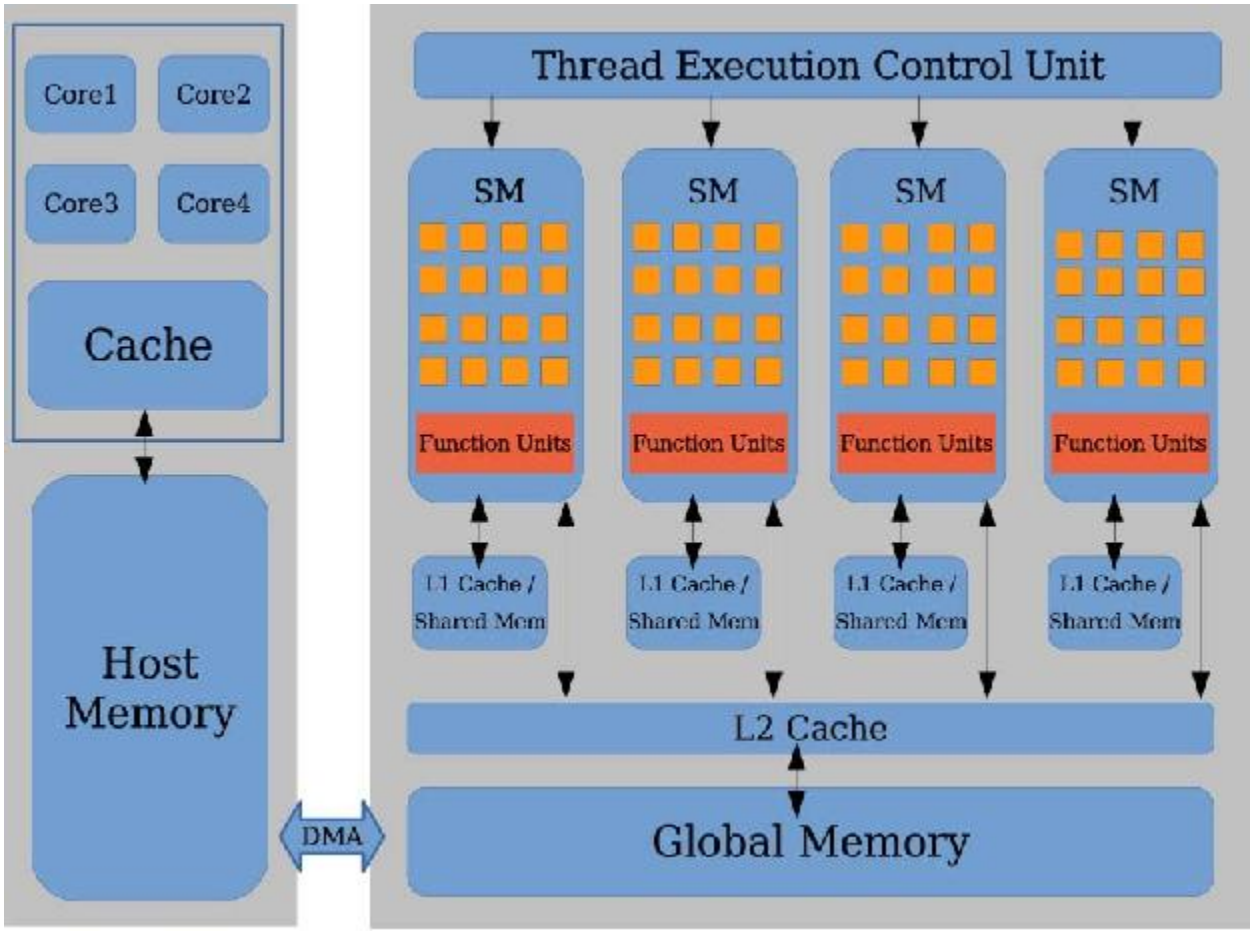
Heterogeneous computing

Heterogeneous computing refers to systems that use more than one kind of processor or cores. These systems gain performance or energy efficiency not just by adding the same type of processors, but by adding dissimilar coprocessors, usually incorporating specialized processing capabilities to handle particular tasks



C++ AMP Overview

++ Accelerated Massive Parallelism (C++ AMP) accelerates execution of C++ code by taking advantage of data-parallel hardware such as a graphics processing unit (GPU) on a discrete graphics card. By using C++ AMP, you can code multi-dimensional data algorithms so that execution can be accelerated by using parallelism on heterogeneous hardware. The C++ AMP programming model includes multidimensional arrays, indexing, memory transfer, tiling, and a mathematical function library. You can use C++ AMP language extensions to control how data is moved from the CPU to the GPU and back, so that you can improve performance.

system Requirements

- Windows 7 or later
- Windows Server 2008 R2 or later
- DirectX 11 Feature Level 11.0 or later hardware
- For debugging on the software emulator, Windows 8 or Windows Server 2012 is required. For debugging on the hardware, you must install the drivers for your graphics card. For more information, see Debugging GPU Code.
- Note: AMP is currently not supported on ARM64.

Introduction

The following two examples illustrate the primary components of C++ AMP. Assume that you want to add the corresponding elements of two one-dimensional arrays. For example, you might want to add `{1, 2, 3, 4, 5}` and `{6, 7, 8, 9, 10}` to obtain `{7, 9, 11, 13, 15}`. Without using C++ AMP, you might write the following code to add the numbers and display the results.

C++Copy

```
#include <iostream>

void StandardMethod() {

    int aCPP[] = {1, 2, 3, 4, 5};
    int bCPP[] = {6, 7, 8, 9, 10};
    int sumCPP[5];

    for (int idx = 0; idx < 5; idx++)
    {
        sumCPP[idx] = aCPP[idx] + bCPP[idx];
    }
```

```
    for (int idx = 0; idx < 5; idx++)
    {
        std::cout << sumCPP[idx] << "\n";
    }
}
```

The important parts of the code are as follows:

- Data: The data consists of three arrays. All have the same rank (one) and length (five).
- Iteration: The first `for` loop provides a mechanism for iterating through the elements in the arrays. The code that you want to execute to compute the sums is contained in the first `for` block.
- Index: The `idx` variable accesses the individual elements of the arrays.

Using C++ AMP, you might write the following code instead.

C++Copy

```
#include <amp.h>
#include <iostream>
using namespace concurrency;

const int size = 5;

void CppAmpMethod() {
    int aCPP[] = {1, 2, 3, 4, 5};
    int bCPP[] = {6, 7, 8, 9, 10};
    int sumCPP[size];

    // Create C++ AMP objects.
    array_view<const int, 1> a(size, aCPP);
    array_view<const int, 1> b(size, bCPP);
    array_view<int, 1> sum(size, sumCPP);
    sum.discard_data();

    parallel_for_each(
        // Define the compute domain, which is the set of threads that are created.
        sum.extent,
        // Define the code to run on each thread on the accelerator.
        [=](index<1> idx) restrict(amp) {
            sum[idx] = a[idx] + b[idx];
        }
    );
```

```
    // Print the results. The expected output is "7, 9, 11, 13, 15".
    for (int i = 0; i < size; i++) {
        std::cout << sum[i] << "\n";
    }
}
```

Debugging a C++ AMP Application

This topic demonstrates how to debug an application that uses C++ Accelerated Massive Parallelism (C++ AMP) to take advantage of the graphics processing unit (GPU). It uses a parallel-reduction program that sums up a large array of integers. This walkthrough illustrates the following tasks:

- Launching the GPU debugger.
- Inspecting GPU threads in the GPU Threads window.
- Using the **Parallel Stacks** window to simultaneously observe the call stacks of multiple GPU threads.
- Using the **Parallel Watch** window to inspect values of a single expression across multiple threads at the same time.
- Flagging, freezing, thawing, and grouping GPU threads.
- Executing all the threads of a tile to a specific location in code.

Prerequisites

Before you start this walkthrough:

- Read C++ AMP Overview.
- Make sure that line numbers are displayed in the text editor. For more information, see How to: Display Line Numbers in the Editor.
- Make sure you are running at least Windows 8 or Windows Server 2012 to support debugging on the software emulator.