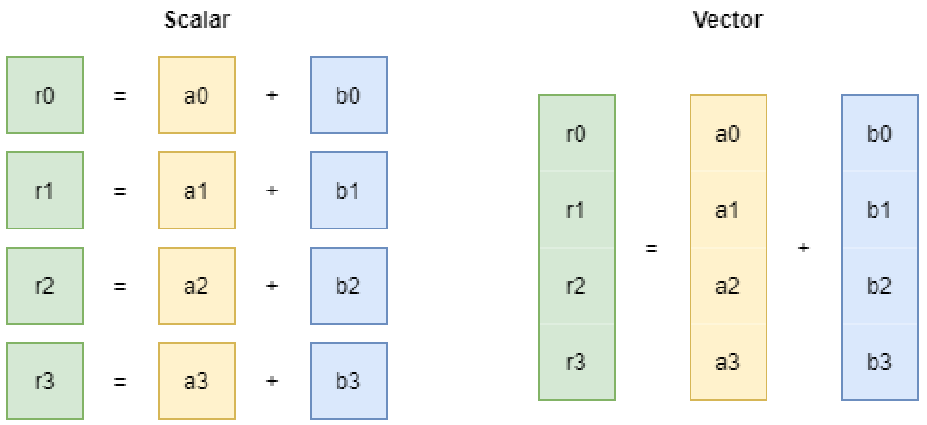
SYCL Academy

VECTORIZATION

LEARNING OBJECTIVES

- Learn about scalar and vector instructions
- Learn about horizontal and vertical vectorization
- Learn how to write explicit vector code
- learn how to use swizzles

VECTOR INSTRUCTIONS

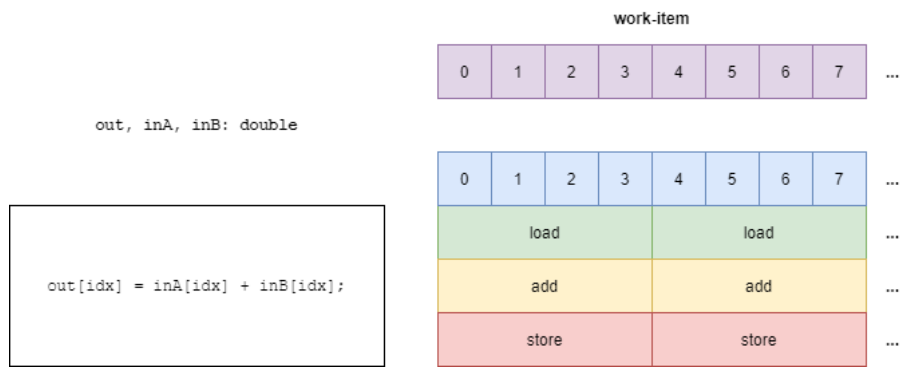


- Data parallel devices such as GPUs, SIMD CPUs and other accelerators are vector processors.
- This means they can execute vector instructions.
- Vector instructions are single instructions which perform loads, stores, or operations such as add or multiply on multiple elements at once.

VECTORIZATION

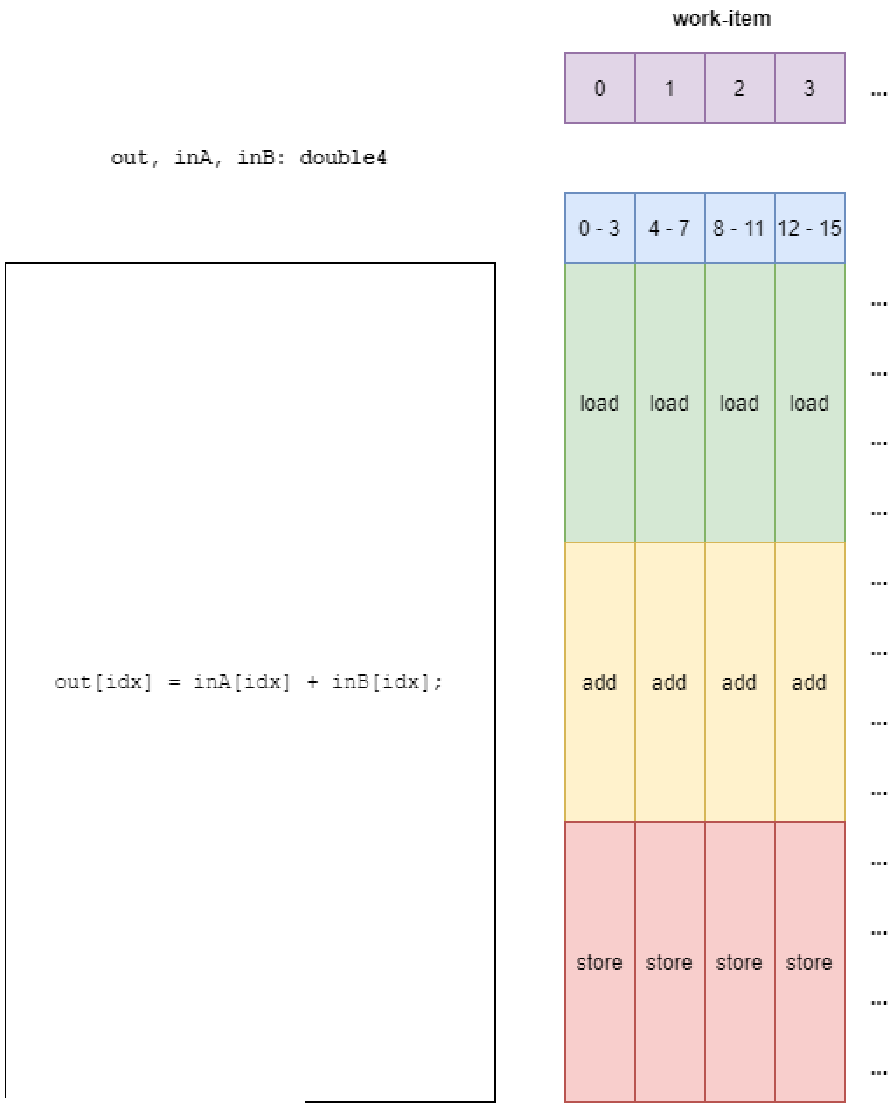
- Vectorization is the process of converting scalar code into vectorized code.
- In a SPMD programming model like SYCL vectorization is important.
- Vectorization can be performed in two ways, and it depends on how you write your code and can impact the mapping to hardware.

HORIZONTAL VECTORIZATION



- Horizontal (or auto-) vectorization is done automatically by the compiler.
- It maps the scalar operation of each work-item to a single processing element, or element of a vector instruction.

VERTICAL VECTORIZATION



- Vertical (or explicit) vectorization is done by using vector types explicitly.
- It maps the vector instruction of each work-item to multiple processing elements, or elements of vector instructions.

HORIZONTAL VS VERTICAL VECTORIZATION

- Both horizontal and vertical vectorization generally achieve the same result.
- It can be useful to specify vectorization explicitly, particularly for describing aligned loads and stores.
- An important distinction to make is that whether a kernel function uses explicit vector types can impact the mapping of work-items to processing elements.
- It's not always a 1:1 mapping.

VEC CLASS

```
template <typename dataT, int numElements>
class vec;
```

- The vec class template is used to represent explicit vectors in SYCL.
- It has a type which represents the type of elements it stores and a number of elements.
- The valid number of elements are 1, 2, 3, 4, 8, 16.
- Note that vectors of 3 elements are padded to the size of 4.

ALIASES

```
using float4 = vec<float, 4>;
...
```

- A number of aliases are provided for shorthand with the notation of the type followed by the size, such as `float4`.

VEC CONSTRUCTORS

```
auto f4 = sycl::float4{1.0f, 2.0f, 3.0f, 4.0f}; // {1.0f, 2.0f, 3.0f, 4.0f}
```

```
auto f2 = sycl::float4{2.0f, 3.0f}; // {2.0f, 3.0f}
auto f4 = sycl::float4{1.0f, f2, 4.0f}; // {1.0f, 2.0f, 3.0f, 4.0f}
```

```
auto f4 = sycl::float4{0.0f};  // {0.0f, 0.0f, 0.0f, 0.0f}
```

- A vec can be constructed with any combination of scalar and vector values which add up to the correct number of elements.
- A veccan also be constructed from a single scalar in which case it will initialize every element to that value.

VEC OPERATORS

```
auto f4a = sycl::float4{1.0f, 2.0f, 3.0f, 4.0f}; // {1.0f, 2.0f, 3.0f, 4.0f}

auto f4b = sycl::float4{2.0f}; // {2.0f, 2.0f, 2.0f, 2.0f}

auto f4r = f4a * f4b; // {2.0f, 4.0f, 6.0f, 8.0f}
```

- The vec class provides a number of operators such as +, -, *, / and many more, which perform the operation element-wise.

SWIZZLES

```
auto f4 = sycl::float4{1.0f, 2.0f, 3.0f, 4.0f}; // {1.0f, 2.0f, 3.0f, 4.0f}
auto f2 = f4.swizzle<0, 3>(); // {1.0f, 4.0f}
```

```
auto f4 = sycl::float4{1.0f, 2.0f, 3.0f, 4.0f}; // {1.0f, 2.0f, 3.0f, 4.0f}
f4.swizzle<1, 2>() = sycl::float2{9.0f, 9.0f}; // f4 becomes {1.0f, 9.0f, 9.0f, 4.0f}
```

- The `swizzle` function returns a representation of the specified elements of a `vec` which can be used on the lhs or rhs of an expression.

SIMPLE SWIZZLES

```
auto f4 = sycl::float4{1.0f, 2.0f, 3.0f, 4.0f}; // {1.0f, 2.0f, 3.0f, 4.0f}
auto f2 = f4.xw(); // {1.0f, 4.0f}
```

```
auto f4 = sycl::float4{1.0f, 2.0f, 3.0f, 4.0f}; // {1.0f, 2.0f, 3.0f, 4.0f}
f4.yz() = sycl::float2{9.0f, 9.0f}; // f4 becomes {1.0f, 9.0f, 9.0f, 4.0f}
```

- If `SYCL_SIMPLE_SWIZZLES` is defined before including `sycl/sycl.hpp` simplified swizzle member functions can also be used in place of `swizzle`.

VECTORIZED IMAGE CONVOLUTION PERFORMANCE

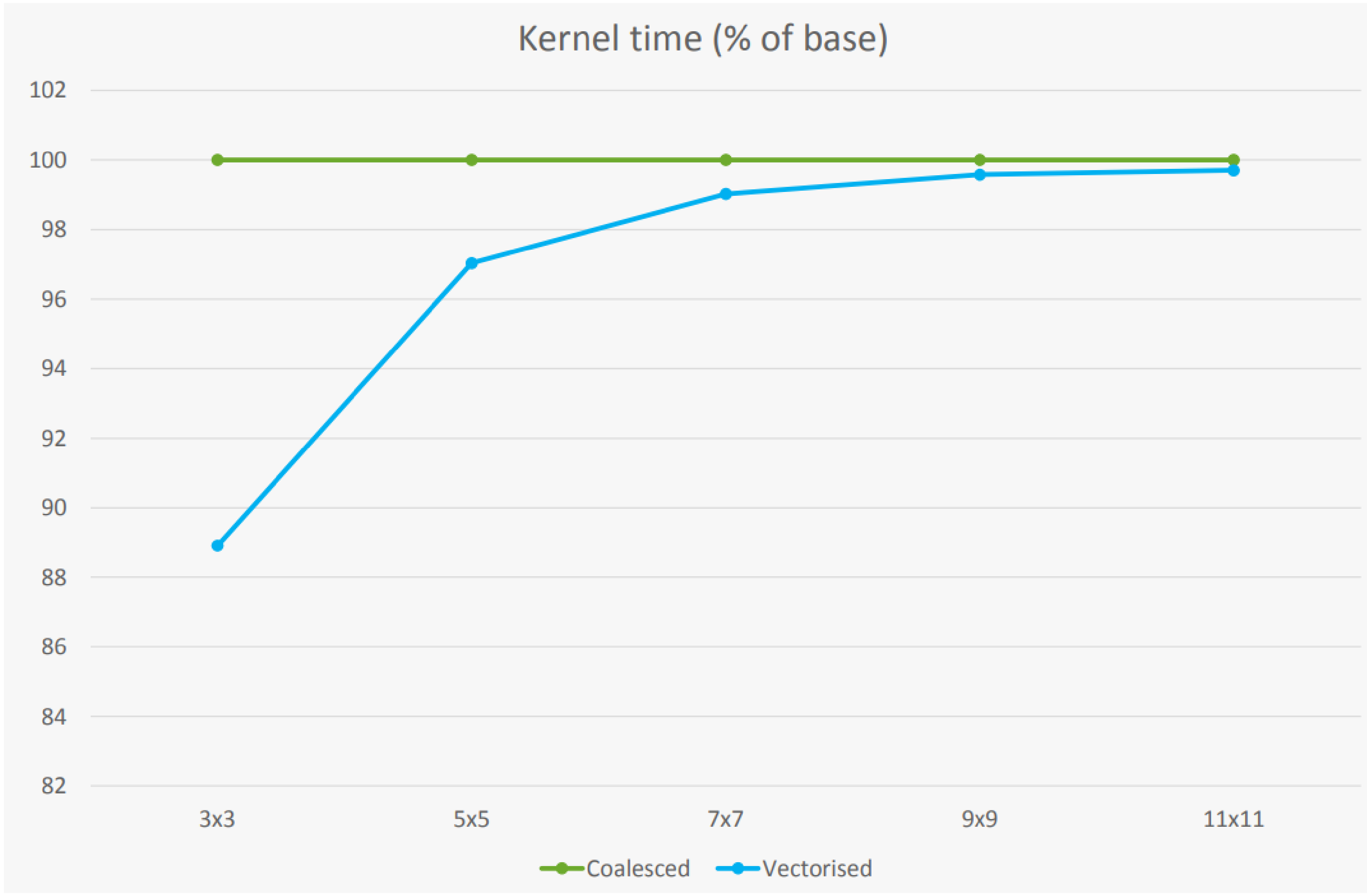


Image conv

512x512 source image

Intel HD Graphics 530

MARRAY CLASS

```
template <typename DataT, std::size_t NumElements>
class marray;
```

- Motivation
  - vec class is mostly used as a vector in SPMD code
  - For SIMD use vec class, for vectors in SPMD use new math array types
- marray is templated on its element type and number of elements
- The type of elements must be a numeric type
- Improved usage
  - Size never contains padding (vec of 3 elements is 4 long)
  - Element access doesn’t require swizzle and write to element is allowed

ALIASES

```
using mfloat4 = marray<float, 4>;
...
```

- A number of aliases are provided for shorthand with the notation of the type followed by the size, such as float4.
- More examples:
    - muint4 is the alias to marray<uint32_t, 4>
    - mfloat16 is the alias to marray<float, 16>

SYCL Academy

QUESTIONS

EXERCISE

Code_Exercises/Exercise_17_Vectors/source

Update the image convolution application to use vectors types.