University of East Anglia

General Purpose computation on Graphical Processing Units (GPGPU)

Wadud Miah

Research Computing Services Group

Scientific programming (1)

University of East Anglia

- Early scientific codes were mainly sequential and were executed on single core CPUs;
- CPU performance accelerated from 1 MHz and have peaked at around 3.2 GHz;
- CPU features include: L1, L2 and L3 cache. SIMD extensions, instruction units and registers;
- CPU manufacturers adhere to the IEEE 754 floating point number representation to ensure standard computation across architectures.

Scientific programming (2)

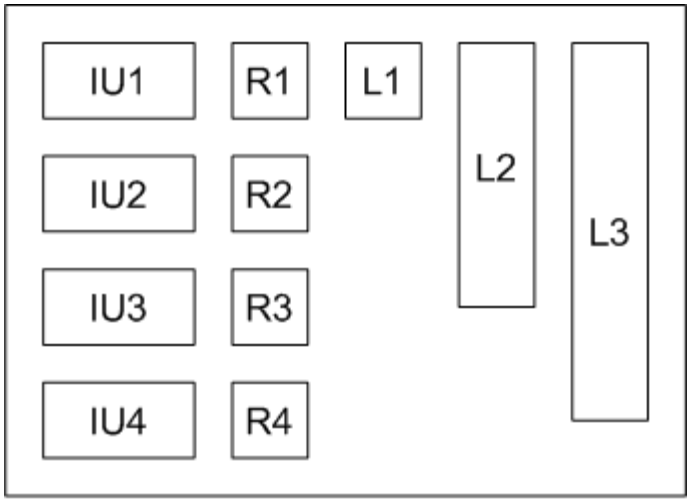
University of East Anglia

- CPU speeds have stagnated and have reached their maximum performance;
- To overcome this, vendors have now introduced multi-core CPUs;
- However, scientists want to a) refine and expand computational domains b) explore challenging scientific problems;
- *Solution: parallel programming on distributed and SMP many-core architectures!*

CPU architecture

University of East Anglia

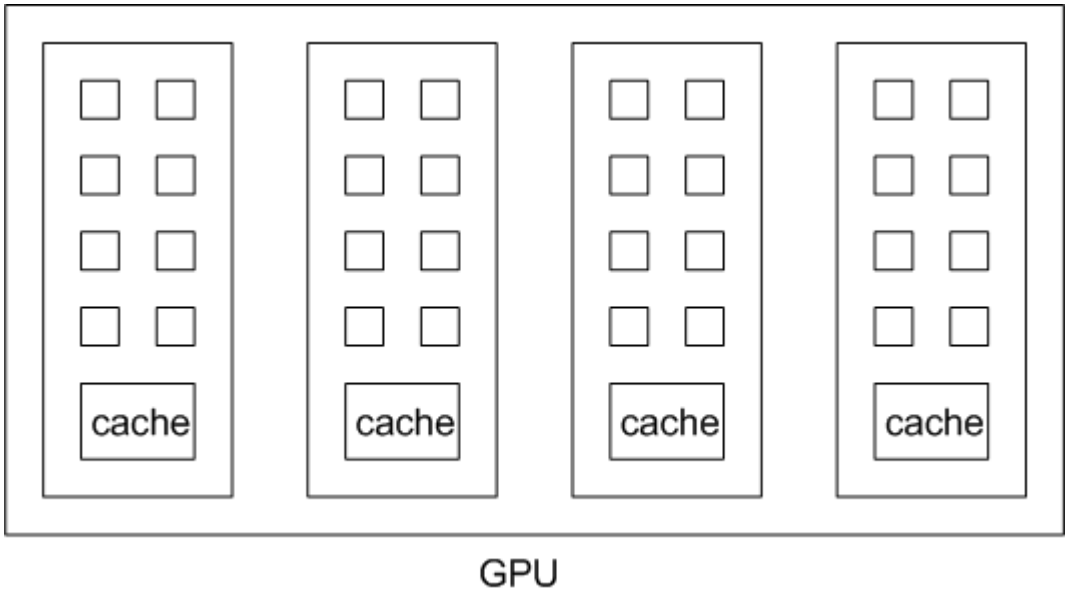
A CPU has a) L1, L2 and L3 cache b) floating point and integer instruction units c) registers d) SIMD instruction units;



GPU architecture

University of East Anglia

GPU cores are “lightweight”, but there are more of them. They operate at ≈ 1.5 GHz also known as “army of snails”!



CUDA: Compute Unified Device Architecture

University of East Anglia

In the early years of GPUs, the OpenGL and DirectX libraries were used as an API to interface with GPUs;

Nvidia then released CUDA as an API and SDK as an interface for Nvidia GPUs;

This is an ANSI C based language with additional extensions – this alleviates the learning curve for developers;

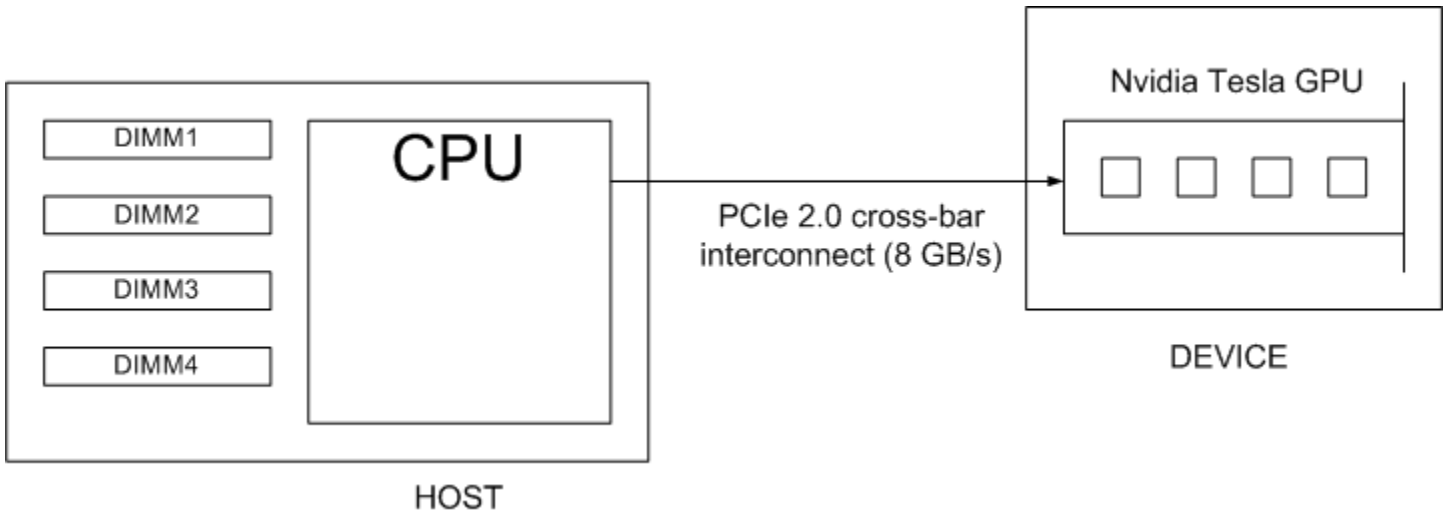
PGI (a compiler company) have developed CUDA for Fortran.

CUDA architecture (1)

University of East Anglia

The CUDA architecture consists of a host and device;

Each have their own respective memory space;



CUDA architecture (2)

University of East Anglia

Data has to be transferred from the host to the device – bottleneck is created at the PCIe 2.0 interconnect (8 GB/s);

The PCIe interconnect has high bandwidth (version 3.0 will have 16 GB/s) but has high latency. Approximately 3 times the latency of RAM;

Ensure large amounts of data are transferred to the device to hide the latency and to increase thread parallelism.

Memory management in CUDA

University of East Anglia

Memory is allocated on host in the usual manner using `malloc();`

Memory is allocated on device using the `cudaMalloc()` call and a pointer to memory is passed as an argument – any attempts to de-reference will result in memory fault!

Data is cleared on host using `free()` and `cudaFree()` on device.

Data management in CUDA (1)

University of East Anglia

Data is transferred from host to device, and device to host using `cudaMemcpy()` and is asynchronous to hide latency;

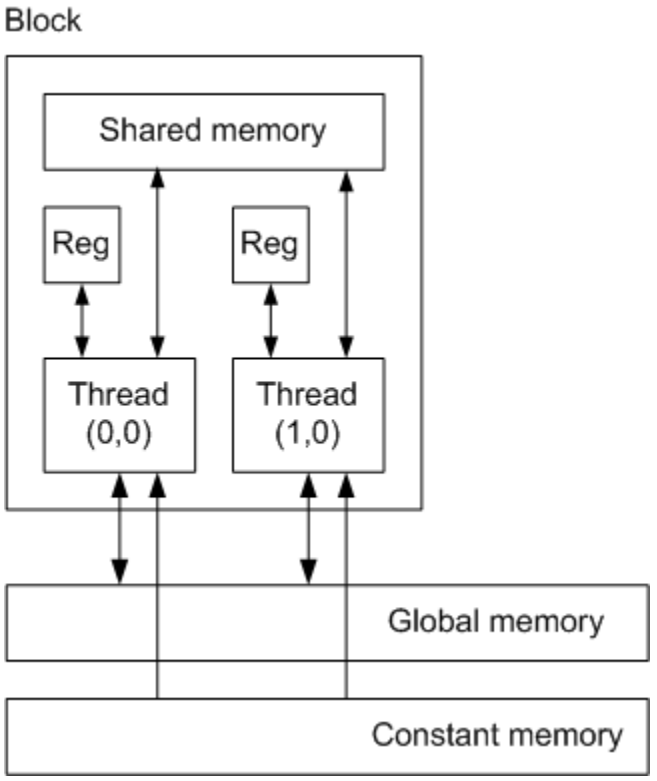
Direction of data movement is also specified using: `cudaMemcpyHostToDevice` and `cudaMemcpyDeviceToHost`;

This function cannot be used to transfer data from one GPU to another directly – to achieve this, move memory to host and then to the other device.

Data management in CUDA (2)

University of East Anglia

`cudaMemcpy()` moves data to the global memory of the device;
This allows all thread blocks to access the data;
If memory as accessed a lot then move it to shared memory.



CUDA thread structure (1)

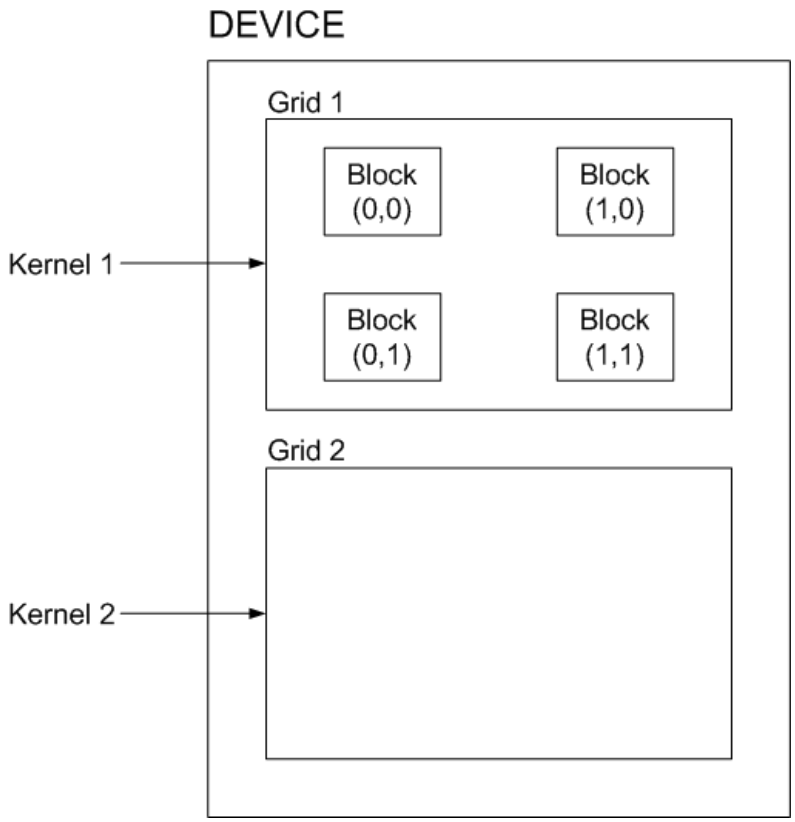
University of East Anglia

Threads are structured in grids and blocks;

Threads in a block can:

- synchronise;
- share data through low latency shared memory;

Threads from different blocks can exchange data via global memory but at a higher latency.

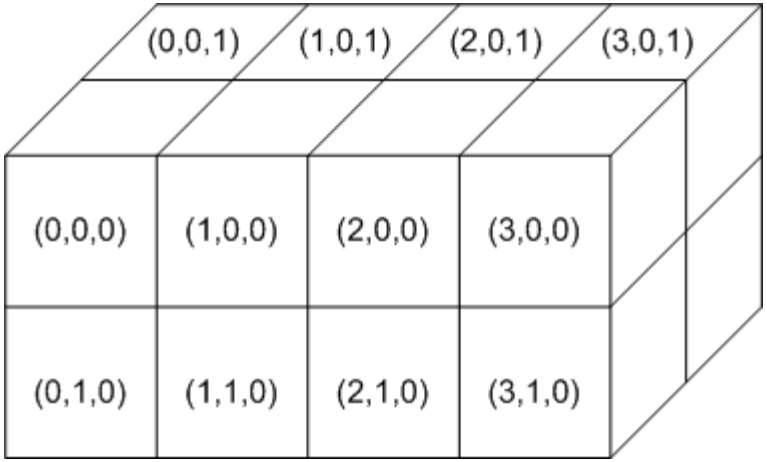


CUDA thread structure (2)

University of East Anglia

A thread block is a three-dimensional set of threads;

It can also be one- or two-dimensional.



CUDA kernel function (1)

University of East Anglia

The function that is to be executed by the GPU is called the “kernel”;

It is defined in a special way and executed likewise using additional extensions:

```
__device__ float deviceFunc();
// executed on device and called by device
__global__ void kernelFunc();
// executed on device and called by host
```

CUDA kernel function (2)

University of East Anglia

```
__host__ float hostFunc();
```

// executed on host and called by host

To launch a kernel from the host first determine a) dimensions of the block b) how many blocks are required, e.g.

```
dim3 dimBlock(32,16);
dim3 dimGrid(1,1);
kernel_func<<<dimGrid,dimBlock>>>
(M,N,P,512);
```

CUDA kernel function (3)

University of East Anglia

Then define the kernel function:

```
      __global__ void kernel(float *M,
float *N, float *P, int size) {
      int tx = threadIdx.x;
      int ty = threadIdx.y;
            int tz = threadIdx.z;
      // execute work on (tx,ty,tz)
     }
```

Index variables

University of East Anglia

Index variables determine which block of data thread is responsible for – this is like the MPI rank;

As well as the three-dimensional `threadIdx` struct, there is also the two-dimensional `blockIdx` struct:

```
blockIdx.x
blockIdx.y
```

This is to determine block of data for thread to operate on.

Block synchronisation

University of East Anglia

Threads in a block can be coordinated;

This is allowed using the barrier synchronisation routine `__syncthreads()` which is called by every thread in the block;

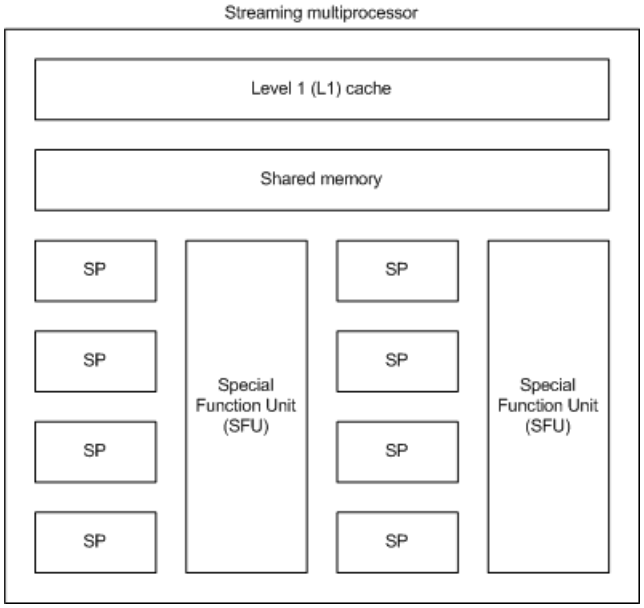
Threads of different blocks cannot be synchronised – only way to achieve this is wait for kernel call to complete.

Streaming Multiprocessors

University of East Anglia

Threads are executed on streaming multiprocessors – 32 cores on each;

Maximum 1024 threads can execute on each SM or 8 blocks.



Thread scheduling - warps

University of East Anglia

Threads are scheduled to be executed on SMs in batches called warps – this is typically 32 threads;

For 512 threads (which is the max) in a block, this is divided into 512/32 = 16 warps;

The large number of warps is required to hide latency within the host/GPU – as one warp is waiting for data, another warp can be executed on an SM.

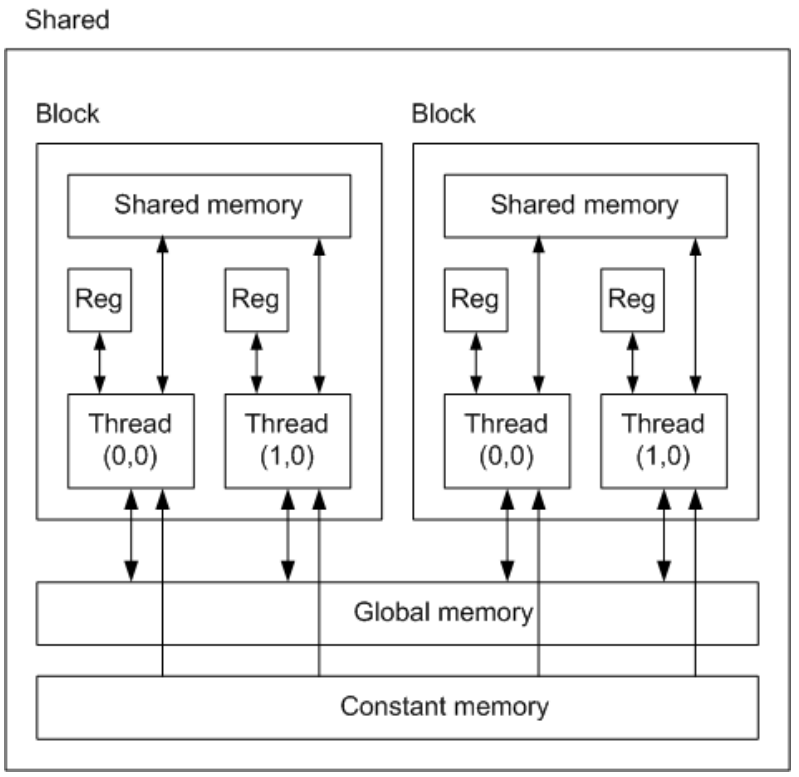
Nvidia Tesla memory hierarchy (1)

University of East Anglia

The Nvidia Tesla memory hierarchy is similar to that of CPUs, but more complex;

Bottleneck in exists at the global memory level;

To hide the latency, ensure data locality, e.g. use constant and shared memory or registers.



Nvidia Tesla memory hierarchy (2)

University of East Anglia

CUDA variable type qualifiers

Variable declaration	Memory type	Scope	Lifetime
automatic scalar	register	thread	kernel
automatic array	global	thread	kernel
__shared__	shared	block	kernel
__device__	global	grid	application
__constant__	constant	grid	application

CUDA libraries and interfaces

University of East Anglia

Many numerical libraries have been ported to CUDA;

- CUBLAS – Basic Linear Algebra Subprograms;
- CUFFT – Fast Fourier Transform;
- LAPACK – Numerical Linear Algebra.

MATLAB CUDA toolbox;

Maple interface with CUDA;

Mathematica interface with CUDA;

PyCUDA – Python interface to CUDA.

Nvidia Fermi Architecture (1)

University of East Anglia

Nvidia have released a new GPU architecture known as “Fermi” which is a significant upgrade from the GT80 and GT200 GPUs;

It is evolutionary and revolutionary, and provides greater performance and maps closer to programming models;

The features of the Fermi architecture include:

- Greater number of CUDA cores;
- Massive increase in double precision performance (≈ half single precision performance);

Nvidia Fermi Architecture (2)

University of East Anglia

The features of the Fermi architecture include:

- Twice the number of special function units (e.g. sin, cos, exp);
- Dual warp scheduler – two warps are dispatched to each SM;
- Greater amount of shared memory for each thread block;
- Configurable L1 cache and L2 cache availability;
- ECC memory support for reliable computing – include CRC checking over the PCIe interconnect;

Nvidia Fermi Architecture (3)

University of East Anglia

The features of the Fermi architecture include:

- Greater load/store address width (64 bit) – transfer more data;
- Global memory address namespace for C++ programs;
- Full IEEE-754-2008 implementation for greater accuracy;
- Better branch prediction for streamlining codes;
- Concurrent kernel execution to pipeline GPUs for better efficiency.

Conclusion (1)

University of East Anglia

GPUs provide a many-core parallel device;

Many codes have been ported to GPUs and work is still on going;

The CUDA SDK allows scientists to develop their own codes for execution on GPUs;

If the CUDA SDK is too complex, investigate whether libraries or codes exist in CUDA;

The many-core paradigm is the future of scientific computing!

Conclusion (2)

University of East Anglia

The Research Computing Services Group has a host with two C2050 cards with following specification:

- 515 GFLOP/s double precision or 1000 GFLOPs single precision;
- 3 GB of ECC GDDR5 memory to ensure data integrity;
- 448 Tesla cores operating at 1.15 GHz;
- 144 GB/s of memory bandwidth.

Contact us at `hpc.admin@uea.ac.uk` for further information.

References

http://www.nvidia.com/object/cuda_home_new.html

“Programming Massively Parallel Processors”, D. Kirk and W. Hwu;

http://www.nvidia.com/object/tesla_computing_solutions.html