Akshay Shukla (04/08/10)

Simulation on programmable graphics hardware (GPUs)

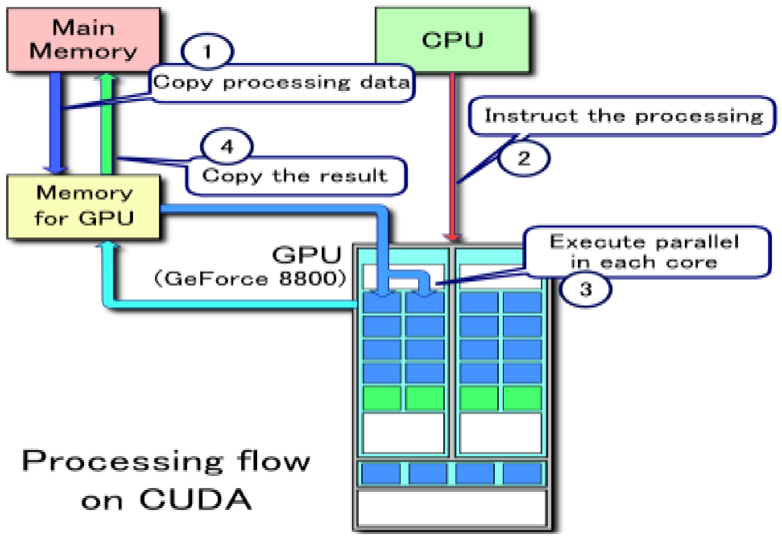
GPGPU:

GPGPU stands for *General-Purpose computation on Graphics Processing Units*, also known as *GPU Computing*. Graphics Processing Units (GPUs) are high-performance many-core processors capable of very high computation and data throughput. Once specially designed for computer graphics and difficult to program, today's GPUs are general-purpose parallel processors with support for accessible programming interfaces and industry-standard languages such as C. But GPGPU programming is too cumbersome and it requires previous knowledge of OpenGL.

CUDA:

CUDA (an acronym for Compute Unified Device Architecture) is NVIDIA's parallel computing architecture that enables dramatic increases in computing performance by harnessing the power of the GPU (graphics processing unit). Computing is evolving from "central processing" on the CPU to "co-processing" on the CPU and GPU. To enable this new computing paradigm, NVIDIA invented the CUDA parallel computing architecture that is now shipping in GeForce, ION, Quadro, and Tesla GPUs, representing a significant installed base for application developers. It provides an API for GPGPU and hence is as powerful as GPGPU but is much easier to code. CUDA can be used for computer graphics code using OpenCL (Open Computing Language), DirectX and FORTRAN. Programmers use 'C for CUDA' (C with NVIDIA extensions), compiled through a PathScale Open64 C compiler, to code algorithms for execution on the GPU.

With the recent launches of Microsoft Windows 7 and Apple Snow Leopard, GPU computing is going mainstream. In these new operating systems, the GPU will not only be the graphics processor, but also a general purpose parallel processor accessible to any application.



Example of CUDA processing flow

1. Copy data from main memory to GPU memory
2. CPU instructs the process to GPU
3. GPU execute parallel in each core
4. Copy the result from GPU memory to main memory

Close To Metal:

Close To Metal ("CTM" in short, originally called *Close-to-the-Metal*) is the name of a beta version of a low-level programming interface developed by ATI (now AMD Graphics Products Group), aimed at enabling GPGPU computing. AMD recently switched from CTM to OpenCL.

GPU has SIMD (Single Instruction Multiple Data) architecture. Using GPUs to perform computations holds a lot of potential for some applications because of the fundamental differences of GPU microarchitectures compared to CPUs. GPUs achieve much greater throughput (calculations per second) by executing many programs in parallel and restricting flow control (the ability of one program to execute instructions independently of another). Modern GPUs also have addressable on-die memory and extremely high performance multi-channel external memory.

OpenCL:

OpenCL is the first open, royalty-free standard for cross-platform, parallel programming of modern processors found in personal computers, servers and handheld/embedded devices. OpenCL (Open Computing Language) is a framework for writing programs that execute across heterogeneous platforms consisting of CPUs, GPUs, and other processors. OpenCL was initially developed by Apple Inc., which holds trademark rights, and refined into an initial proposal in collaboration with technical teams at AMD, IBM, Intel, and NVidia. It is similar to CUDA and is like a cross platform version of CUDA. OpenCL includes a language (based on C99) for writing *kernels* (functions that execute on OpenCL devices), plus APIs that are used to define and then control the platforms. OpenCL provides parallel computing using task-based and data-based parallelism. Its architecture shares a range of computational interfaces with two competitors, NVidia's Compute Unified Device Architecture and Microsoft's DirectCompute.

OpenCL gives any application access to the Graphical Processing Unit for non-graphical computing. The GPU had previously been available for graphical applications only. The GPU memory would be available to the operating system and or applications essentially as faster system memory than the main system memory. Thus, OpenCL extends the power of the Graphical Processing Unit beyond graphics.

CUDA programing:

Simple CUDA programs have a basic flow:

- The host initializes an array with data.

- The array is copied from the host to the memory on the CUDA device.
- The CUDA device operates on the data in the array.
- The array is copied back to the host.

A sample CUDA code is given below (code taken from: http://llpanorama.wordpress.com/2008/05/21/my-first-cuda-program/):

```
// example1.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"

#include <stdio.h>
#include <cuda.h>

// Kernel that executes on the CUDA device
__global__ void square_array(float *a, int N)
{
  int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (idx<N) a[idx] = a[idx] * a[idx];
}

// main routine that executes on the host
int main(void)
{
  float *a_h, *a_d;  // Pointer to host & device arrays
  const int N = 10;  // Number of elements in arrays
  size_t size = N * sizeof(float);
  a_h = (float *)malloc(size);        // Allocate array on host
  cudaMalloc((void **) &a_d, size);   // Allocate array on device
  // Initialize host array and copy it to CUDA device
  for (int i=0; i<N; i++) a_h[i] = (float)i;
  cudaMemcpy(a_d, a_h, size, cudaMemcpyHostToDevice);
  // Do calculation on device:
  int block_size = 4;
  int n_blocks = N/block_size + (N%block_size == 0 ? 0:1);
  square_array <<< n_blocks, block_size >>> (a_d, N);
  // Retrieve result from device and store it in host array
  cudaMemcpy(a_h, a_d, sizeof(float)*N, cudaMemcpyDeviceToHost);
  // Print results
  for (int i=0; i<N; i++) printf("%d %f\n", i, a_h[i]);
  // Cleanup
  free(a_h); cudaFree(a_d);
}
```

Reference:

1. http://www.nvidia.com/object/what_is_cuda_new.html
2. http://www.nvidia.com/docs/IO/47904/VolumeI.pdf
3. http://llpanorama.wordpress.com/2008/05/21/my-first-cuda-program/
4. http://en.wikipedia.org/wiki/OpenCL
5. http://en.wikipedia.org/wiki/CUDA
6. http://www.khronos.org/opencl/
7. http://en.wikipedia.org/wiki/Close_to_Metal
8. http://www.nvidia.com/object/cuda_gpus.html
9. http://www.nvidia.com/object/cuda_opencl_new.html
10. www.gpgpu.org
11. http://en.wikipedia.org/wiki/GPGPU