Joint Matrix: A Unified SYCL Extension for Matrix Hardware Programming

Dounia Khaldi

Intel Corp.

IXPUG Annual Conference 2025 hosted by TACC

Apr. 15th, 2025

Introduction

- Programming abstractions for matrix computing: tradeoff between increasing level of abstraction and programmer control
- Related Work: SYCL ESIMD, Triton, CUTLASS, oneDNN, CUDA wmma, etc.
- Deliver unified SYCL matrix interface across matrix hardware: Intel AMX, Intel XMX, Nvidia Tensor Cores, AMD Matrix Cores
- Status
  - Unified interface is part of oneAPI releases (starting oneAPI 2023.1 release)
  - Use of SPIRV Cooperative Matrix extension
  - Comparable performance to oneDNN but more flexibility and control on kernel fusions and matrix hardware
  - Full transition to SPIRV cooperative matrix in oneAPI 2025.1 release
  - Performance kernels with special tuning (tiling factors, matrix size, prefetch) for different devices https://github.com/dkhaldi/sycl_joint_matrix_kernels/

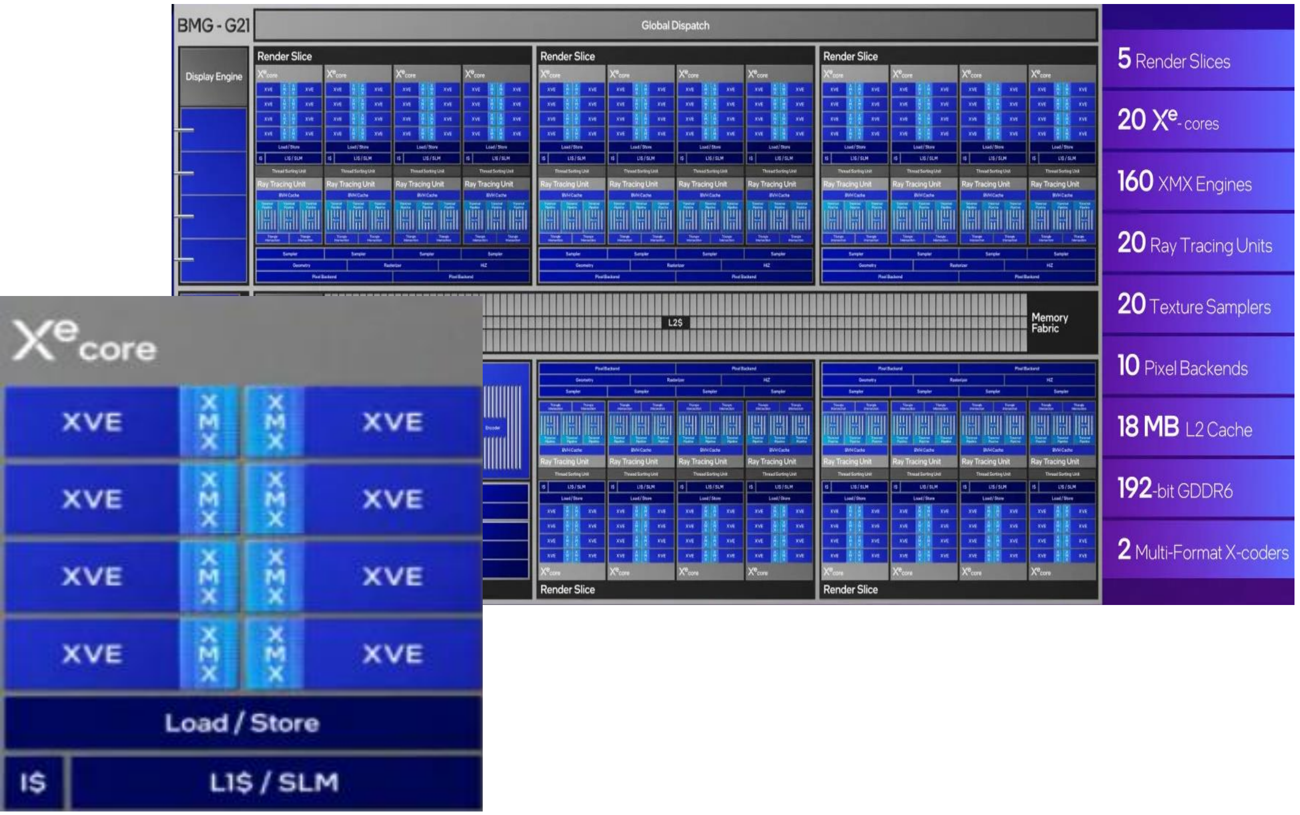
Outline

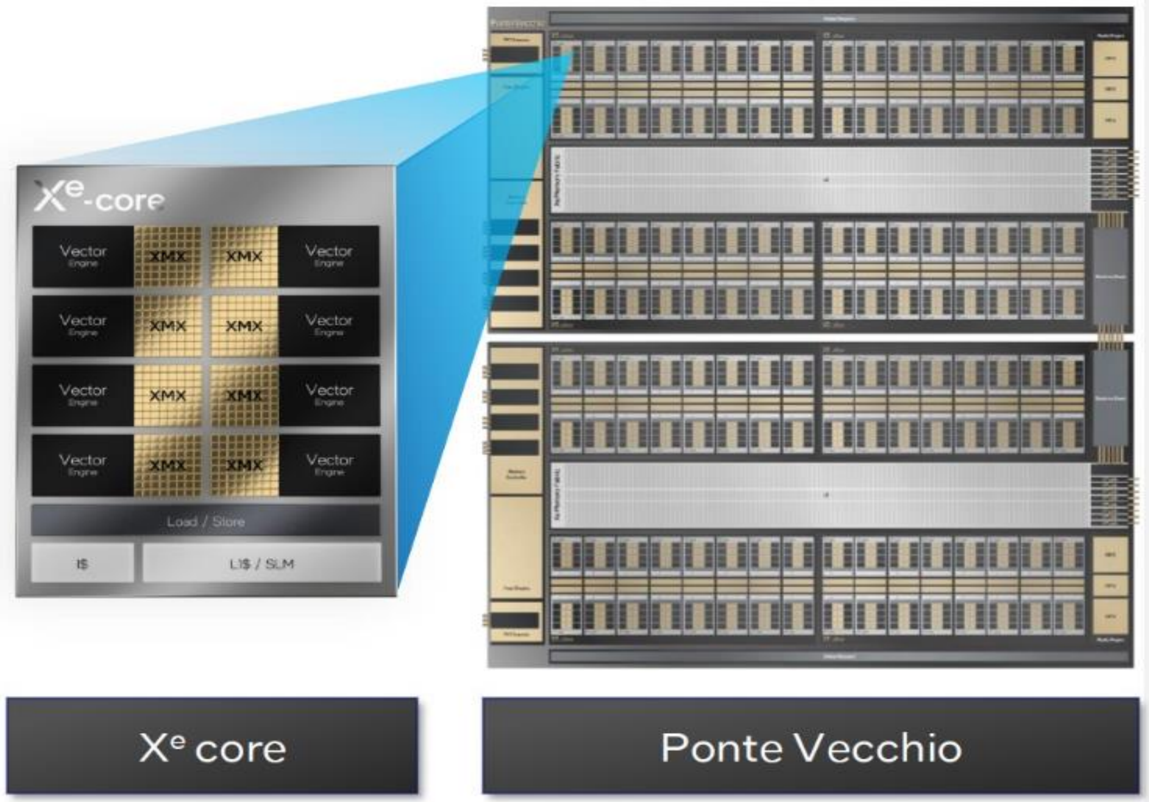
- Matrix Hardware
- SYCL Joint Matrix Extension
- Tuning for Performance
- Conclusion and Next Steps

Matrix Hardware

Intel XMX in Intel® Client GPU Arc B-Series Discrete Graphics

Intel XMX in Intel® Data Center GPU Max Series





5 slices with each Slice contains 4 Xe-core

An Xe -core contains 8 vector and 8 matrix engines

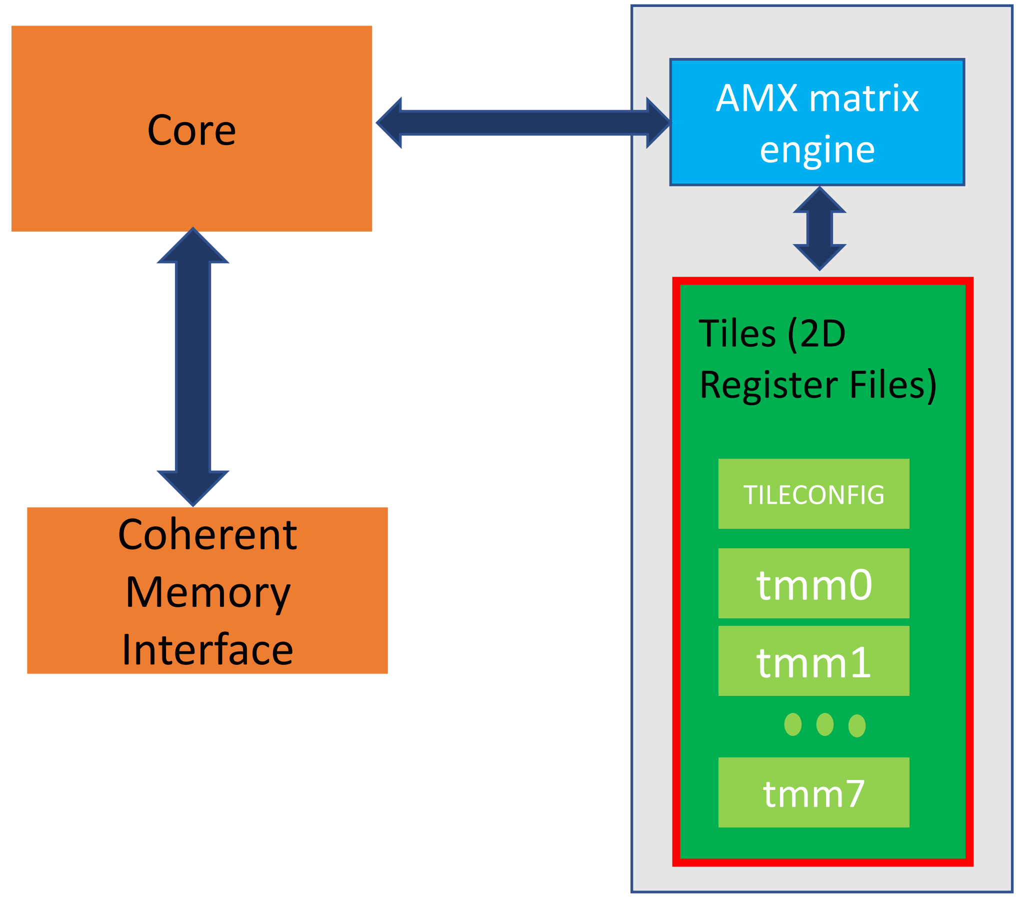
Each Xe-Stack has 4 slices with each Xe-slice contains 16 Xe-core

An Xe -core contains 8 vector and 8 matrix engines (512 XMX Engines)

Intel XMX multiplication of matrices of bfloat16, half, tf32, int8, and int4 elements

Intel AMX High-Level Architecture

- Intel® Xeon® 6 processor
- Intel AMX, an Intel x86 extension for multiplication of matrices of bfloat16, half, and int8 elements



SYCL Joint Matrix Extension

SYCL Joint Matrix Extension: New Data Type

- Namespace sycl::ext::oneapi::experimental::matrix
- New matrix data type with group scope
- Defined with the following fields:
  - a specified type,
  - use (a, b, accumulator),
  - shape,
  - and layout

```
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};
```

SYCL Joint Matrix Extension: Memory Operations

- Separate memory operations from the compute
- Group execution scope → *joint, Group as argument*
- Prefetch for more performance

```
• 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);
```

SYCL Joint Matrix Extension: Compute Operations

- Multiply and add
- Element-wise operations, activation functions, quantization
- Copy between matrix for type and use conversions

```
•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);
```

SYCL joint_matrix

```
// inputA is MxK, inputB is KxN, inputC is MxN

#define tM=16  tN=16  tK=16


void gemm(size_t global_idx, size_t global_idy, size_t local_idx, size_t local_idy, sub_group sg)  {

  joint_matrix<sub_group, half, use::a, tM, tK, row_major> matA;
  joint_matrix<sub_group, half, use::b, tK, tN, row_major> matB;
  joint_matrix<sub_group, float, use::accumulator, tM, tN> matC;

  const auto sg_startx = global_idx - local_idx;
  const auto sg_starty = global_idy - local_idy;

  joint_matrix_fill(matC, 0.0f);

  for (int step = 0; step < K; step += tK) {

   uint AStart = sg_startx * tM * K + step;
   uint BStart = step * N + sg_starty;
   joint_matrix_load(sg, matA, inputA + AStart, K);
   joint_matrix_load(sg, matB, inputB + BStart, N);
   joint_matrix_mad(sg, matC, matA, matB, matC);

  }


  joint_matrix_apply(sg, matC, [=](T& x) { x *= alpha; });

  joint_matrix_store(sg, matC, output + sg_startx * tM * N + sg_starty, N, row_major);

}
```

Parameters are not portable → use query

CUDA Fragments

```
// inputA is MxK, inputB is KxN, inputC is MxN

#define tM=16 tN=16 tK=16


__global__ void wmma_ker(blockidx) {

  fragment<matrix_a, tM, tN, tK, half, col_major> matA;
  fragment<matrix_b, tM, tN, tK, half, row_major> matB;
  fragment<accumulator, tM, tN, tK, float> matC;

  uint row = (blockIdx%x - 1)*tM + 1;
  uint col = (blockIdx%y - 1)*tN + 1;

  fill_fragment(matC, 0.0f);

  for (uint step = 0; step < K; step += matrixDepth) {

      uint AStart = row * rowStrideA + step;
      uint BStart = col * colStrideB + step;

      load_matrix_sync(matA, inputA + AStart, K);

      load_matrix_sync(matB, inputB + BStart, N);

      mma_sync(matC, matA, matB, matC);

  }
  for(int t=0; t<matC.num_elements; t++)

      matC.x[t] *= alpha;

  store_matrix_sync(inputC+row*N+col, matC, N, mem_row_major);

}
```

- The same joint matrix code can run on Intel AMX, Intel XMX, Nvidia* Tensor Cores and AMD Matrix Cores
- SYCL joint matrix is available in Intel® DPC++ Compatibility Tool

SYCL Matrix Extension: Intel Specific Features
SYCL joint_matrix Indexing with Coordinates

- Element wise ops that apply to a set of elements of the matrix → Mapping is required
- Example: Quantization Calculations
- *A*B + sum_rows_A + sum_cols_B + scalar_zero_point*
- *sum_rows_A* returns a single row of A

```
using namespace sycl::ext::intel::experimental::matrix;

void sum_rows_A(joint_matrix<T, rows, cols>& subA)
{
  joint_matrix_apply(sg, subA, [=](T &val, size_t row, size_t  col) {
     global_row = row + global_idx * rows;
     sum_local_rows[global_row] += val;
  });
}
```

Tuning for Performance

Blocking for Data Reuse

Bfloat16 type example	SYCL terms	Compute unit and data	Intel XMX	Intel AMX
Workgroup execution hierarchy level/ Cache locality	Local range	• Data in L1 cache	(MCACHExNCACHExKCACHE) =256x256x32 for 4096 GEMM size	(MCACHExNCACHExKCACHE) = 256x256x1024 for 4096 GEMM size
Subgroup execution hierarchy level	Inside the kernel	• Data in registers	MSGxNSGxKSG = 32x64x16 → 32 SGs/WG	MSGxNSGxKSG = 32x32x32 to occupy the 8 AMX tiles →64 SGs/WG
Joint matrix shape	Inside the kernel	Matrix hardware	MJMxNJMxKJM = 8x16x16 → 16 DPAS instructions	MJMxNJMxKJM = 16x16x32 → 4 AMX instructions

Blocking for Data Reuse

```
using namespace sycl::ext::oneapi::experimental::matrix;
queue q;
range<2> G = {M/8, N/16 * SG_SIZE};
range<2> L = {1, 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<… 8, 16…,> A;
     joint_matrix<…, 16, 16,…> B;
     joint_matrix<… 8, 16, …> C;
     joint_matrix_fill(sg, subC, 0);
     for (int k = 0; k < K; k += tk) {
       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;
```

Hardware tile size

```
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;
```

SG tile or hardware tile

Kernel tile	MxNxK	4096x4096x4096
Work group tile	MWGxNWGxKWG	256x256x32
Subgroup tile	MSGxNSGxKSG	32x64x16
Joint matrix	MJMxNJMxKJM	8x16x16 or 32x64x16

oneAPI

intel

More Tuning for Performance

- **Prefetch:**
  - Prefetch of Matrix A and Matrix B using joint_matrix_prefetch API
- **Cache control on annotated pointers**
  - Use of cache control properties on annotated pointers
  - Use annotated pointer instead of multi_ptr in joint_matrix_load/store
  - Bypass the cache for Matrix A

```
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});
```

```
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);
```

Conclusion and Next Steps

- Full support of SYCL joint matrix extension on Intel AMX, Intel XMX, Nvidia Tensor Cores, and AMD Matrix Cores
- Matrix extensions using SPIRV cooperative matrix
- Comparable performance to oneDNN but more flexibility and control on kernel fusions and matrix hardware
- Effective usage in MLIR integration and CUDA code migration
- SYCL joint matrix is available in Intel® DPC++ Compatibility Tool
- In progress: more data types like FP8, more execution scopes, and more kernels like softmax (**https://github.com/dkhaldi/sycl_joint_matrix_kernels/**)

Legal Notices & Disclaimers