、bfloat16 システリック・ハードウェアを使用して単精度 GEMM を近似する 2 つのアルゴリズムです。内部的には、単精度入力データが bfloat16 に変換され、システリック配列と乗算されます。3 つのバージョン（bfloat16、bfloat16x2、bfloat16x3）を使用して、精度とパフォーマンスの間のトレードオフを選択できます。bfloat16 が最も高速で、bfloat16x3 が最も正確です（標準 GEMM と同様）。3 つのバージョンはすべて、標準 float を上回ります（**図 7**）。



システム構成： 1x インテル® データセンター GPU マックス 1550、2 ソケットのインテル® Xeon® 8480+ プロセッサーでホスト、512GB DDR5-4800 メモリー、Ubuntu* 22.04、カーネル 5.15、IFWI 2023WW28、インテル® oneMKL 2023.1、ICX 2024.0。https://github.com/oneapi-src/oneAPI-samples/tree/master/Libraries/oneMKL/matrix_mul_mkl（英語）のコードを使用。

法務上の注意書き

性能は、使用状況、構成、その他の要因によって異なります。詳細については、http://www.intel.com/PerformanceIndex/（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。構成の詳細は、補足資料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。実際の費用と結果は異なる場合があります。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。

まとめ

この記事では、最新のインテル® CPU とインテル® GPU に導入されたインテル® AMX とインテル® XMX 命令セットの概要を説明しました。インテル® AMX とインテル® XMX 命令セットは、コンパイラー組込み関数から SYCL* ジョイント行列抽象化、インテル® oneMKL とインテル® oneDNN まで、異なるレベルのプログラミングで呼び出すことができます。コーディング・パラダイムの抽象化レベルが高くなるほど、これらの命令セットの呼び出しが容易になり、呼び出しに対するプログラマーの制御が少なくなります。インテル® AMX とインテル® XMX はパフォーマンス向上のためのテクノロジーです。第 4 世代インテル® Xeon® スケーラブル・プロセッサーの GEMM ベンチマークの例では、適切な低精度データ型を使用してインテル® AMX を呼び出すだけで、パフォーマンスがどのように向上するか明確に理解できます。インテル® データセンター GPU マックス 1550 の GEMM ベンチマークの例では、適切な低精度データ型を使用してインテル® XMX を呼び出すだけで、パフォーマンスがどのように向上するか明確に理解できます。パフォーマンスの利点、使いやすさ、および低水準プログラミング、ライブラリーとフレームワークでプログラムできる機能のため、ユーザーは AI ワークロードでこれらの拡張機能をテストして使用することを推奨します。