統合ジョイント行列の SYCL 拡張

この記事は、インテルのウェブサイトで公開されている「IXPUG Take-Out: Unified Joint Matrix SYCL* Extension」の日本語参考訳です。原文は更新される可能性があります。原文と翻訳文の内容が異なる場合は原文を優先してください。

モダン・コンピューティングの主要分野は、計算オブジェクトの特性を効率良く表現し操作するため、行列とテンソルに大きく依存しています。

これは特に、シミュレーション、高度な数値微分方程式、ビジュアル・コンピューティング、CGI VFX、金融リスク分析、ゲーム、そして AI の分野において顕著です。つまり、大規模なソフトウェア・アプリケーションで影響を受けていないものはほとんどありません。近年 GPU などのアクセラレーターが重要な役割を果たしているのは、それらが行列処理用ハードウェアとして設計されていたためであり、コンピューター・グラフィックスの分野では当初から恩恵をもたらしてきました。

SPIR-V 協調行列拡張と互換性のある行列およびテンソル演算用の統合インターフェイスが存在し、それが SYCL の一部として提供され、さまざまなハードウェアやヘテロジニアス・コンピューティング環境で利用可能であったらどうでしょうか?

これを、SYCL コンピュート・エコシステム全体に簡単に調整および最適化できる、頻繁の高い演算を対象とした高性能カーネル群と組み合わせることで、行列ハードウェア・プログラミング向けの共通の統一された抽象化に近づいています。この行列用の統合 API は、さまざまなライブラリーやユーティリティーで使用できます。

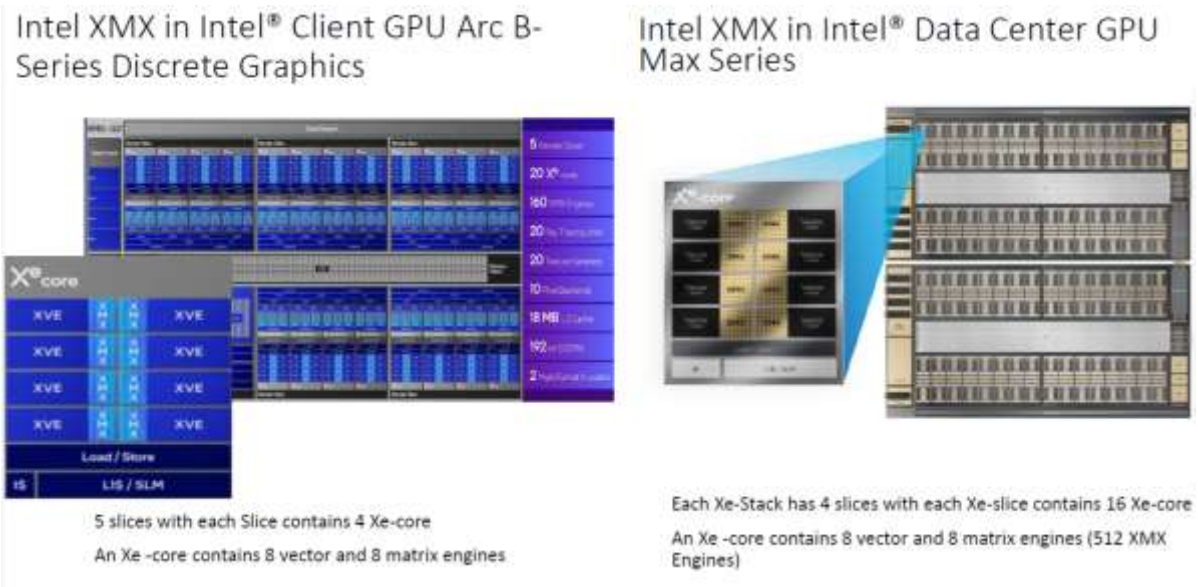

図 1: GPU におけるタイルとスライスの構成

ここでは、Dounia Khaldi (英語) による洞察に富んだ講演 (以下のリンクからアクセス可能) の主題について簡単に紹介します。

PyTorch や TensorFlow などのフレームワーク、またはインテル® oneAPI ディープ・ニューラル・ネットワーク (インテル® oneDNN) などのライブラリーは、行列ハードウェアによるアクセラレーションを効率良く活用するように設計されていますが、最新のアクセラレーションがまだ組み込まれていなかったり、既製のアプローチでは汎用すぎたり、オーバーヘッドが大きすぎる場合に、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と思うことがあります。

このような場合、より低レベルの API 抽象化が、カスタム・ワークロード固有の最適化を記述するのに役立ちます。ジョイント行列は、カスタム・ソリューション向けにこの低レベルの抽象化を提供し、パフォーマンス、生産性、融合機能、そして単一コードベースで異なる行列ハードウェアをターゲットにすることを可能にします。

「ジョイント行列: 行列ハードウェア・プログラミング向けの統合 SYCL 拡張」(英語)
⇒ IXPUG 2025 の講演ビデオ (英語)
⇒ 講演資料 (英語)
講演者: インテル コーポレーション Dounia Khald (英語)

行列ハードウェアの例

行列ハードウェアの例として、インテル® X^e マトリックス・エクステンション (インテル® XMX) が挙げられます。これは、クライアント・アプリケーションとデータセンター・アプリケーションの両方のインテル® GPU に追加されたシストリック・アレイです (図 1)。これは、同一の密結合プロセシング要素 (PE) がグリッド状に相互接続された並列処理アーキテクチャーで構成されています。各 PE は特定の計算を実行し、その結果を隣接する PE に渡すことで、行列全体に周期的なデータフローを生み出します。

インテル® Arc™ B シリーズ・ディスクリート・グラフィックスには 160 基のインテル® XMX エンジンが搭載されており、インテル® データセンター GPU マックス・シリーズには 512 基のインテル® XMX エンジンが搭載されています。これらのエンジンは、bfloat16、half、tf32、int8、int4 などのデータ型を用いた行列の乗算カーネルと加算カーネルを高速に処理するように設計されています。

同様に、現在のインテル® Xeon® スケーラブル・プロセッサーとインテル® Xeon® 6 プロセッサーには、行列計算カーネルを高速化する 2 次元レジスターを備えたインテル® アドバンスト・マトリクス・エクステンション (インテル® AMX) が搭載されています。

SYCL ジョイント行列拡張 API

試験的な SYCL 名前空間 sycl::ext::oneAPI::experimental::matrix は、ジョイント行列演算で使用される新しいデータ型を定義します。

```
using namespace sycl::ext::intel::experimental::matrix;
template <typename Group, typename T, use Use, size_t Rows,
          size_t Cols, layout Layout = layout::dynamic>
struct joint_matrix;
enum class use { a, b, accumulator};
enum class layout {row_major, col_major, dynamic};
```

メモリー操作

一般的なメモリーのロード、ストア、ポインター操作は、SYCL ワークグループ全体に適用でき、データをプリフェッチして効率良く実行でき、ポインターはオーバーロード可能です。

```
void joint_matrix_mad(Group g, joint_matrix<>&D,
     joint_matrix<>&A, joint_matrix<>&B, joint_matrix<>&C);
void joint_matrix_apply(Group g, joint_matrix<>&jm, F&& func);
void joint_matrix_apply(Group g,
     joint_matrix<>& jm0, joint_matrix<>& jm1, F&& func);
void joint_matrix_copy(Group g, joint_matrix<Group, T1, Use1,
     Rows, Cols, Layout1> &dest,
joint_matrix<Group, T2, Use2, Rows, Cols, Layout2> &src);
```

計算操作

行列の乗算と加算 (D=A*B+C) は、要素ごとの演算、活性化関数、(逆) 量子化、データ型変換を含む行列のコピーなど、基本的な操作をすべてサポートしています。

```
void joint_matrix_fill(Group g, joint_matrix<>&dst, T v);
void joint_matrix_load(Group g, joint_matrix<>&dst,
     multi_ptr<> src, size_t stride, Layout layout);
void joint_matrix_load(Group g, joint_matrix<>&dst,
     multi_ptr<> src, size_t stride);
void joint_matrix_store(Group g, joint_matrix<>src,
     multi_ptr<> dst, unsigned stride, Layout layout);
void joint_matrix_prefetch(Group g, T* ptr, size_t
     stride, layout Layout, Properties properties);
```

サンプルコード

結果として得られるジョイント行列の乗算と加算のコードシーケンスは、基本的な GEMM 実装であり、SYCL がサポートする多くのハードウェア・プラットフォーム (インテル、NVIDIA、AMD、その他のベンダー) で実行できます。異なるアーキテクチャーにおけるパフォーマンスの主な考慮事項は、ハードウェア、ワークグループ、およびサブグループのタイルサイズです。

```
using namespace sycl::ext::oneapi::experimental::matrix;
queue q;
range<2> G = {M / MSG /*32*/, (N / NSG /*64*/) * SG_SIZE};
range<2> L = {MWG / MSG /*32*/, NWG / NSG /*64*/ * SG_SIZE};
…
q.submit([&](sycl::handler& cgh) {
  auto pA = address_space_cast<…>(memA);
  auto pB = address_space_cast<…>(memB);
  auto pC = address_space_cast<…>(memC);
  cgh.parallel_for(nd_range<2>(G, L), [=](nd_item<2> item) {
    joint_matrix<…32,16…> A;
    joint_matrix<…16,64,…> B;
    joint_matrix<…32,64,…> C;
    joint_matrix_fill(sg, subC, 0);
    for (unsigned int kwg = 0; kwg < K; kwg += KWG /*32*/) {
      for (unsigned int ksg = 0; ksg < KWG /*16*/;
        ksg+=KSG/*16*/) {
        joint_matrix_load(sg, A, pA + offsetA, K);
```

```
        joint_matrix_load(sg, B, pB + offsetB, N);
        joint_matrix_mad(sg, C, A, B, C);
      }
    }
    joint_matrix_store(sg, C, pC + offsetC, N, layout);
  });
});q.wait;
```

図 2 : データ・ワークグループ (kwg) とサブグループ (sg) のサイズ

最適化の考慮事項

ブロックサイズ

行列演算の最適化は、処理対象となるデータセットを、用途と使用するハードウェアに最適なテンソル形式にすることから始まります。

これは、アクセス速度の階層構造で考えることができます。つまり、ローカルキャッシュ、レジスター、または特定の行列アクセラレーター・カーネル内にどれだけのデータを配置すべきか、そして、各実行カーネルの論理ワークグループ・サイズとサブグループ・サイズの観点からも考えることができます。

図 2 のコード例では、以下のレイアウトを前提としてサイズを設定しています。

カーネルタイル	M×N×K	4096×4096×4096
ワークグループ・タイル	MWG×NWG×K WG	256×256×32
サブグループ・タイル	MSG×NSG×KSG	32×64×16
ジョイント行列	MJM×NJM×KJM	8×16×16 または 32×64×16

プリフェッチとキャッシュ制御

パフォーマンスをさらに最適化するため、乗算するプリフェッチ行列 A と B を見てみましょう。

```
// Before k loop
constexpr size_t prefDistance = 3;
for (int p = 0; p < prefDistance; p++)
     joint_matrix_prefetch<8, 32>(sg, A + offset, K,
        layout::row_major,
        syclex::properties{syclex::prefetch_hint_L1});
// After joint_matrix_mad (before next load)
joint_matrix_prefetch<8, 32>(
     sg, A + prefetch_offsetA, K, layout::row_major,
     syclex::properties{syclex::prefetch_hint_L1});
```

実行フローとタイミングに有利な場合は、注釈付きポインターを使用してキャッシュを完全にバイパスし、行列 A を順番にロードすることもできます。

```
auto pA = syclex::annotated_ptr{A, syclex::properties{
     syclintelex::read_hint<syclintelex::cache_control<
     syclintelex::cache_mode::uncached,
     syclex::cache_level::L1,syclex::cache_level::L3>>}};
joint_matrix_load(sg, A, pA + offsetA, K);
```

joint_matrix 操作のパフォーマンスをファインチューニングする状況と方法は数多くあります。

マルチアーキテクチャーの行列乗算を効率化

SYCL ジョイント行列拡張のサポートは、最新バージョン 2025.x のインテル® oneAPI DPC++/C++ コンパイラーとインテル® DPC++ 互換性ツールで利用可能です。

最新の実験的なジョイント行列カーネルのソースコードは、Dounia Khaldi の GitHub (英語) で確認できます。

ぜひお試しいただき、CUDA から SYCL へのコード移行、カスタム AI、またはエッジ AI プロジェクトに活用してください。

関連情報

- SYCL ジョイント行列拡張を利用したインテル® XMX のプログラミング (英語)
- SYCL oneAPI ジョイント行列拡張の GitHub (英語)
- IXPUG: ジョイント行列: 行列ハードウェア・プログラミング向け統合 SYCL 拡張の動画 (英語) およびスライド (英語)
- 2025 年 IXPUG 年次カンファレンス (英語)

製品および性能に関する情報

[1] 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。