5th Undergraduate Summer School

High Performance Computing , Research Methodologies & Simulation's

SHIVAM GUPTA

ME(2015-17), CSA

HPC LAB, SERC

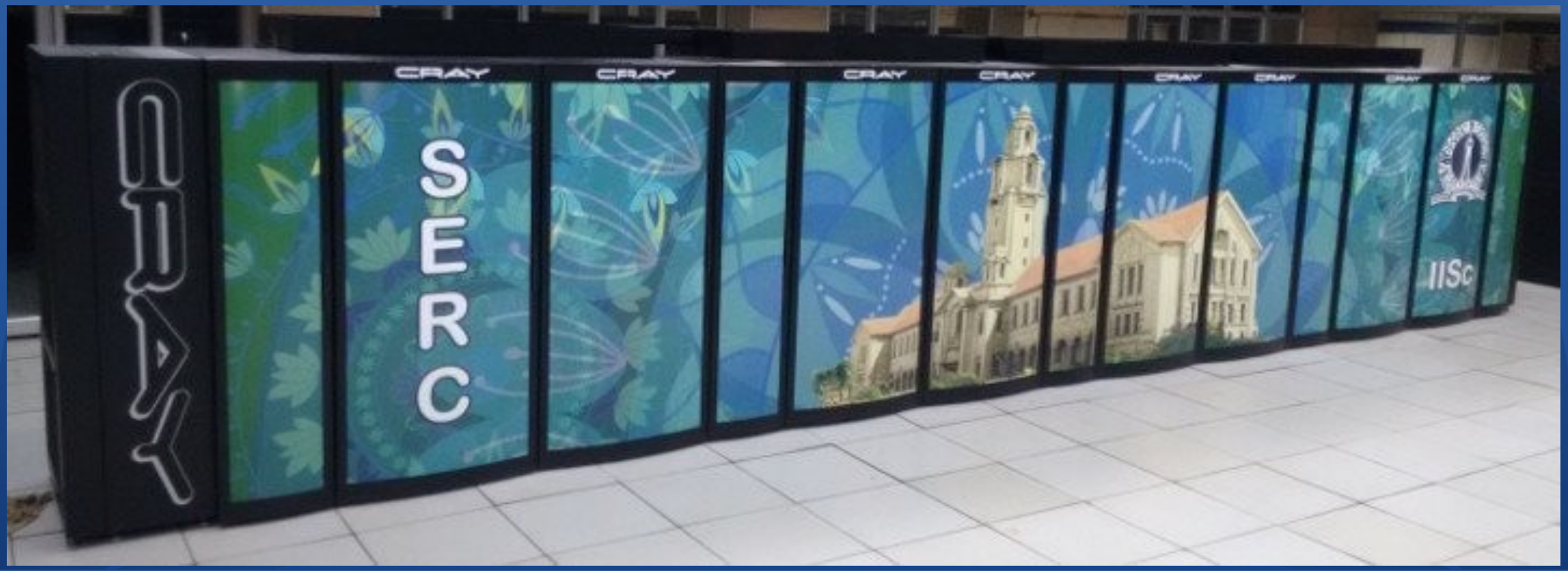
LAB CONVENER

Prof. R. Govindarajan
(SERC- CHAIRMAN)

INDIAN INSTITUTE OF SCIENCE, BANGALORE

What is HPC?

"High Performance Computing(HPC) is the use of supercomputers and parallel processing techniques for solving advance problems and performing Research activites through Computer modelling, Simulations and analysis efficiently,reliably and quickly."



Need & Area of HPC

Evolve to meet increasing demands of Processing Speeds.

Brings together several technologies such as computer architecture, algorithms, programs and electronics, and system software under a single canopy to solve advanced problems effectively and quickly.

HPC technology is implemented in multidisciplinary areas including:

- Biosciences
- Geographical data
- Oil and gas industry modeling
- Electronic design automation
- Climate modeling
- Media and entertainment

Parallel Computation

Parallel computing is a type of computing architecture in which several processors execute or process an application or computation simultaneously.

Helps in performing large computations by dividing the workload between more than one processor, all of which work through the computation at same time.

Parallel computing is also known as parallel processing.

Increase the available computation power for faster application processing or task resolution.

ACCELERATORS

“Accelerators are computing components containing functional units, together with memory and control systems that can be easily added to computers to speed up portions of applications”.

Types

General Purpose Graphical Processing Units (GPGPUs)

- Field Programmable Gate Arrays (FPGAs) boards
- ClearSpeed’s floating-point boards
- IBM’s Cell processors

WHAT IS GPU-ACCELERATED COMPUTING?

Is use of a graphics processing unit (GPU) together with a CPU to accelerate deep learning, analytics, and engineering applications.

Play a huge role in accelerating applications in platforms ranging from artificial intelligence to cars, drones, and robots.

A GPU (Graphics Processing Unit) is a processor attached to a graphics card dedicated to calculating floating point operations.

They are mainly used for playing 3D games or high-end 3D rendering.

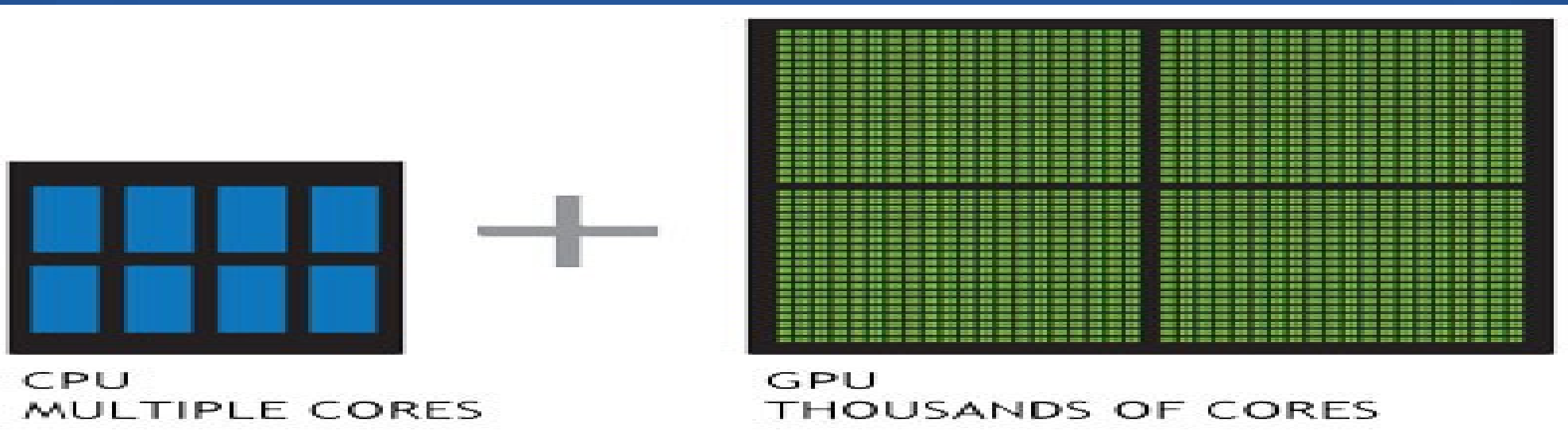
Modern GPU As: GPGPU

The model for GPU computing is to use a CPU and GPU together in a heterogeneous co-processing computing model.

GPU now widely used as General purpose Graphic Processing Unit (GPGPU).

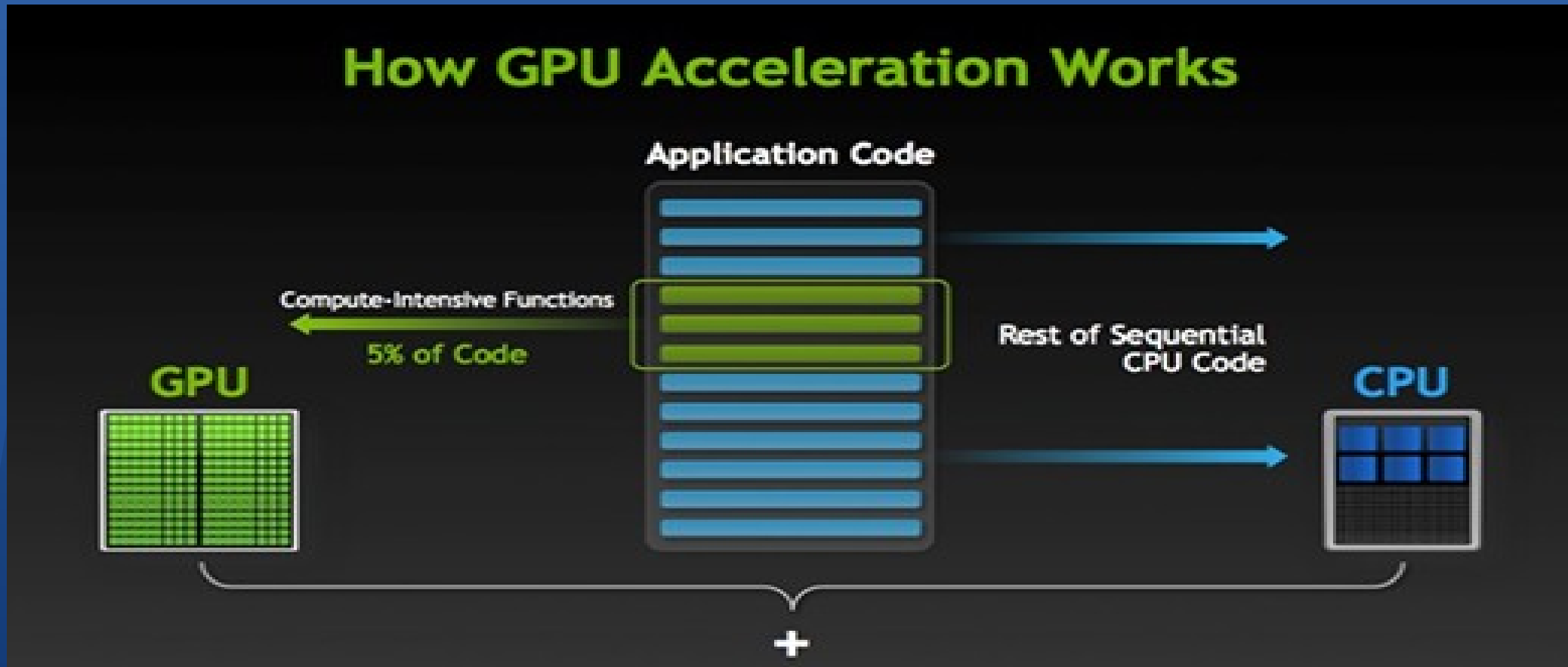
The sequential part of the application runs on the CPU and the computationally-Intensive part accelerated by the GPU and the computationally-intensive part accelerated by the GPU.

From the user's perspective, the application just runs faster because it is using the high-performance of the GPU to boost performance.



GPUs ACCELERATE SOFTWARE APPLICATIONS

GPU-accelerated computing offloads compute-intensive portions of the application to the GPU,while the remainder of the code still runs on the CPU.



NVIDIA GPU COMPUTING :GPGPU

NVIDIA introduced two key technologies—the G80 unified graphics and compute architecture (first introduced in GeForce 8800®, Quadro FX 5600®, and Tesla C870® GPUs).

CUDA, a software and hardware architecture that enabled the GPU to be programmed witha variety of high level programming languages.

The programmer could now write C programs with CUDA extensions and target a general purpose, massively parallel processor.

CUDA® is a parallel computing platform and programming model invented by NVIDIA. It enables dramatic increases in computing performance by harnessing the power of the graphics processing unit (GPU).

CUDA Toolkit

The NVIDIA® CUDA® Toolkit provides a development environment for creating high performance GPU-accelerated applications.

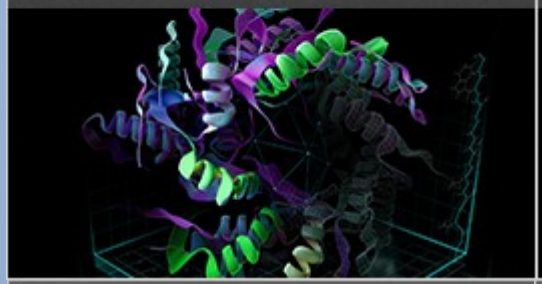
The toolkit includes GPU-accelerated libraries, debugging and optimization tools, a C/C++ compiler and a runtime library to deploy your application.

GPU-accelerated CUDA libraries enable drop-in acceleration across multiple domains such as linear algebra, image and video processing, deep learning and graph analytics.

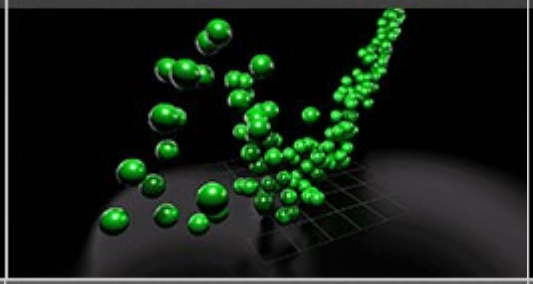
The CUDA compute platform extends from the 1000s of general purpose compute processors featured in our GPU's compute architecture, parallel computing extensions to many popular languages, powerful drop-in accelerated libraries to turn key applications and cloud based compute appliances.

CUDA Acceleration For All Domains

BIOINFORMATICS

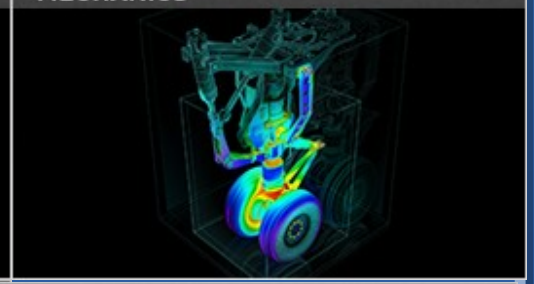


COMPUTATIONAL CHEMISTRY



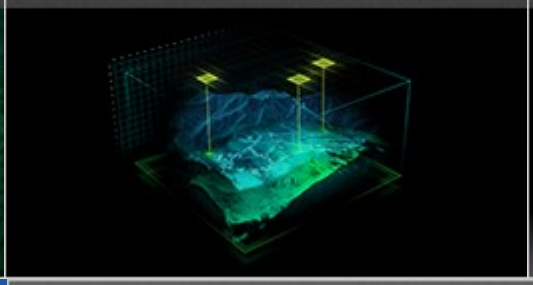
COMPUTATIONAL FLUID DYNAMICS

COMPUTATIONAL STRUCTURAL MECHANICS

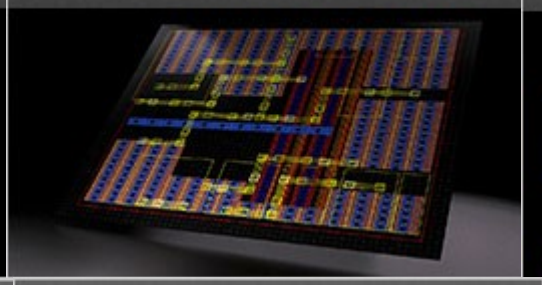


DATA SCIENCE

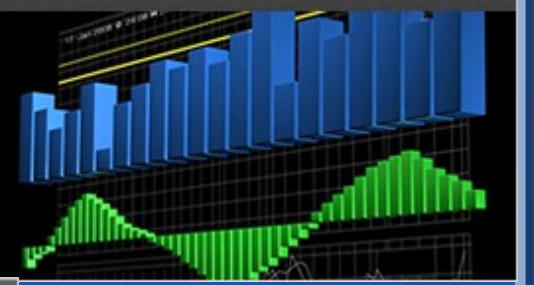
DEFENSE



ELECTRONIC DESIGN AUTOMATION

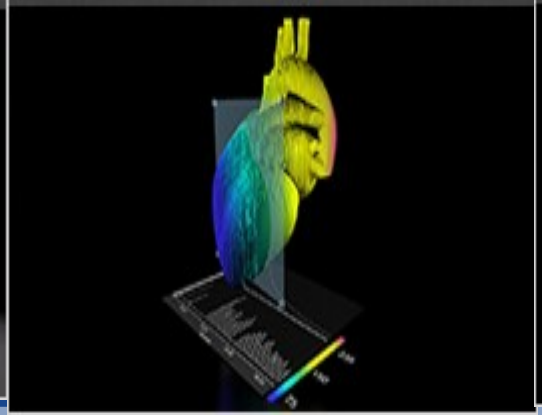


COMPUTATIONAL FINANCE

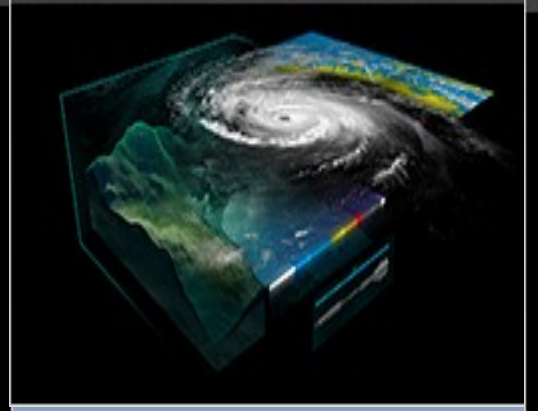


IMAGING & COMPUTER VISION

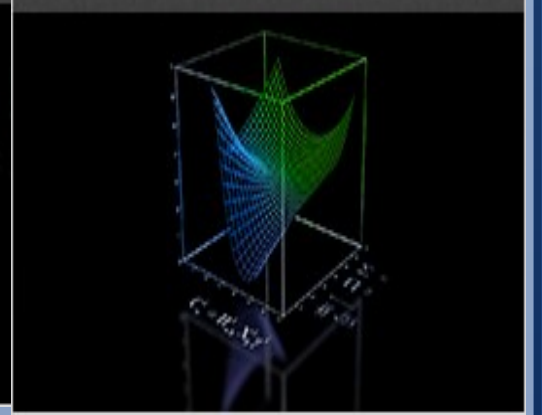
MEDICAL IMAGING



WEATHER AND CLIMATE



NUMERICAL ANALYTICS



The G80 Architecture

NVIDIA's GeForce 8800 was the product that gave birth to the new GPU Computing model.

Introduced in November 2006, the G80 based GeForce 8800 brought several key innovations to GPU Computing:

G80 was the first GPU to support C, allowing programmers to use the power of the GPU without having to learn a new programming language.

G80 introduced the single-instruction multiple-thread (SIMT) execution model where multiple independent threads execute concurrently using a single instruction.

G80 introduced shared memory and barrier synchronization for inter-thread communication.

NVIDIA's Next Generation "Fermi"

Third Generation Streaming Multiprocessor (SM) over 32 CUDA cores per SM,

64 KB of RAM with a configurable partitioning of shared memory and L1 cache

Dual Warp Scheduler simultaneously schedules and dispatches instructions from two independent warps .

Unified Address Space with Full C++ Support

Full IEEE 754-2008 32-bit and 64-bit precision

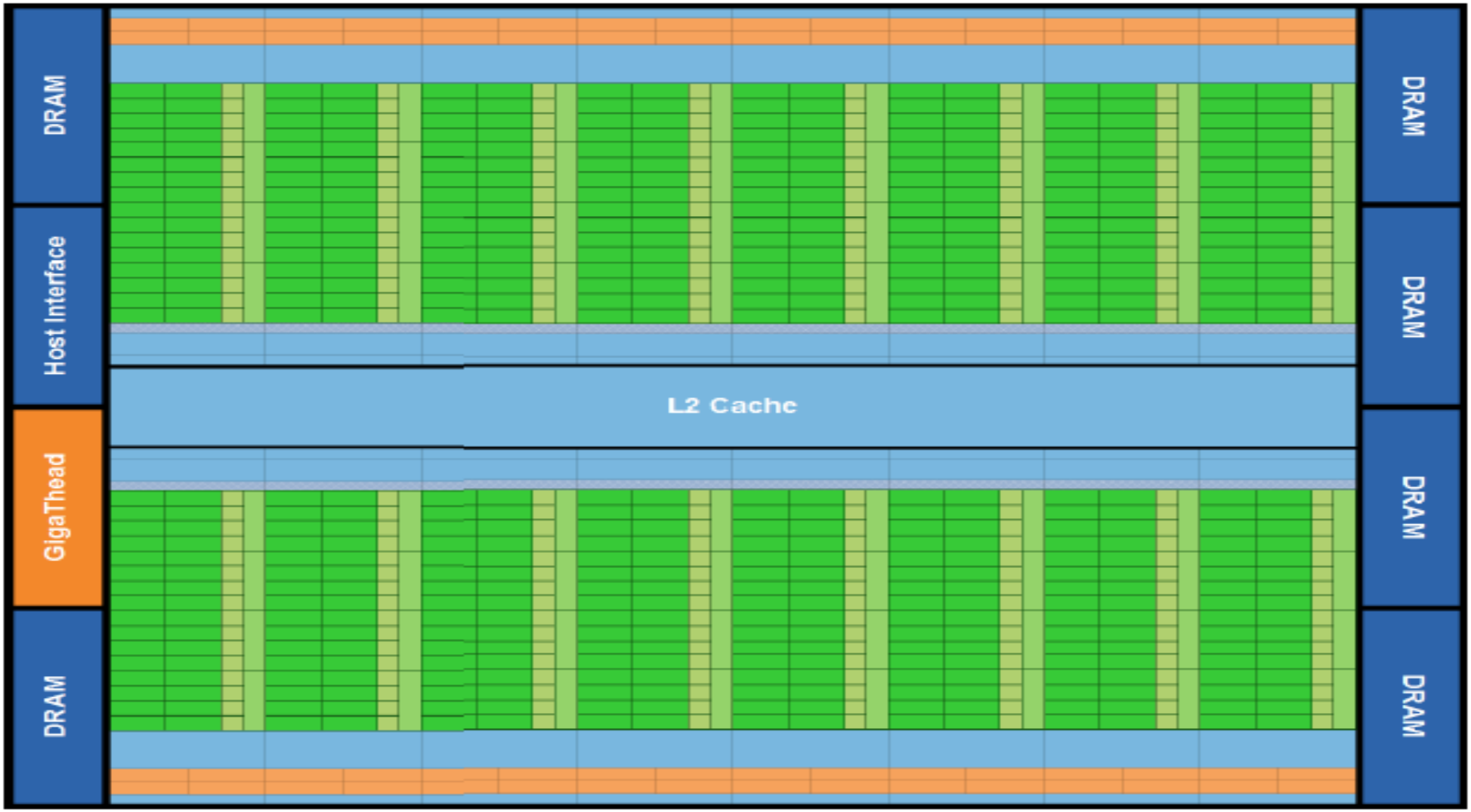
NVIDIA Parallel DataCacheTM hierarchy with Configurable L1 and Unified L2 Caches

10x faster application context switching

Concurrent kernel execution

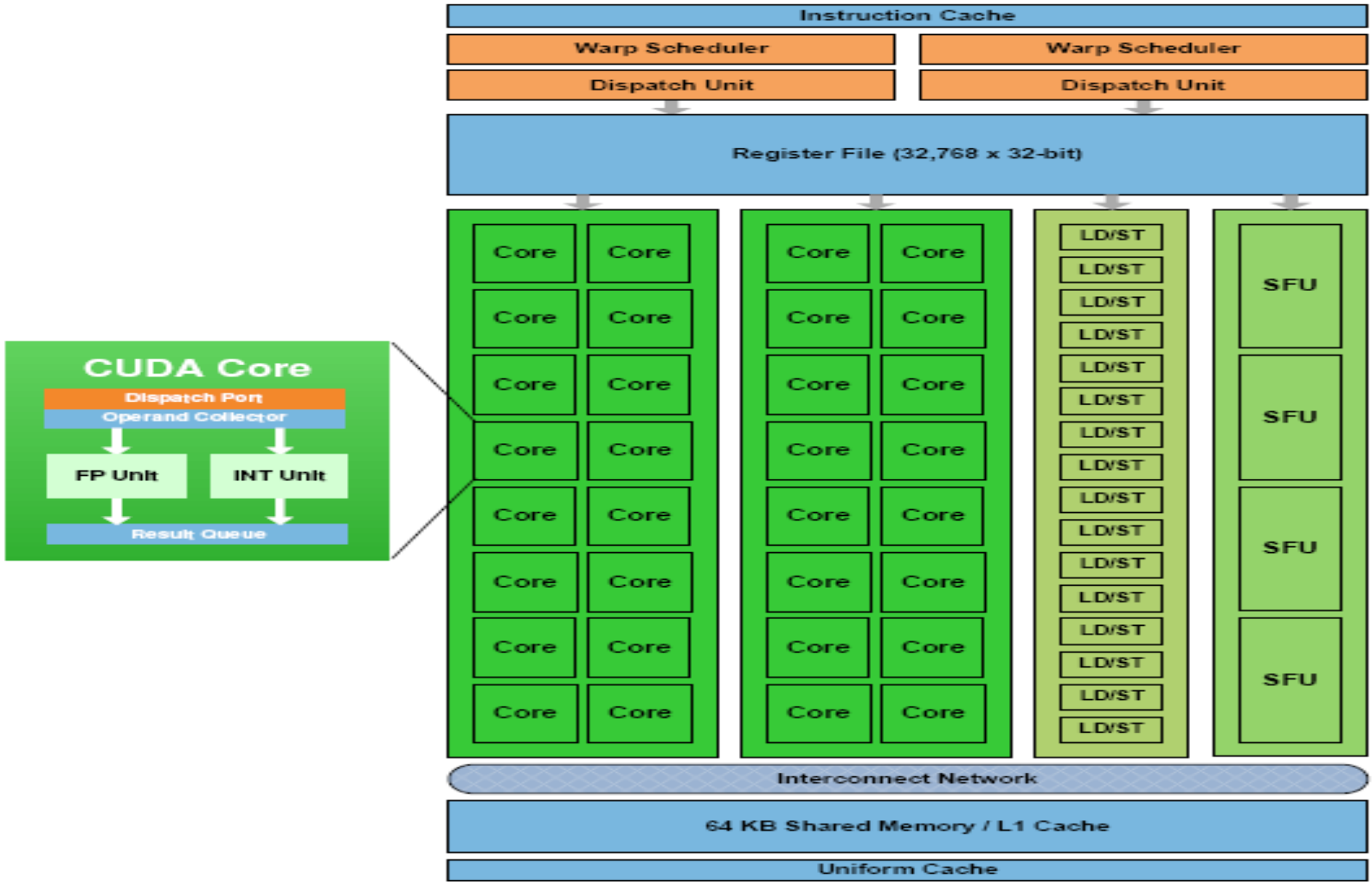
Out of Order thread block execution

Overview of Fermi's Architecture



Fermi's 16 SM are positioned around a common L2 cache. Each SM is a vertical rectangular strip that contain an orange portion (scheduler and dispatch), a green portion (execution units), and light blue portions (register file and L1 cache).

Fermi's Streaming Multiprocessor(SM)



Fermi Streaming Multiprocessor (SM)

CUDA Kernels

CUDA C extends C by allowing the programmer to define C functions, called Kernels.

In computing, a compute kernel is a routine compiled for high throughput accelerators (such as GPUs).

Correspond to inner loops when implementing algorithms in traditional Languages.

A kernel executes in parallel across a set of parallel threads.

when called, are executed N times in parallel by N different CUDA threads, as opposed to only once like regular C functions.

A kernel is defined using the __global__ declaration specifier

the number of CUDA threads that execute that kernel for a given kernel call is specified using a new <<<...>>>execution configuration syntax

Kernel Definition

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
    ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C);
    ...
}
```

Each of the N threads that execute VecAdd() performs one pair-wise addition.

Each thread that executes the kernel is given a unique thread ID that is accessible within the kernel through the built-in threadIdx variable.

Thread Hierarchy

threadIdx is a 3-component vector, so that threads can be identified using a one-dimensional, two-dimensional, or three-dimensional thread index, forming a one-dimensional,two-dimensional, or three-dimensional block of threads, called a **thread block.**

For a one-dimensional block, they are the same.

two-dimensional block of size (Dx, Dy),the thread ID of a thread of index (x, y) is (x + y Dx);

for a three-dimensional block of size (Dx, Dy, Dz), the thread ID of a thread of index (x, y, z) is (x + y Dx + z Dx Dy).

There is a limit to the number of threads per block, since all threads of a block are expected to reside on the same processor core and must share the limited memory resources of that core.

On current GPUs, a thread block may contain up to 1024 threads.

Grids Of Thread Blocks

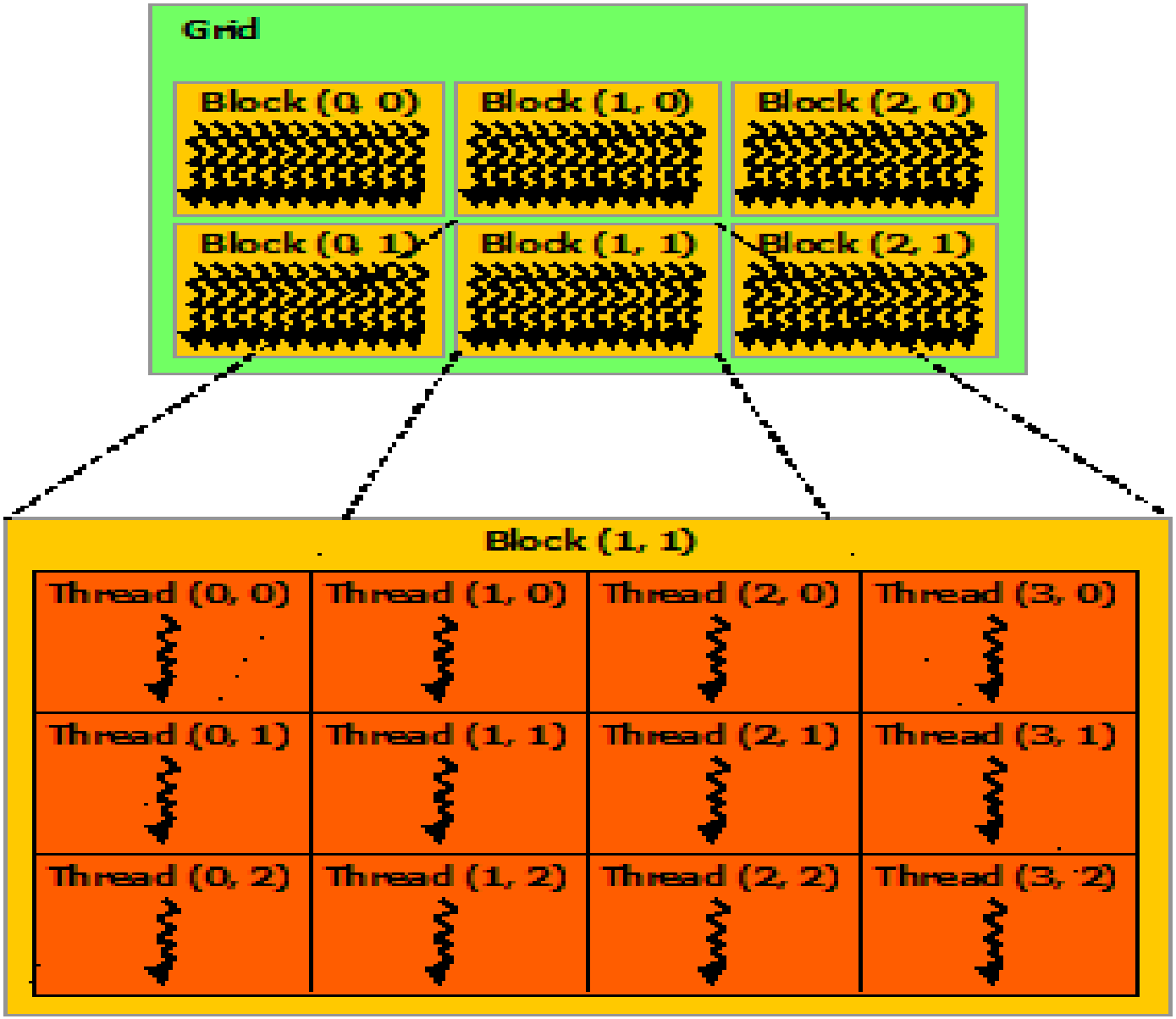
Blocks are organized into a one-dimensional, two-dimensional, or three-dimensional grid of thread blocks.

The number of thread blocks in a grid is usually dictated by the size of the data being processed.

A grid is an array of thread blocks that execute the same kernel, read inputs from global memory, write results to global memory, and synchronize between dependent kernel calls.

Grids of thread blocks share results in Global Memory space after kernel-wide Global synchronization.

Threads Blocks and Grids

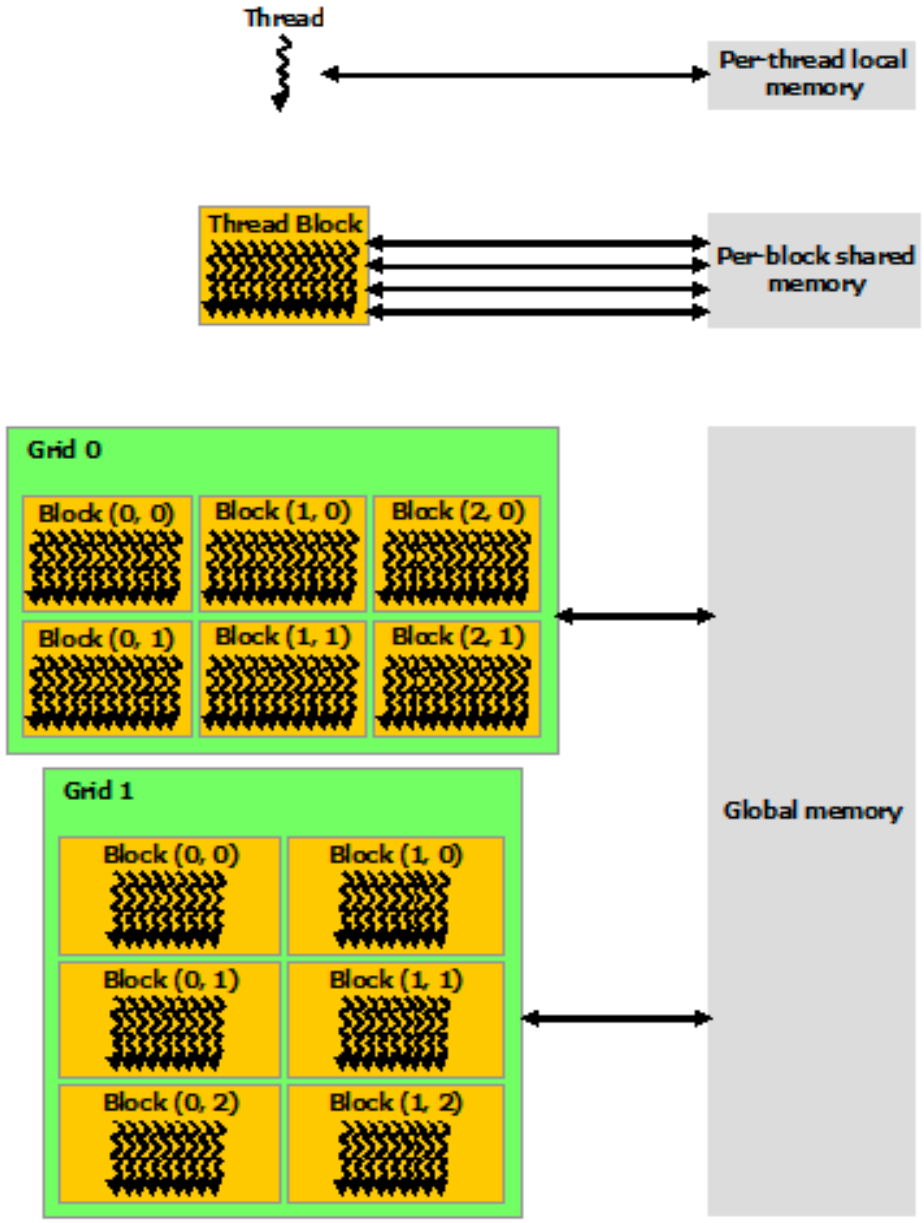


Matrix Multiplication Example

```
// Kernel definition
__global__ void MatAdd(float A[N][N], float B[N][N],
float C[N][N])
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    if (i < N && j < N)
        C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    ...
    // Kernel invocation
    dim3 threadsPerBlock(16, 16);
    dim3 numBlocks(N / threadsPerBlock.x, N / threadsPerBlock.y);
    MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
    ...
}
```

The dimension of the thread block is accessible within the kernel through the built-in blockDim variable.

Memory Hierarchy

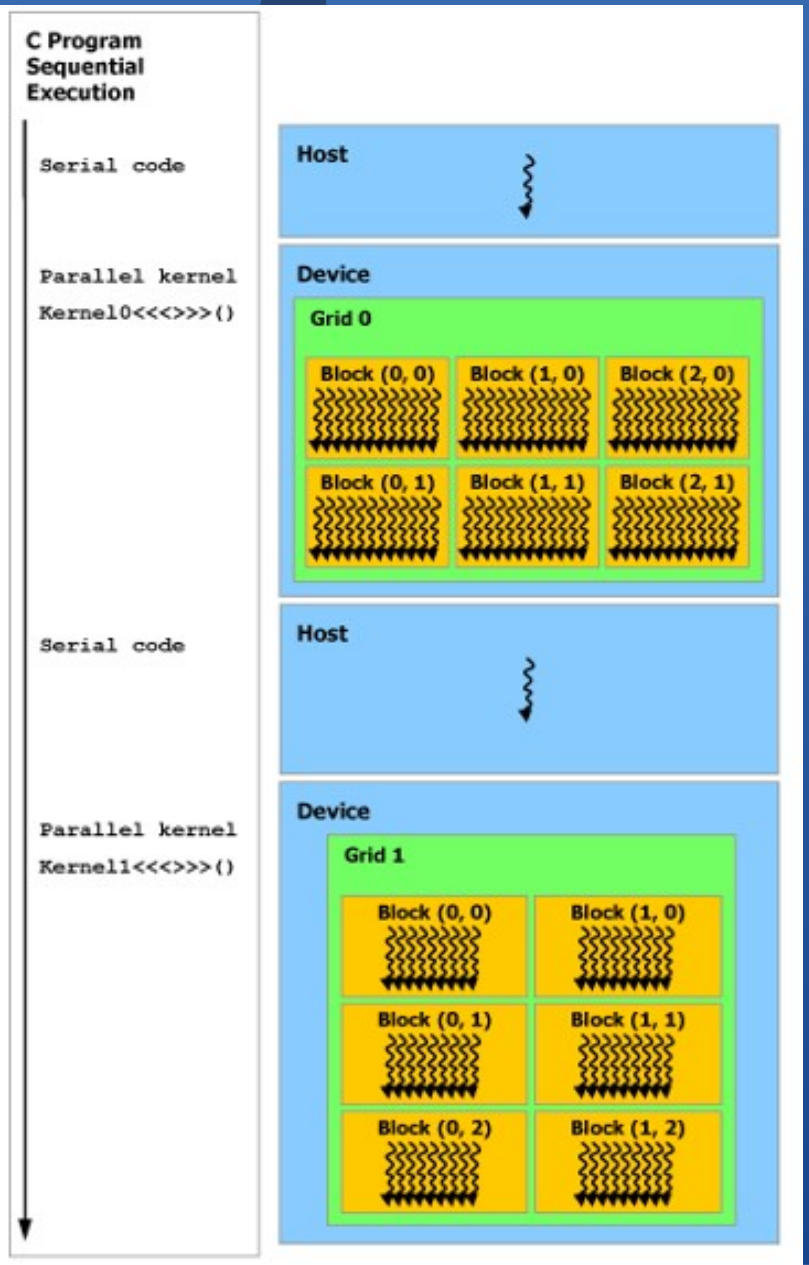


Eachthread has a per-thread private memory space used for registerspills, function calls, and Cautomatic array Variables.

Eachthread block has a per-Block shared memory space used for inter-thread communication, data sharing, and result sharing in parallel algorithms.

Grids of thread blocks share results in Global Memory space after kernel-wide global synchronization

Heterogenous Programming



The CUDA programming model assumes that the CUDA threads execute on a physically separate device that operates as a coprocessor to the host running the C program.

When the kernels execute on a GPU and the rest of the C program executes on a CPU.

Both the host and the device maintain their own separate memory spaces in DRAM, referred to as host memory and device memory

Aprogram manages the global, constant, and texture memory spaces visible to kernels through calls to the CUDA runtime.

NVCC Compilation

Kernels must be compiled into binary code by **nvcc** to execute on the device.

nvcc is a compiler driver that simplifies the process of compiling C

Read more at: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.

Heterogenous Computing CUDA Program

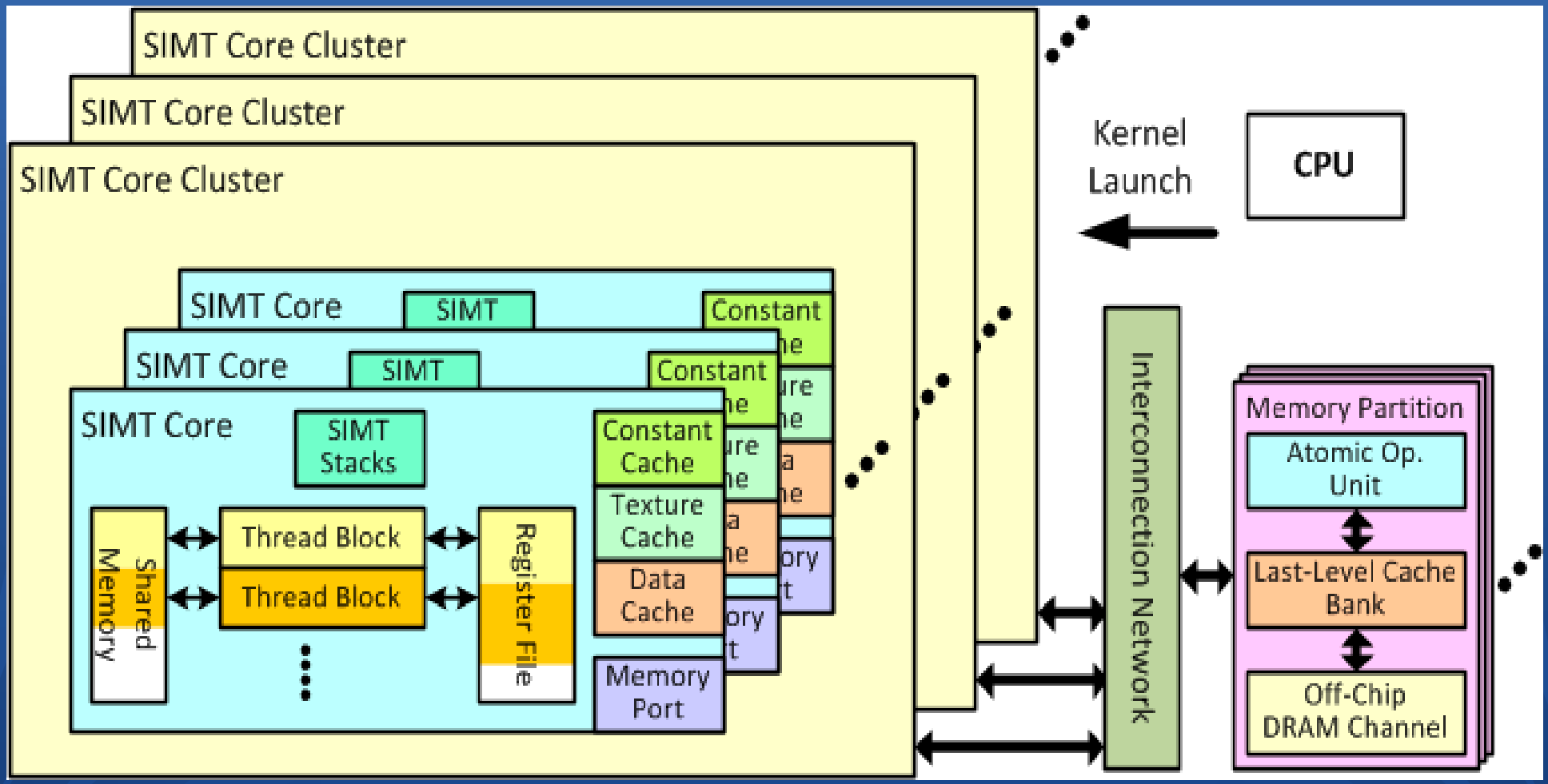
```
// Device code
__global__ void VecAdd(float* A, float* B, float* C, int N)
{ int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < N)
        C[i] = A[i] + B[i];}
  // Host code
int main()
{
    int N = ...;
    size_t size = N * sizeof(float);
// Allocate input vectors h_A and h_B in host memory
    float* h_A = (float*)malloc(size);
    float* h_B = (float*)malloc(size);
 // Initialize input vectors
    ...
 // Allocate vectors in device memory
    float* d_A;
    cudaMalloc(&d_A, size);
    float* d_B;
    cudaMalloc(&d_B, size);
    float* d_C;
    cudaMalloc(&d_C, size);
// Free host memory
    ...}
```

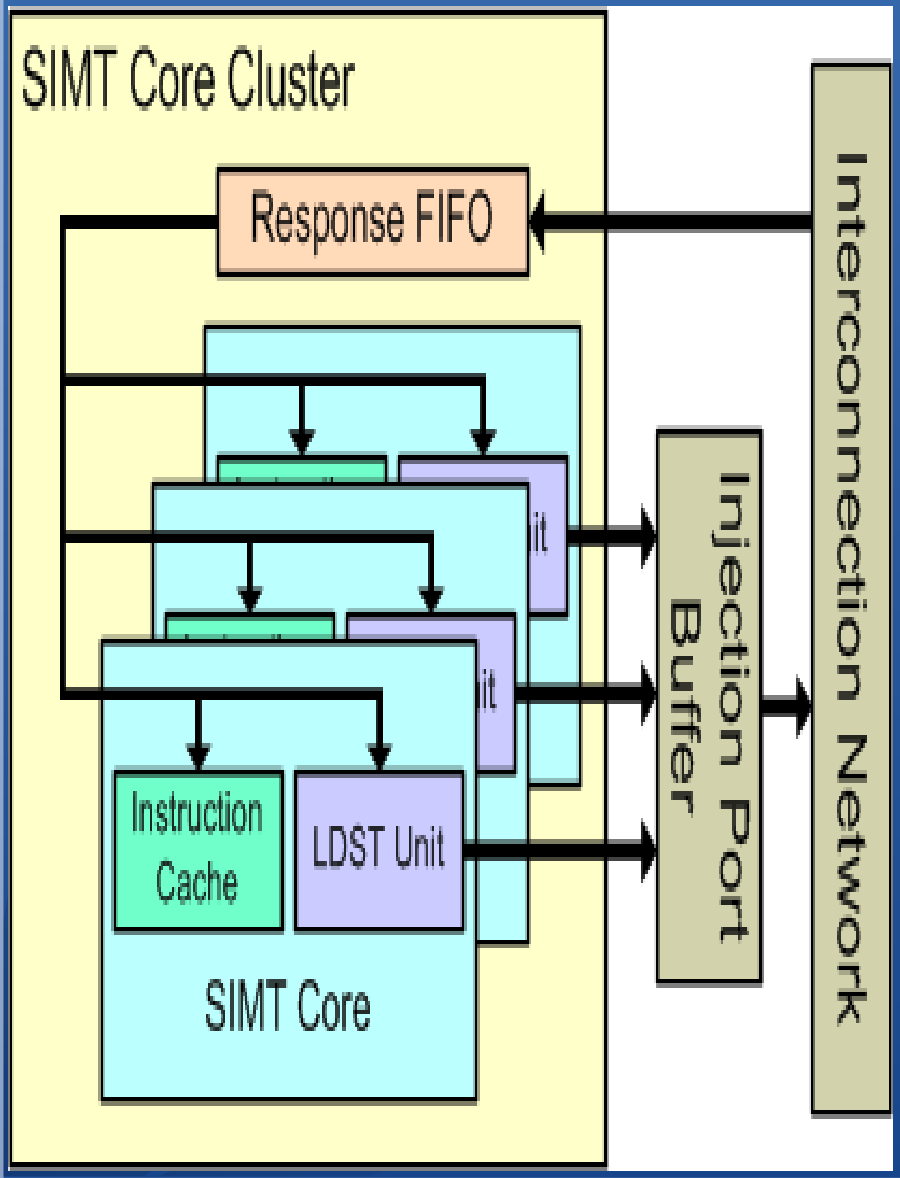
Program Continued....

```
// Copy vectors from host memory to device memory
   cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
   cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);

   // Invoke kernel
   int threadsPerBlock = 256;
   int blocksPerGrid =
         (N + threadsPerBlock - 1) / threadsPerBlock;
   VecAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, N);

   // Copy result from device memory to host memory
   // h_C contains the result in host memory
   cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);

   // Free device memory
   cudaFree(d_A);
   cudaFree(d_B);
   cudaFree(d_C);
```

Research Methodologies and Simulations

GPGPU-Sim





The SIMT Cores are grouped into SIMT Core Clusters.

The SIMT Cores in a SIMT Core Cluster share a common port to the interconnection network .

Each SIMT Core Cluster has a single response FIFO which holds the packets ejected from the interconnection network.

The packets are directed to either a SIMT Core's instruction cache.

The packets exit in FIFO fashion.

Rodinia:Heterogoenous Benchmark Suite

Rodinia, a benchmark suite for heterogeneous computing

To help architects study emerging platforms such as GPUs.

Rodinia includes applications and kernels which target multi-core CPU and GPU platforms.

Rodinia Application/Kernel
Kmeans
Needleman-Wunsh
Hotspot
Back-Propagation
Breadth First Search
Stream Cluster
Similarity Score

DEMO-TIME & Simulations Analysis

Run Demo Hello World Program To see Cpu and GPU Work.

Analyse Result From Log File Helps various Researchers to design GPU.

Show demo Running Rodinia Benchmark application on actual GPU Fermi and running Same benchmark To see simulation through GPGPU-Sim..