Article

Calculating threadgroup and grid sizes

Calculate the optimum sizes for threadgroups and grids when dispatching compute-processing workloads.

Overview

You can ensure your app doesn't underuse threads by specifying the size of the grid and the number of threads per threadgroup. In iOS 11 and macOS 10.13 and later on devices that support nonuniform dispatch sizes, Metal calculates the number of threadgroups and provides nonuniform threadgroups if the grid size isn't a multiple of the threadgroup size.

In earlier versions of iOS and macOS, you specify the size and number of the threadgroups. Metal composes grids of uniform threadgroups that may not match the size of your data. You can ensure your kernel code doesn't execute outside the bounds of the data by adding defensive code to it.

Calculate threads per threadgroup

You calculate the number of threads per threadgroup based on two `MTLComputePipelineState` properties:

`maxTotalThreadsPerThreadgroup`
: The maximum number of threads that can be in a single threadgroup, which depends on the GPU and on the amount of registers and memory your compute kernel needs.

`threadExecutionWidth`
: The number of threads the GPU schedules to execute in parallel.

> **Note**
>
> After you create a compute pipeline state, its `maxTotalThreadsPerThreadgroup` value doesn't change, but two pipeline states on the same device may return different values.

For example, consider a compute pipeline state with 512 maximum threads per threadgroup and a thread execution width of 16. For that compute pipeline state, you can launch the largest possible threadgroup by setting the following:

- The second dimension to the maximum threads per thread group divided by the thread execution width
- The third dimension to 1

Swift Objective-C

```
// Calculate the maximum threads per threadgroup based on the thread execution width
let w = pipelineState.threadExecutionWidth
let h = pipelineState.maxTotalThreadsPerThreadgroup / w
let threadsPerThreadgroup = MTLSizeMake(w, h, 1)
```

On devices that support nonuniform threadgroup sizes, Metal divides a grid into nonuniform, arbitrarily sized threadgroups, such as for an image or texture. The compute command encoder's `dispatchThreads(_:threadsPerThreadgroup:)` method requires the total number of threads because each thread corresponds to a single pixel.

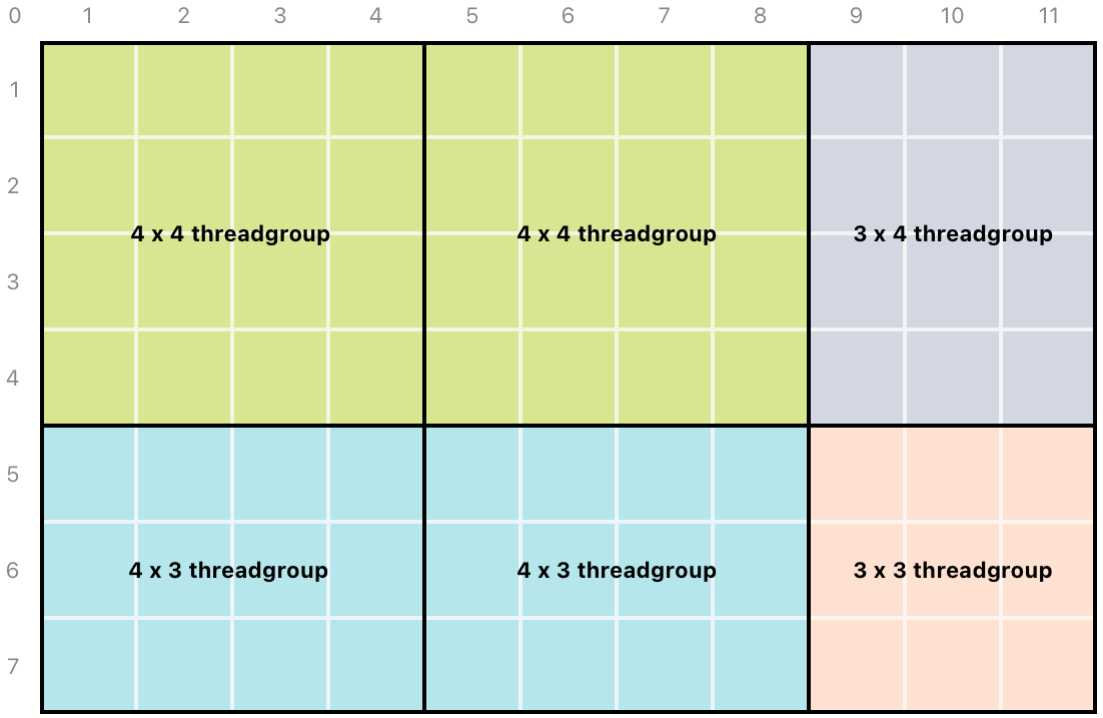
Swift Objective-C

```
let threadsPerGrid = MTLSize(width: texture.width,
                             height: texture.height,
                             depth: 1)

computeCommandEncoder.dispatchThreads(threadsPerGrid,
                                      threadsPerThreadgroup: threadsPerThreadgroup)
```

When Metal performs this calculation, it can generate smaller threadgroups along the edges of your grid. Compared to uniform threadgroups, this technique simplifies kernel code and improves GPU performance.

To determine if a device supports nonuniform threadgroups, see Metal Feature Set Tables.



Calculate threadgroups per grid

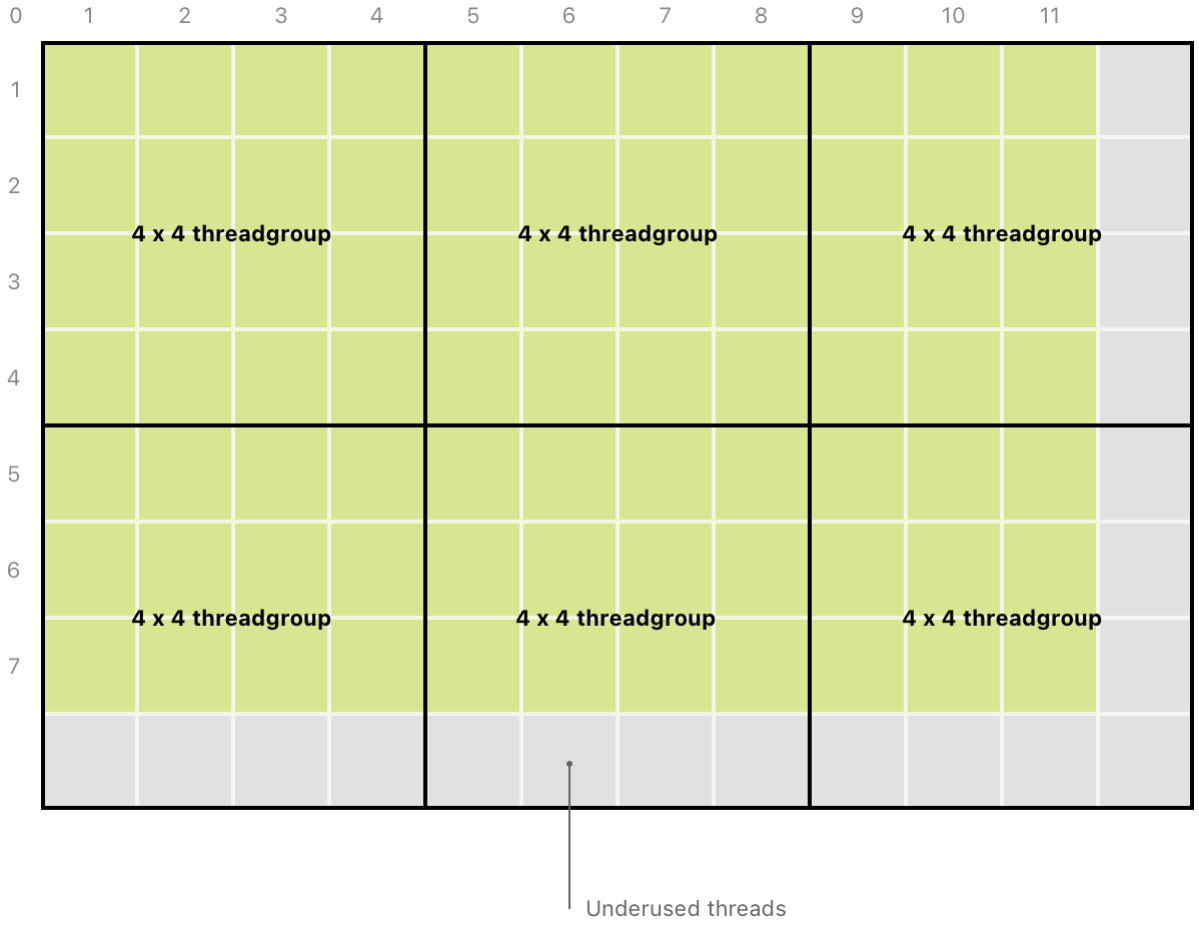
If you need fine control over the size and number of threadgroups, you can manually calculate how to divide the grid. In your code, ensure that there are sufficient threadgroups to cover the entire image.

Swift Objective-C

```
let threadgroupsPerGrid = MTLSize(width: (texture.width + w - 1) / w,
                                  height: (texture.height + h - 1) / h,
                                  depth: 1)
```

For a texture that's 1024 by 768 pixels in size, the code above returns an `MTLSize` instance with a width of 32, a height of 48, and a depth of 1. These values divide the texture into 1536 threadgroups, each of which contains 512 threads, for a total of 786,432 threads. In this case, that number of threads matches the number of pixels in the image, and the GPU processes the entire image with no underuse of threads.

However, the code may round up to ensure there are sufficient threads to process the entire image, such as for an image of 1920 by 1080 pixels in size. This approach can result in the threadgroups generating a grid that's larger than your data.



To compensate for the extra threads, you can make your code exit early if the thread position in the grid is outside the bounds of the data.

```
kernel void
simpleKernelFunction(texture2d<float, access::write> outputTexture [[texture(0)]],
                     uint2 position [[thread_position_in_grid]]) {

    if (position.x >= outputTexture.get_width() || position.y >= outputTexture.get_h
        return;
    }

    outputTexture.write(float4(1.0), position);
}
```

> **Note**
>
> You don't need to check a thread's position in a grid if you use the `dispatchThreads(_:threadsPerThreadgroup:)` technique.

Encode the command that executes your custom threadgroup size by calling the encoder's `dispatchThreadgroups(_:threadsPerThreadgroup:)` method.

Swift Objective-C

```
computeCommandEncoder.dispatchThreadgroups(threadgroupsPerGrid,
                                           threadsPerThreadgroup: threadsPerThreadgr
```

See Also

Encoding a compute pass

Creating threads and threadgroups

Learn how Metal organizes compute-processing workloads.

`protocol` MTL4ComputeCommandEncoder

Encodes a compute pass and other memory operations into a command buffer.

`protocol` MTLComputeCommandEncoder

An interface for dispatching commands to encode in a compute pass.