CUDA is an extension to the C language that allows GPU code to be written in regular C. The code

is either targeted for the host processor (the CPU) or targeted at the device processor (the GPU). The

host processor spawns multithread tasks (or kernels as they are known in CUDA) onto the GPU device.

The GPU has its own internal scheduler that will then allocate the kernels to whatever GPU hardware is

present. We’ll cover scheduling in detail later. Provided there is enough parallelism in the task, as the

number of SMs in the GPU grows, so should the speed of the program.

However, herein hides a big problem. You have to ask what percentage of the code can be run in

parallel. The maximum speedup possible is limited by the amount of serial code. If you have an infinite

amount of processing power and could do the parallel tasks in zero time, you would still be left with the

time from the serial code part. Therefore, we have to consider at the outset if we can indeed parallelize

a significant amount of the workload.

NVIDIA is committed to providing support to CUDA. Considerable information, examples, and

tools to help with development are available fromits website at http://www.nvidia.com under CudaZone.

CUDA, unlike its predecessors, has now actually started to gain momentum and for the first time it

looks like there will be a programming language that will emerge as the one of choice for GPU

programming. Given that the number of CUDA-enabled GPUs now number in the millions, there is

a huge market out there waiting for CUDA-enabled applications.

There are currently manyCUDA-enabled applications and the list growsmonthly.NVIDIA showcases

many of these on its community website at http://www.nvidia.com/object/cuda\_apps\_flash\_new.html.

In areas where programs have to do a lot of computational work,for example, making a DVD

from your home movies (video transcoding),we see most mainstream video packages now supporting

CUDA. The average speedup is 5 to 10 times in this domain.

ALTERNATIVES TO CUDA

OpenCL

So what of the other GPU manufacturers, ATI (now AMD) being the prime example.AMD’s product

range is as impressive as the NVIDIA range in terms of raw computer power. However, AMD brought

its stream computing technology to the marketplace a long time after NVIDIA brought out CUDA. As

a consequence, NVIDA has far more applications available for CUDA than AMD/ATI does for its

competing stream technology.

OpenCL and Direct compute is not something we’ll cover in this book, but they deserve a mention

in terms of alternatives to CUDA. CUDA is currently only officially executable on NVIDIA hardware.

While NVIDIA has a sizeable chunk of the GPU market, its competitors also hold a sizeable chunk. As

developers, we want to develop products for as large a market as possible, especially if we’re talking

about the consumer market. As such, people should be aware there are alternatives to CUDA, which

support both NVIDIA’s and others’ hardware.

OpenCL is an open and royalty-free standard supported by NVIDIA, AMD, and others. The

OpenCL trademark is owned by Apple. It sets out an open standard that allows the use of compute

devices. A compute device can be a GPU, CPU, or other specialist device for which an OpenCL driver

exists. As of 2012, OpenCL supports all major brands of GPU devices, including CPUs with at least

SSE3 support.

Anyone who is familiar with CUDA can pick up OpenCL relatively easily, as the fundamental

concepts are quite similar. However, OpenCL is somewhatmore complex to use thanCUDA, in thatmuch

of thework theCUDAruntime API does for the programmer needs to be explicitly performed in OpenCL.

You can read more about OpenCL at http://www.khronos.org/opencl/. There are also now a number

of books written on OpenCL. I’d personally recommend learning CUDA prior to OpenCL as CUDA is

somewhat of a higher-level language extension than OpenCL.

DirectCompute

DirectCompute is Microsoft’s alternative to CUDA and OpenCL. It is a proprietary product linked to

the Windows operating system, and in particular, the DirectX 11 API. The DirectX API was a huge

leap forward for any of those who remember programming video cards before it. It meant the

developers had to learn only one library API to program all graphics cards, rather than write or license

drivers for each major video card manufacturer.

DirectX 11 is the latest standard and supported under Windows 7. With Microsoft’s name behind

the standard, you might expect to see some quite rapid adoption among the developer community. This

is especially the case with developers already familiar with DirectX APIs. If you are familiar with

CUDA and DirectCompute, then it is quite an easy task to port a CUDA application over to Direct-

Compute. According to Microsoft, this is something you can typically do in an afternoon’s work if you

are familiar with both systems. However, being Windows centric, we’ll exclude DirectCompute from

many high-end systems where the various flavors of UNIX dominate.

Microsoft are also set to launch Ctt AMP, an additional set of standard template libraries (STLs),

which may appeal more to programmers already familiar with Ctt-style STLs.

CPU alternatives

The main parallel processing languages extensions are MPI, OpenMP, and pthreads if you are

developing for Linux. For Windows there is the Windows threading model and OpenMP. MPI and

pthreads are supported as various ports from the Unix world.

MPI (Message Passing Interface) is perhaps the most widely known messaging interface. It is

process-based and generally found in large computing labs. It requires an administrator to

configure the installation correctly and is best suited to controlled environments. Parallelism is

expressed by spawning hundreds of processes over a cluster of nodes and explicitly exchanging

messages, typically over high-speed network-based communication links (Ethernet or

InfiniBand). MPI is widely used and taught. It’s a good solution within a controlled cluster

environment.

OpenMP (Open Multi-Processing) is a system designed for parallelism within a node or

computer system. It works entirely differently, in that the programmer specifies various

parallel directives through compiler pragmas. The compiler then attempts to automatically

split the problem into N parts, according to the number of available processor cores. OpenMP

support is built into many compilers, including the NVCC compiler used for CUDA. OpenMP

tends to hit problems with scaling due to the underlying CPU architecture. Often the memory

bandwidth in the CPU is just not large enough for all the cores continuously streaming data to

or from memory.

Pthreads is a library that is used significantly for multithread applications on Linux. As with

OpenMP, pthreads uses threads and not processes as it is designed for parallelism within

a single node. However, unlike OpenMP, the programmer is responsible for thread management

and synchronization. This provides more flexibility and consequently better performance for

well-written programs.

ZeroMQ (0MQ) is also something that deserves a mention. This is a simple library that you link to,

and we will use it later in the book for developing a multinode, multi-GPU example. ZeroMQ

supports thread-, process-, and network-based communications models with a single crossplatform

API. It is also available on both Linux and Windows platforms. It’s designed for

distributed computing, so the connections are dynamic and nodes fail gracefully.

Hadoop is also something that you may consider. Hadoop is an open-source version of Google’s

MapReduce framework. It’s aimed primarily at the Linux platform. The concept is that you take

a huge dataset and break (or map) it into a number of chunks. However, instead of sending the

data to the node, the dataset is already split over hundreds or thousands of nodes using a parallel

file system. Thus, the program, the reduce step, is instead sent to the node that contains the data.

The output is written to the local node and remains there. Subsequent MapReduce programs take

the previous output and again transform it in some way. As data is in fact mirrored to multiple

nodes, this allows for a highly fault-tolerant as well as high-throughput system.