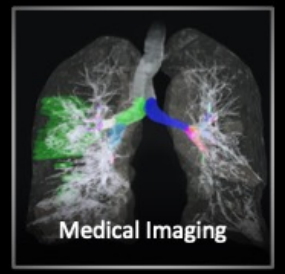
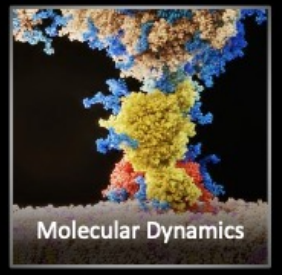
INTRODUCTION TO GPU COMPUTING

N-WAYS GPU BOOTCAMP





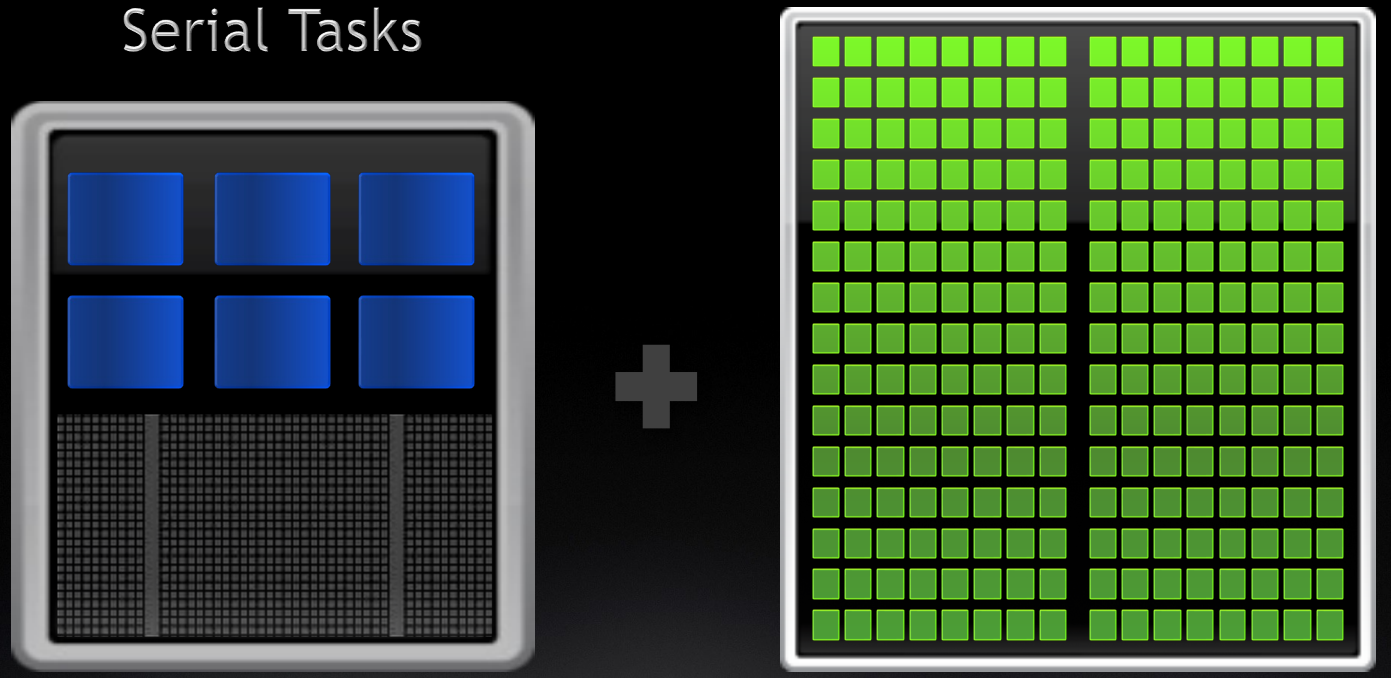
Universe of GPU Computing

GPU PROGRAMMING
(FOUNDATIONS)

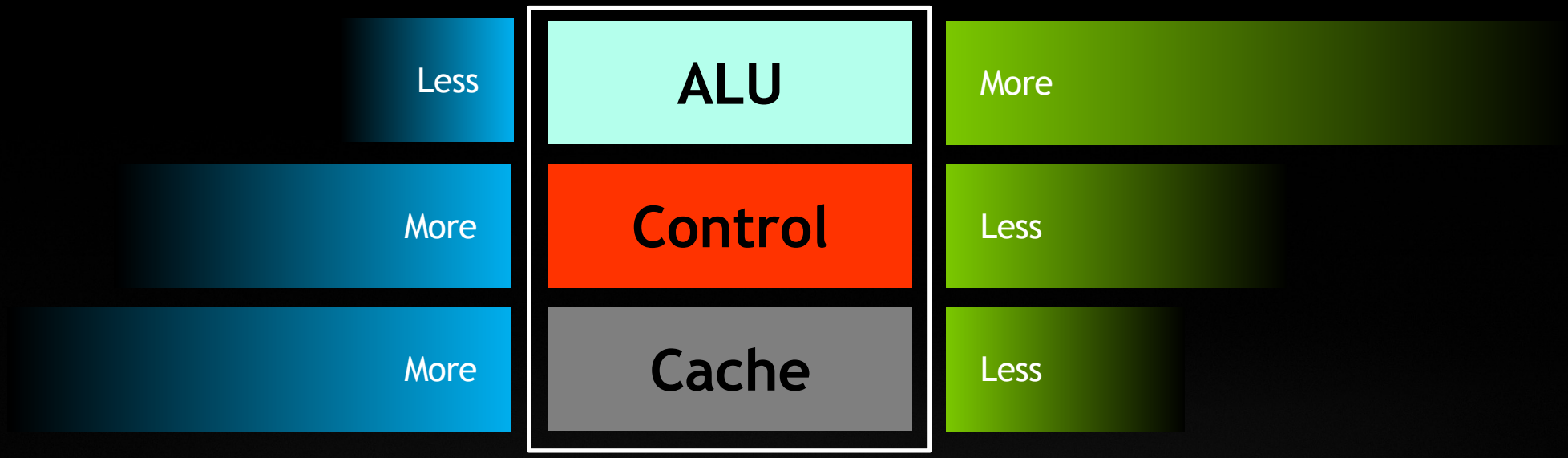
FUNDAMENTALLY DIFFERENT

CPU
Optimized for
Serial Tasks

GPU Accelerator
Optimized for
Parallel Tasks

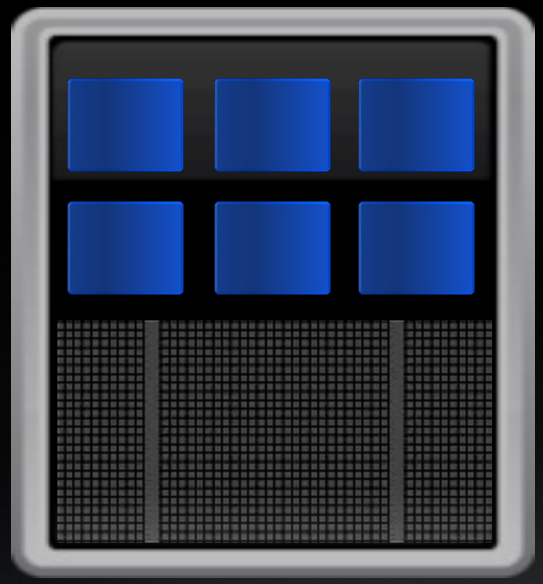


SILICON BUDGET



CPU IS A LATENCY REDUCING ARCHITECTURE

CPU
Optimized for
Serial Tasks



CPU Strengths

- Very large main memory
- Very fast clock speeds
- Latency optimized via large caches
- Small number of threads can run very quickly

CPU Weaknesses

- Relatively low memory bandwidth
- Cache misses very costly
- Low performance/watt

GPU IS ALL ABOUT HIDING LATENCY

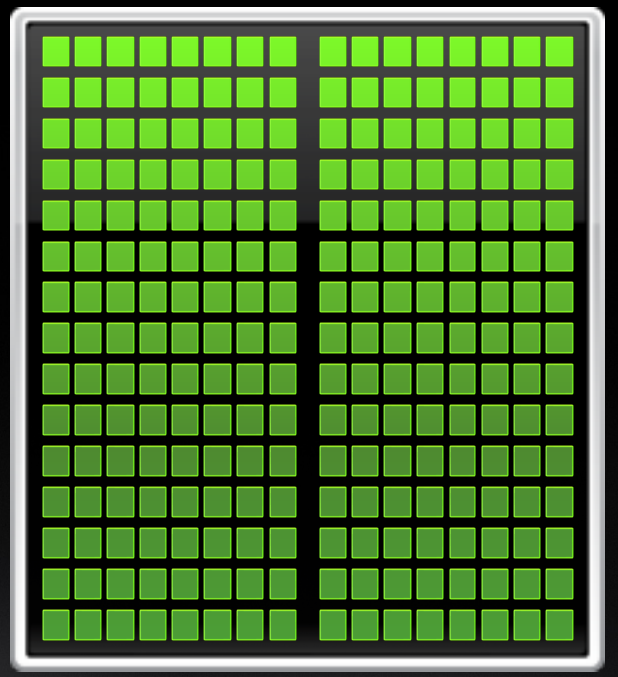
GPU Strengths

- High bandwidth main memory
- Significantly more compute resources
- Latency tolerant via parallelism
- High throughput
- High performance/watt

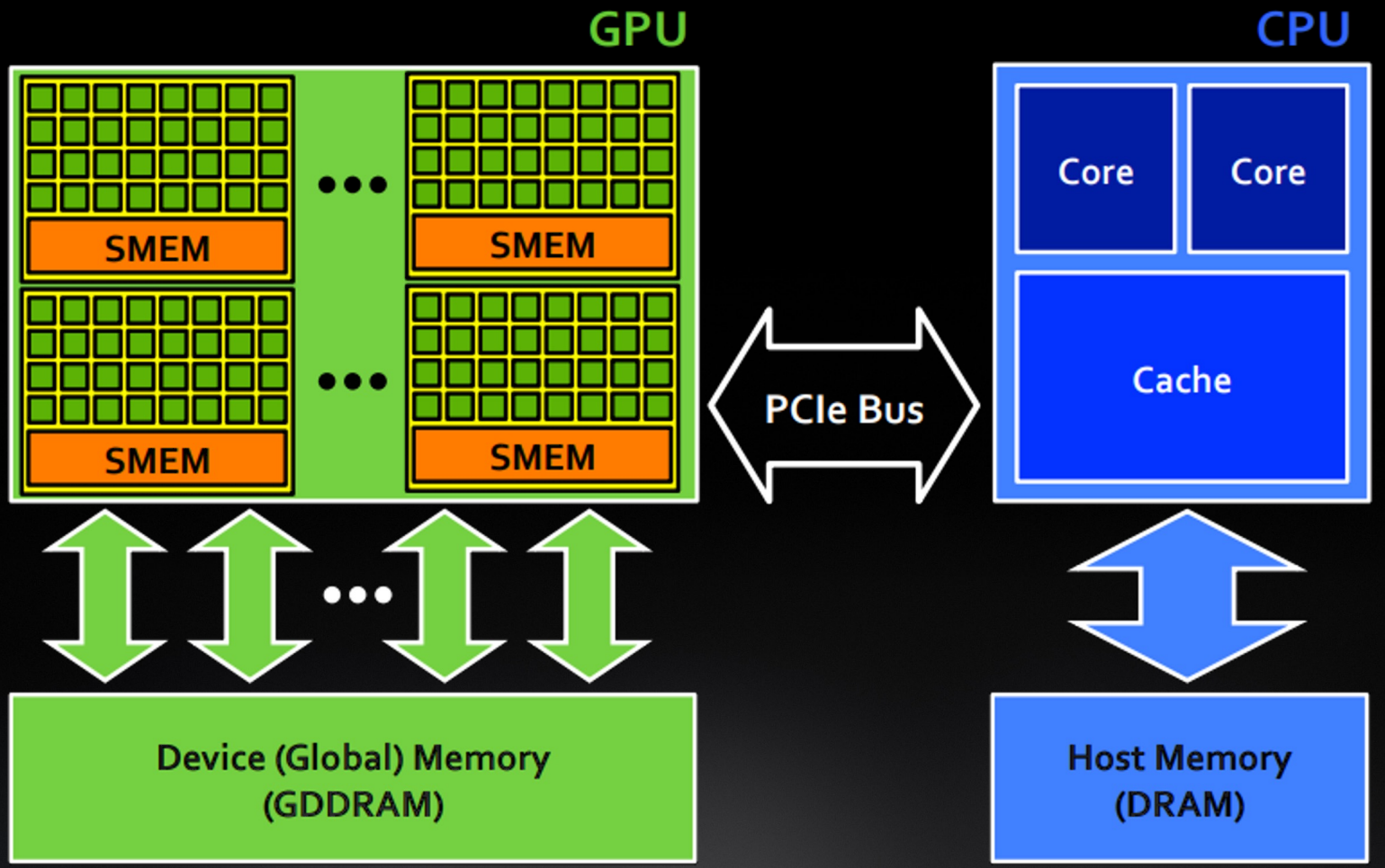
GPU Weaknesses

- Relatively low memory capacity
- Low per-thread performance

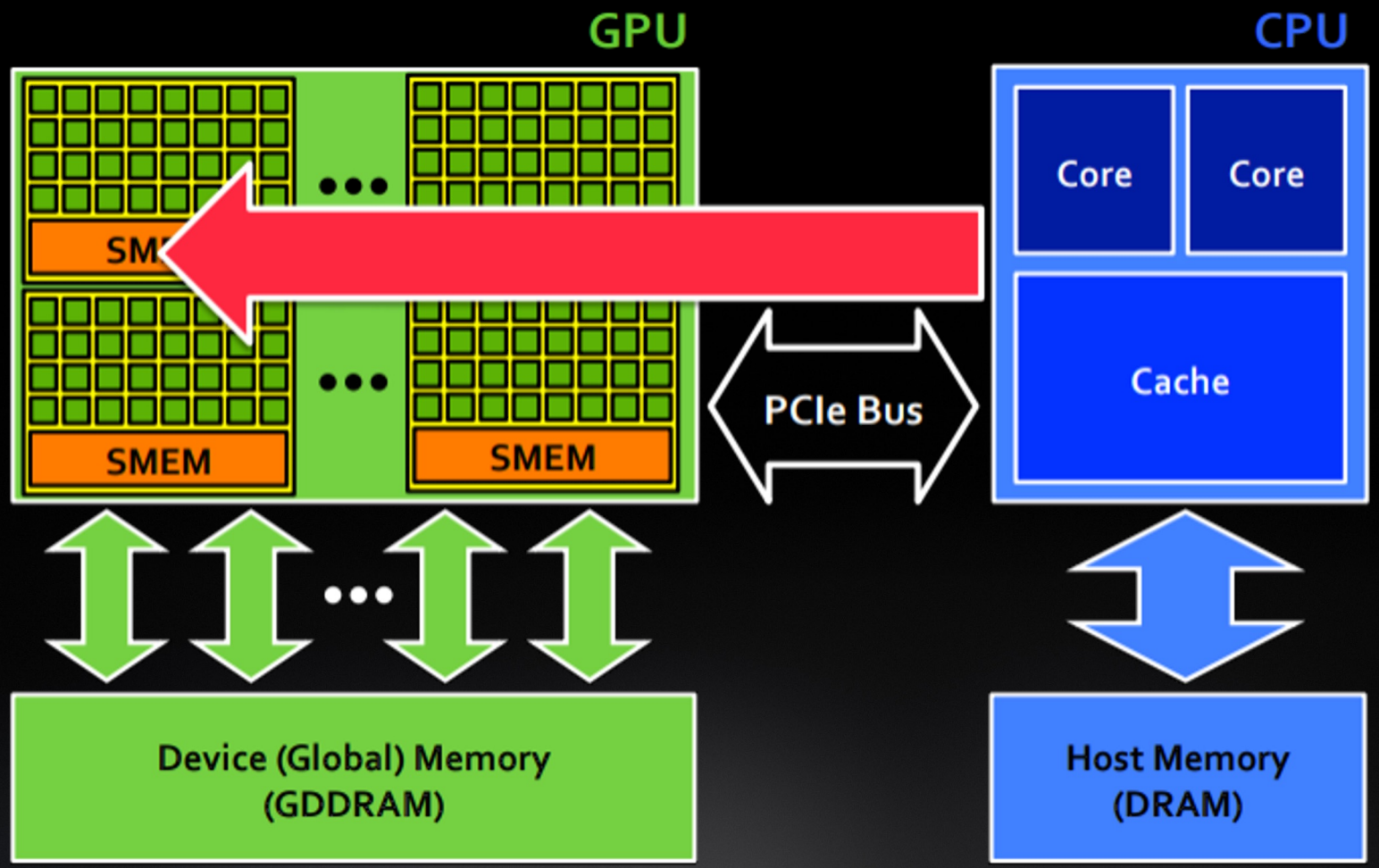
GPU Accelerator
Optimized for
Parallel Tasks



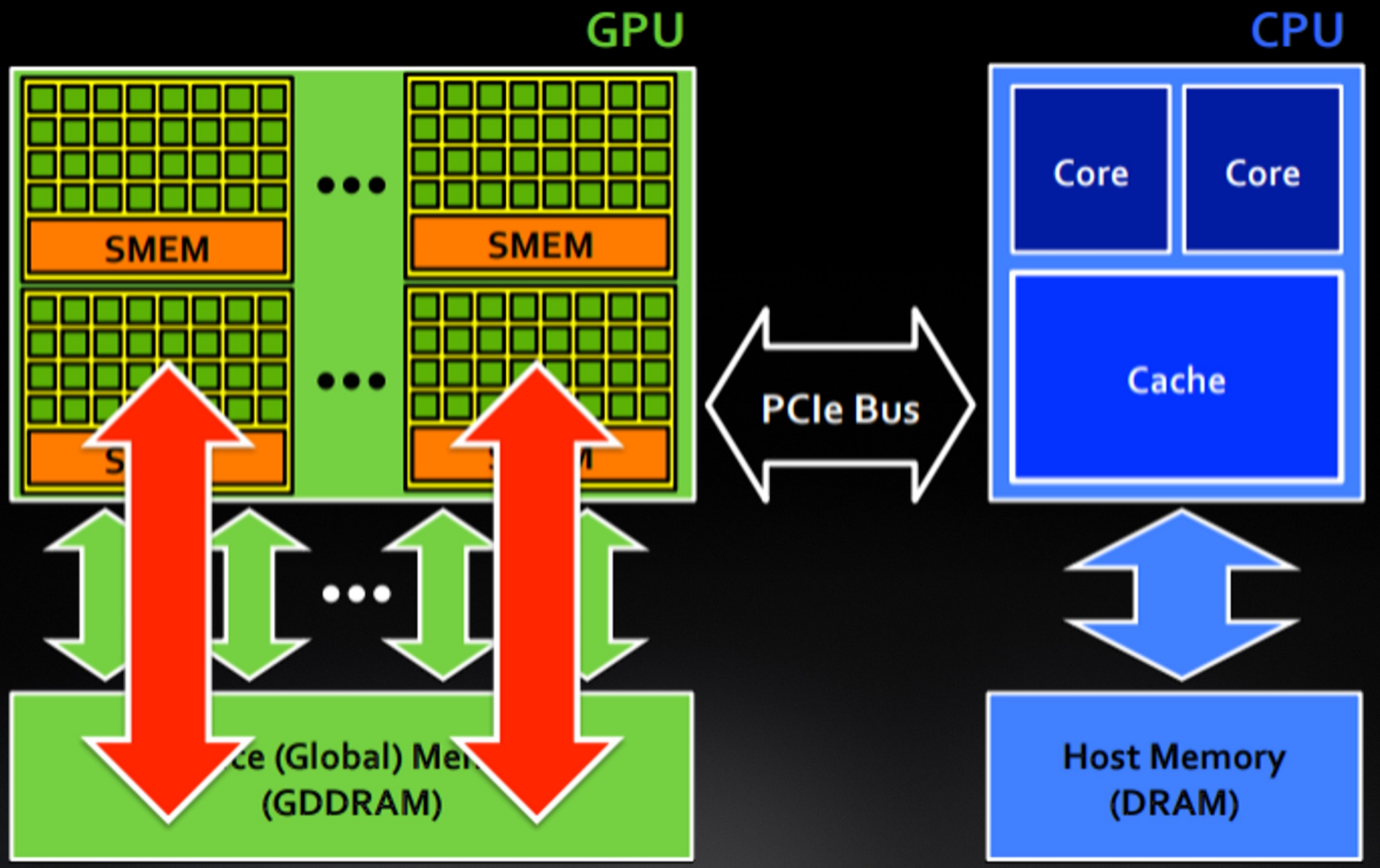
HETEROGENEOUS PROGRAMMING



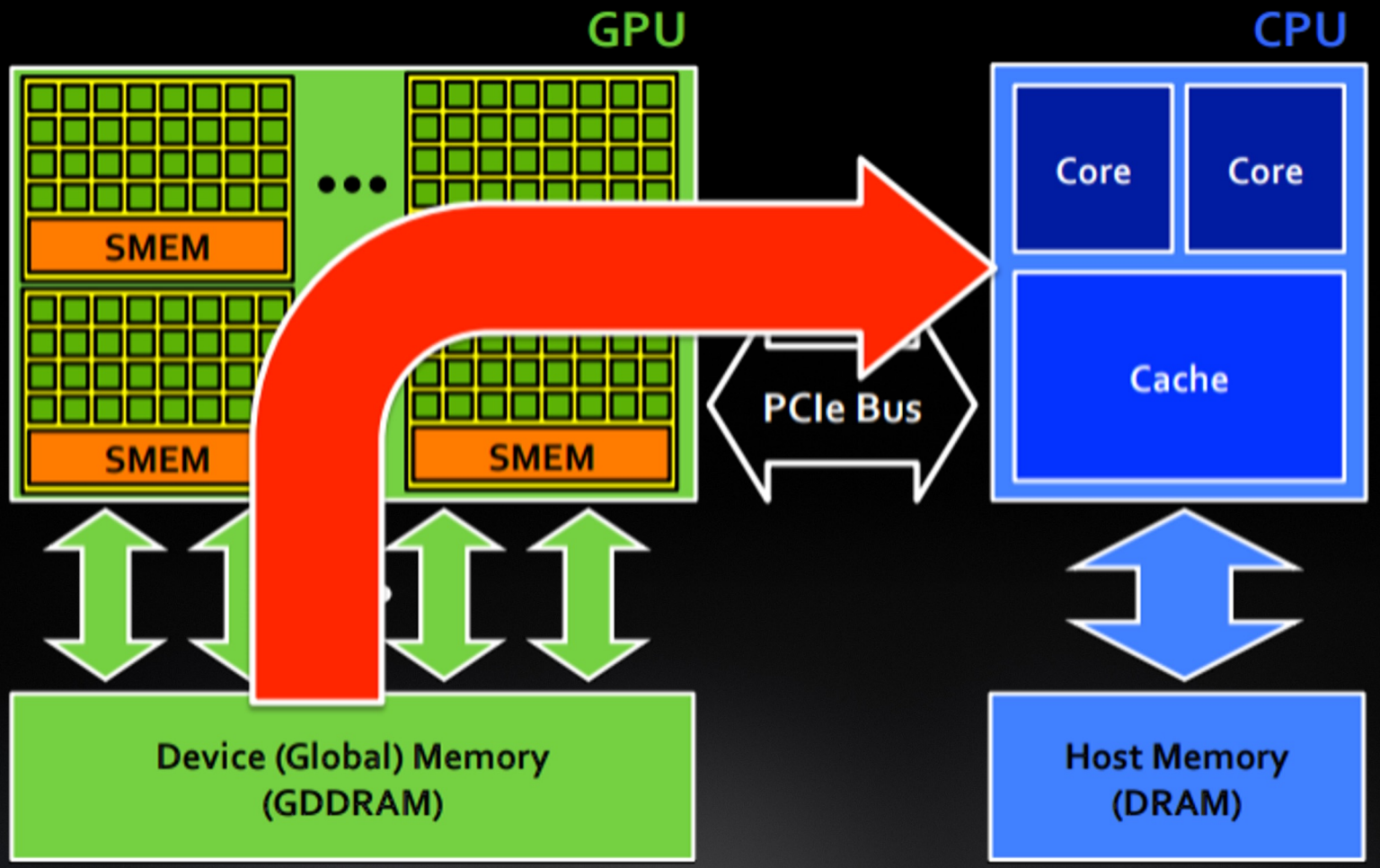
EXECUTION FLOW - H2D



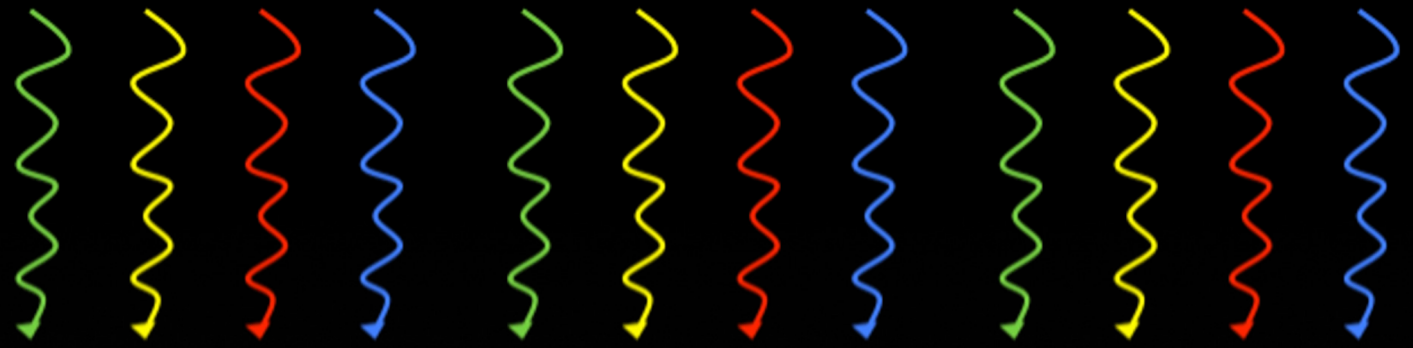
EXECUTION FLOW - KERNEL LAUNCH



EXECUTION FLOW - D2H

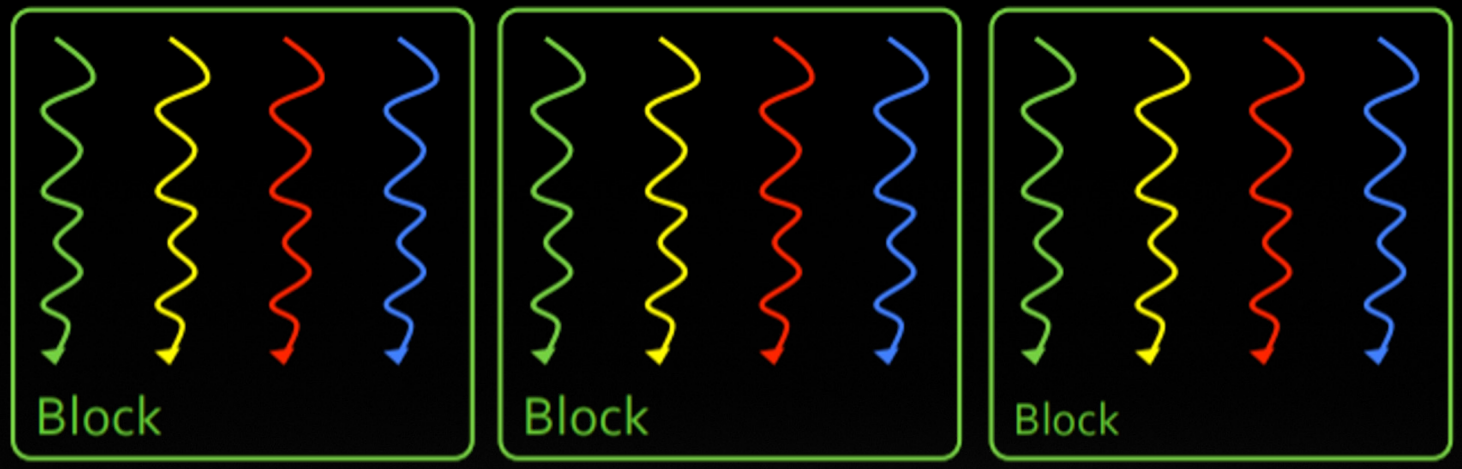


(MANY MANY) THREADS



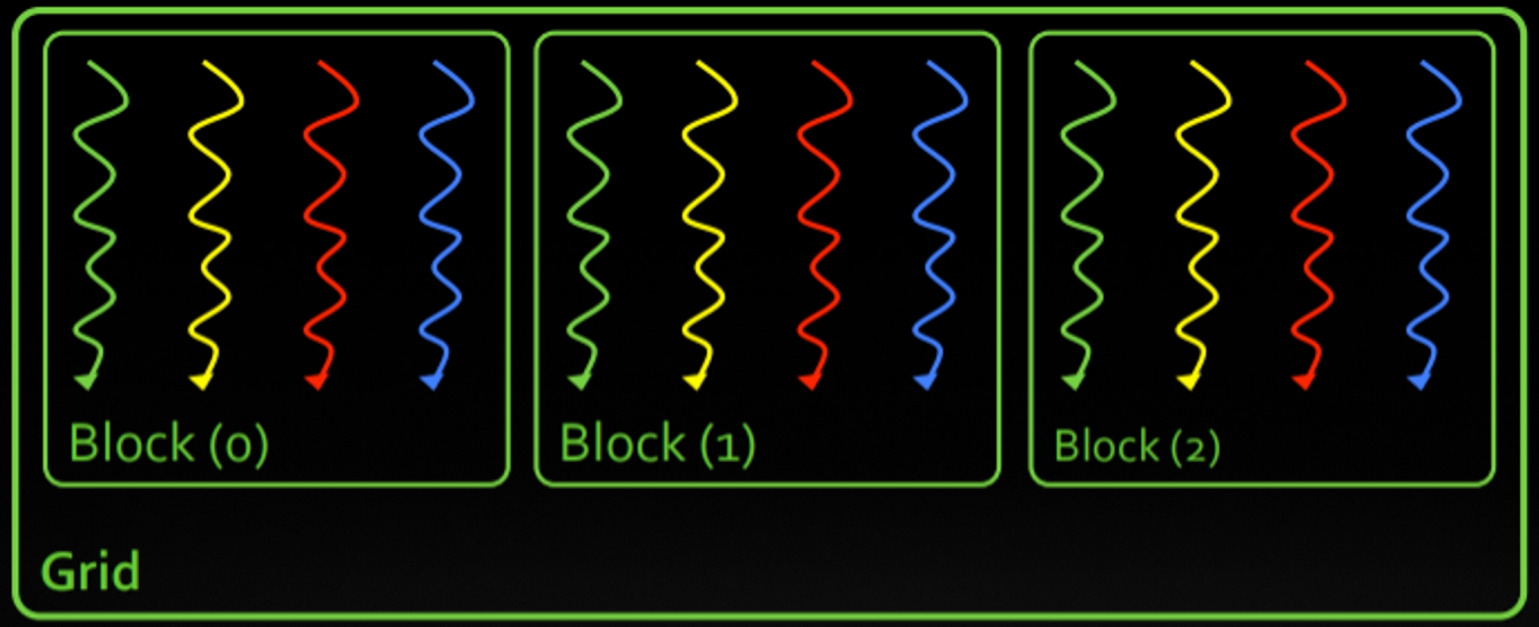
- GPUs can handle thousands of concurrent threads
- CUDA programming model supports even more
  - Allows a kernel launch to specify more threads than the GPU can execute concurrently
  - Helps to amortize kernel launch times

BLOCKS OF THREADS



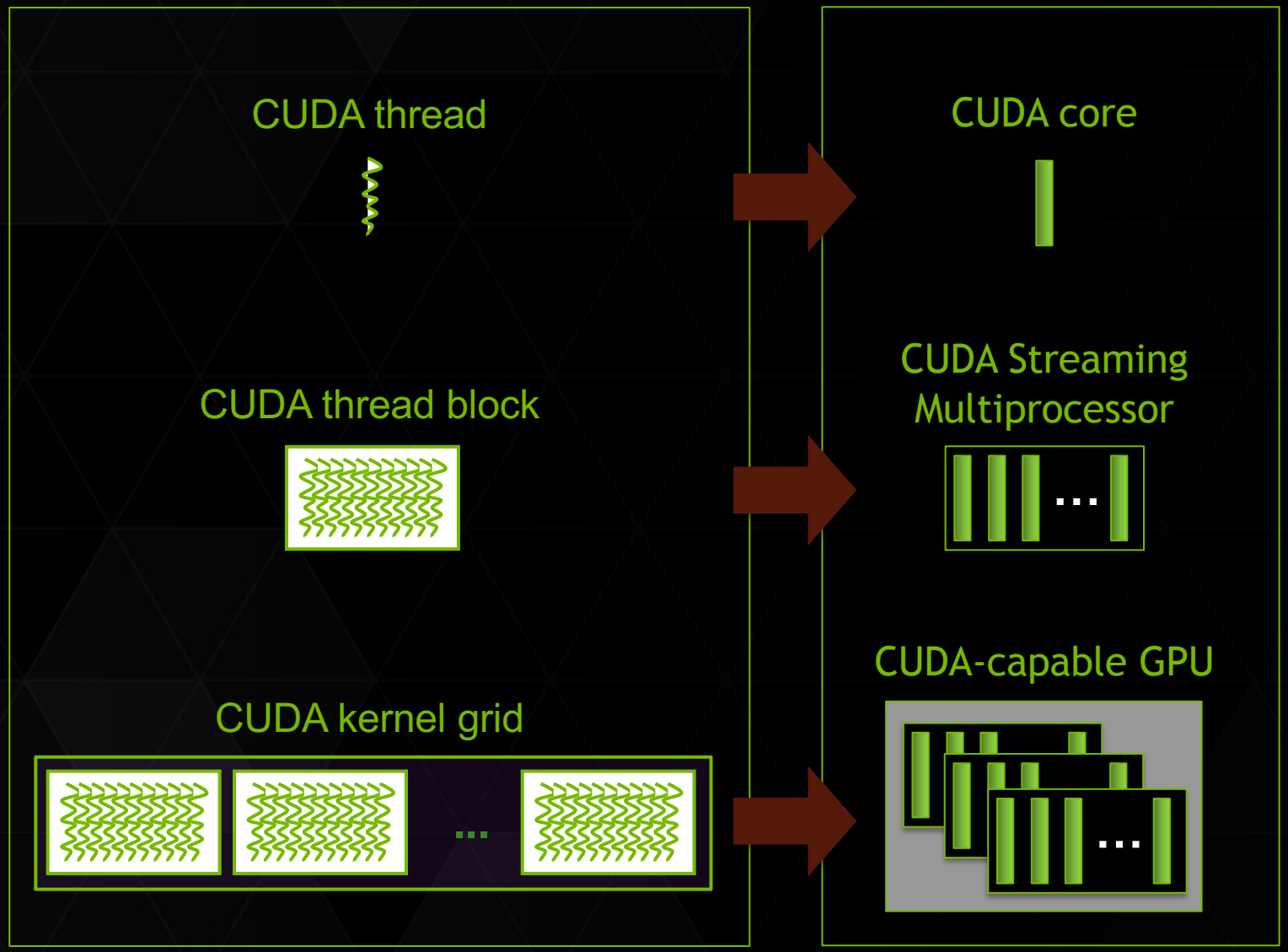
- Threads are grouped into blocks

GRID OF BLOCKS



- Threads are grouped into blocks
- Blocks are grouped into a grid
- A kernel is executed as a grid of blocks of threads

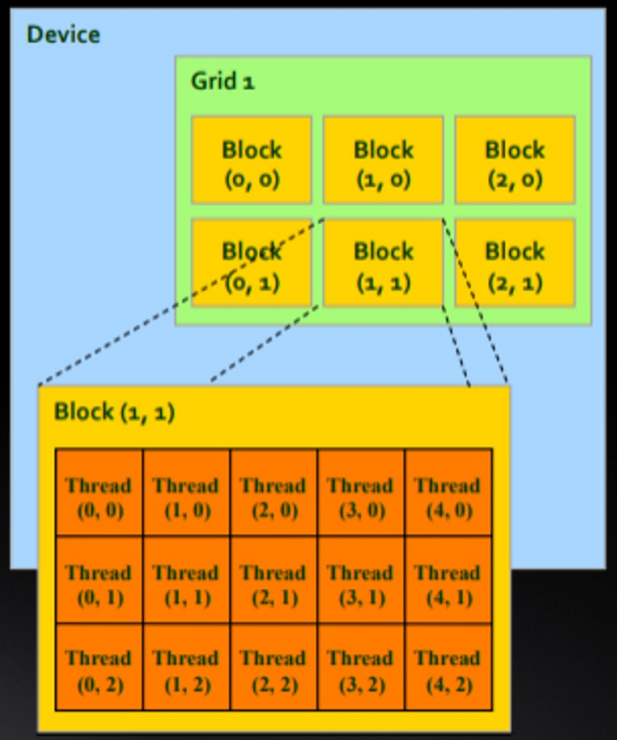
MAPPING SW<->HW



- Each thread is executed by a core
- Each block is executed by one SM and does not migrate
- Several concurrent blocks can reside on one SM depending on the blocks' memory requirements and the SM's memory resources
- Each kernel is executed on one device
- Multiple kernels can execute on a device at one time

MAPPING SW<->HW

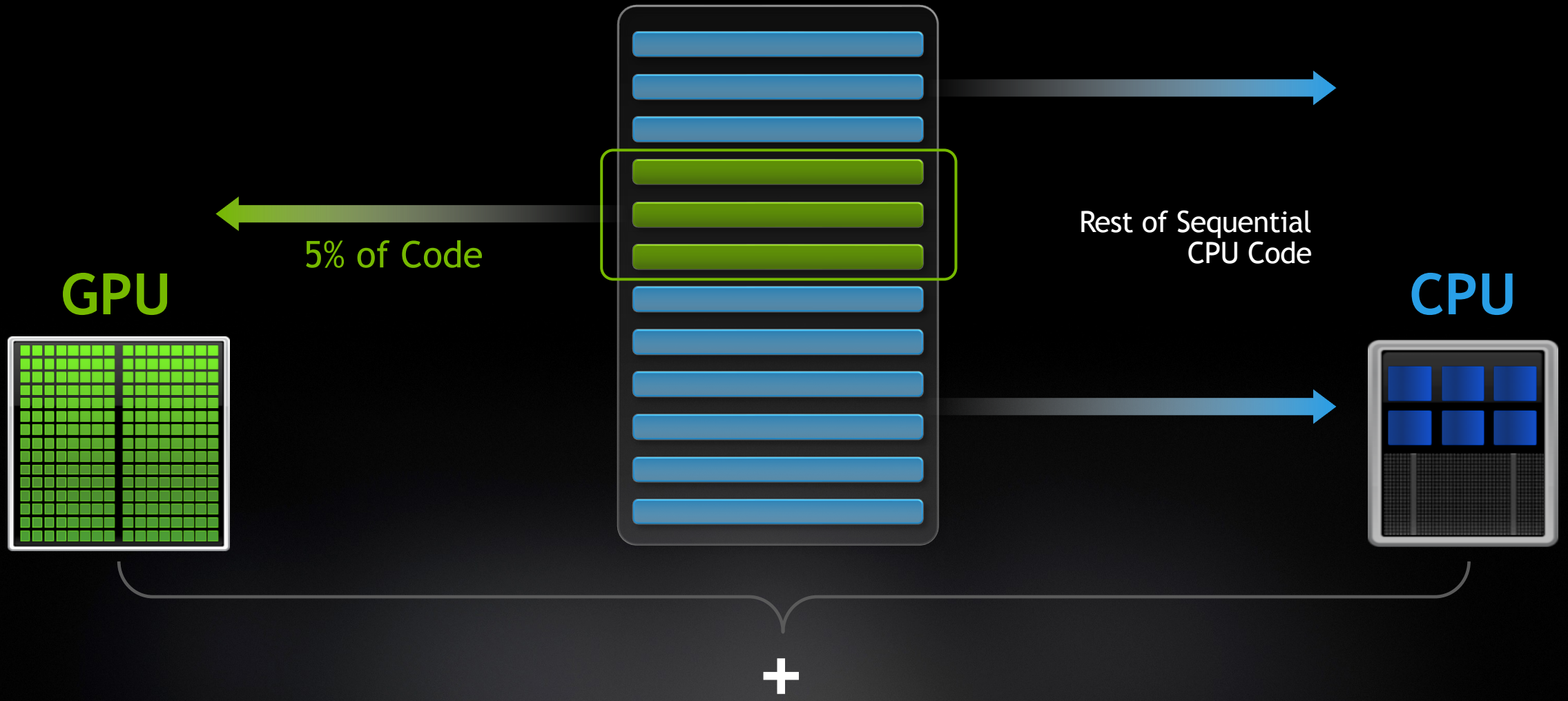
- Threads
  - 3D IDs, unique within a block
- Thread Blocks
  - 3D IDs, unique within a grid
- Dimensions set at launch
  - Can be unique for each grid
- Built-in variables
  - `threadIdx`, `blockIdx`
  - `blockDim`, `gridDim`
- Programmers usually select dimensions that simplify the mapping of the application data to CUDA threads



MANY WAYS TO PROGRAM A GPU

(PROGRAMMING MODELS)

HOW GPU ACCELERATION WORKS



GPU PROGRAMMING IN 2023 AND BEYOND

Math Libraries | Standard Languages | Directives | CUDA

```
std::transform(par, x, x+n, y, y,
    [=](float x, float y) {
        return y + a*x;
});
```

```
do concurrent (i = 1:n)
   y(i) = y(i) + a*x(i)
enddo
```

GPU Accelerated C++ and Fortran

```
#pragma acc data copy(x,y)
{

...

std::transform(par, x, x+n, y, y,
    [=](float x, float y) {
        return y + a*x;
});

...

}
```

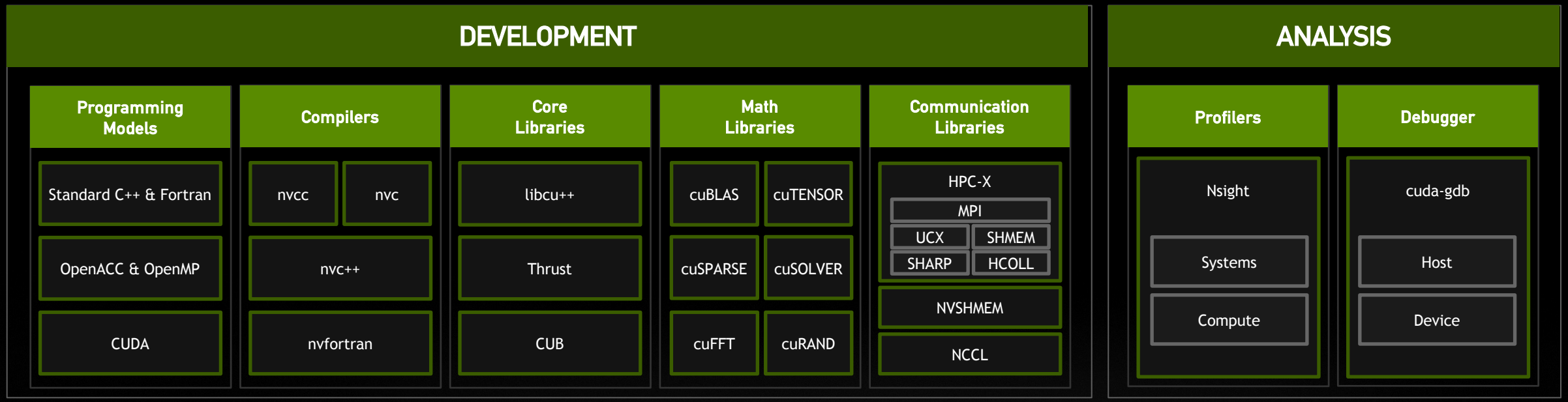
Incremental Performance Optimization with Directives

```
__global__
void saxpy(int n, float a,
           float *x, float *y) {
  int i = blockIdx.x*blockDim.x +
          threadIdx.x;
  if (i < n) y[i] += a*x[i];
}

int main(void) {
  cudaMallocManaged(&x, ...);
  cudaMallocManaged(&y, ...);
  ...
  saxpy<<<(N+255)/256,256>>>(...,x, y)
  cudaDeviceSynchronize();
  ...
}
```

Maximize GPU Performance with CUDA C++/Fortran

GPU Accelerated Math Libraries

NVIDIA HPC SDK

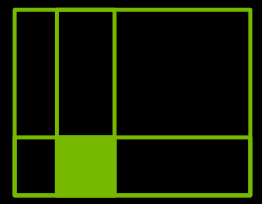


Develop for the NVIDIA Platform: GPU, CPU and Interconnect

Libraries | Accelerated C++ and Fortran | Directives | CUDA

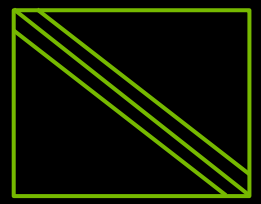
7-8 Releases Per Year | Freely Available

GPU ACCELERATED MATH LIBRARIES



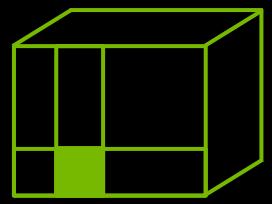
cuBLAS

BF16, TF32 and FP64 Tensor Cores



cuSPARSE

Increased memory BW, Shared Memory & L2



cuTENSOR

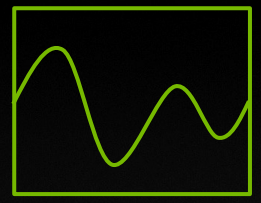
BF16, TF32 and FP64 Tensor Cores

cuSOLVER

BF16, TF32 and FP64 Tensor Cores

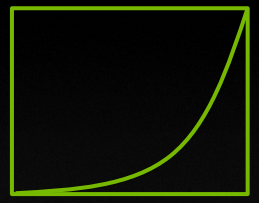
nvJPEG

Hardware Decoder



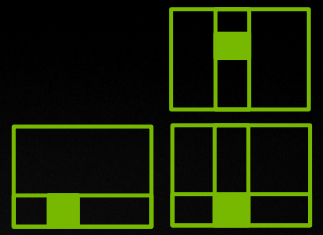
cuFFT

BF16, TF32 and FP64 Tensor Cores



CUDA Math API

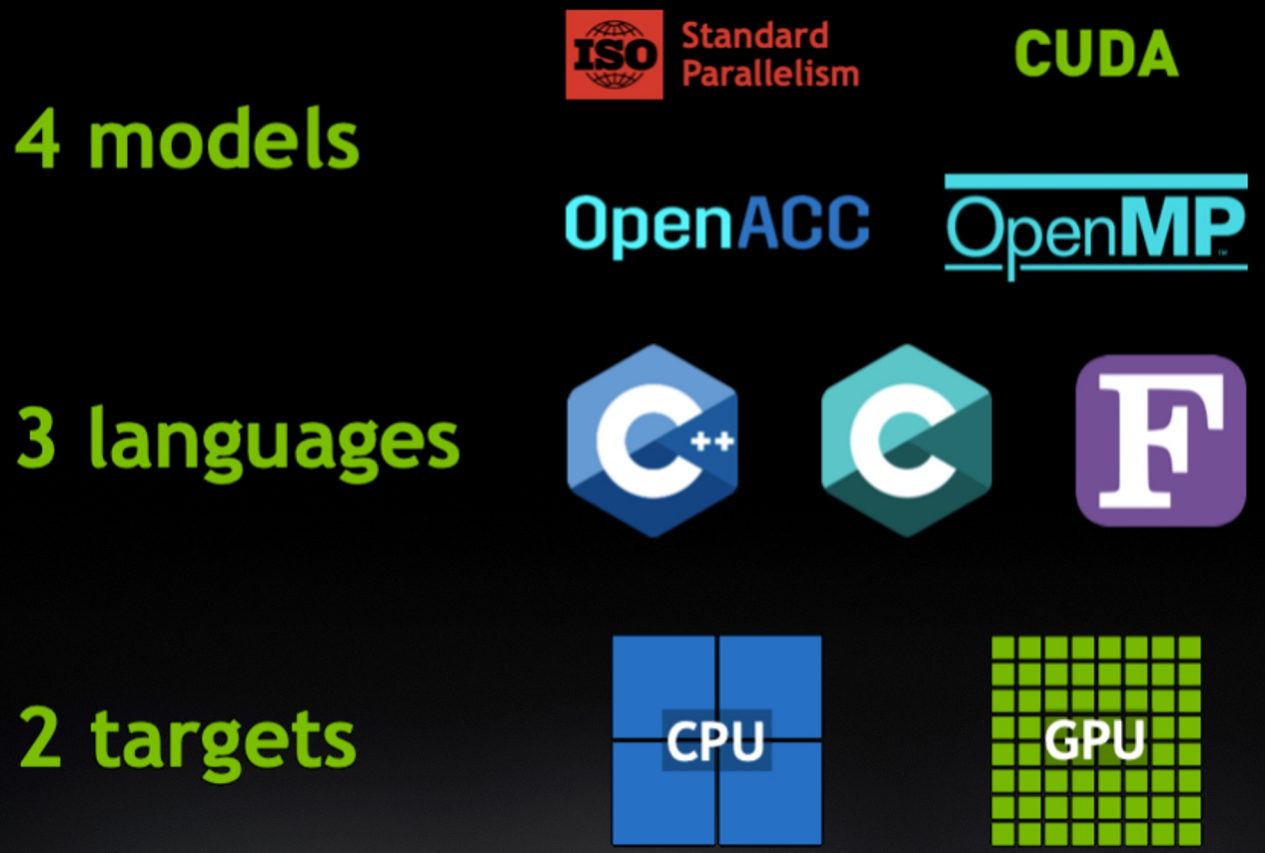
Increased memory BW, Shared Memory & L2



CUTLASS

BF16 & TF32 Support

NVIDIA HPC SDK



Programming Model Interoperability

All programming models are interoperable:

- Can be used in the same source file.
- ABI compatible with each other and NVCC.

```
nvc++ -stdpar -cuda -acc -mp
```

CUDA

OpenACC

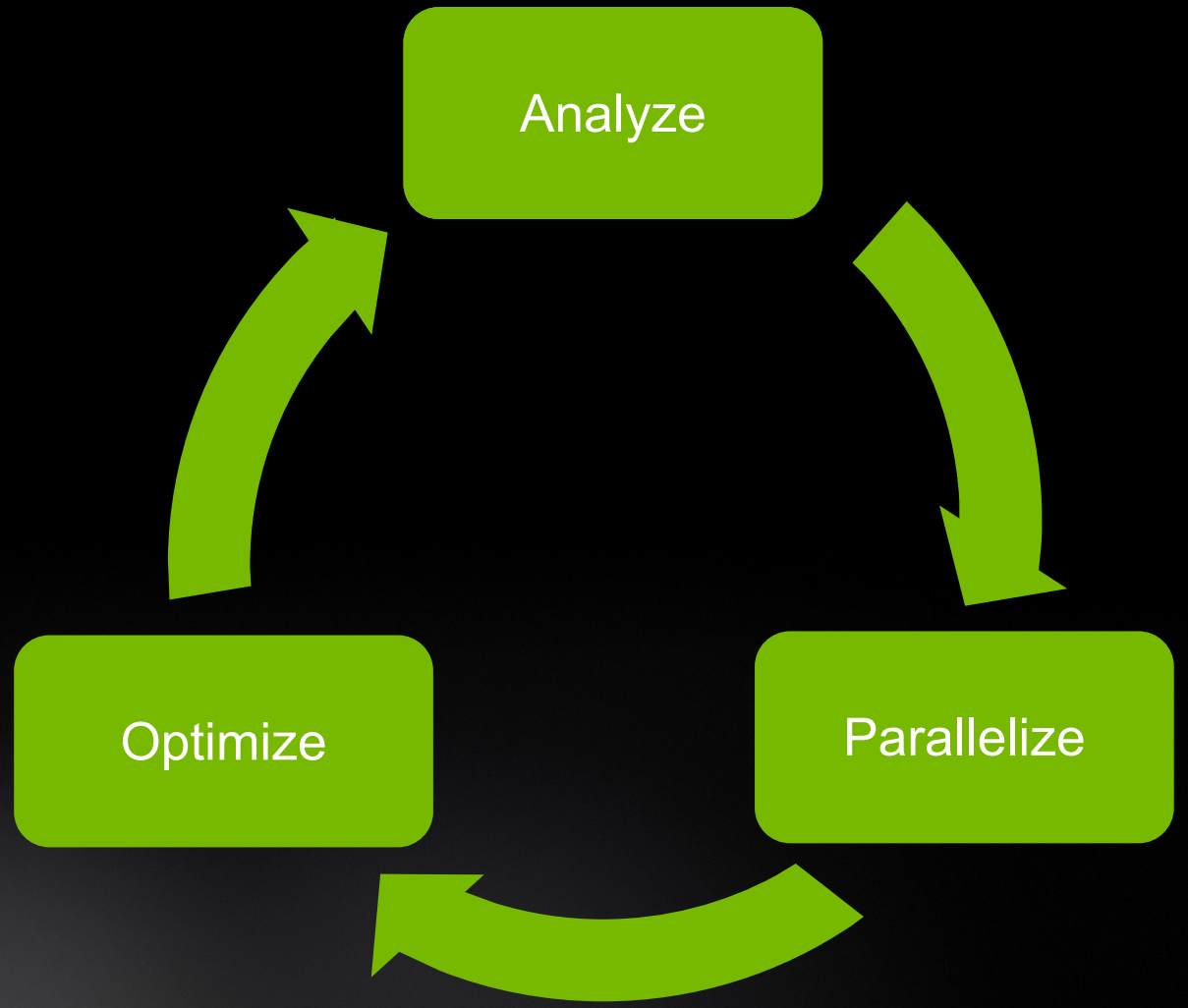
OpenMP

```
__global__ void
times_two(float* first, uint64_t n) {
  auto t = blockIdx.x*blockDim.x+threadIdx.x;
  if (t < n) first[t] *= 2;
}

void compute(std::vector<float>& x) {
  auto const b = ((x.size() - 1)/ 64) + 1;
  times_two<<<b, 64>>>(x.data(), x.size());
  cudaDeviceSynchronize();

  #pragma omp target teams loop
  for (auto& e : x) e *= e;

  #pragma acc parallel loop
  for (auto& e : x) e += e;

  std::sort(std::execution::par,
            x.begin(), x.end());
}
```


PROFILING WITH NSIGHT
SYSTEM
AND NVTX

DEVELOPMENT CYCLE

- **Analyze** your code to determine most likely places needing parallelization or optimization.
- **Parallelize** your code by starting with the most time consuming parts and check for correctness.
- **Optimize** your code to improve observed speed-up from parallelization.



PROFILING SEQUENTIAL CODE

Profile Your Code

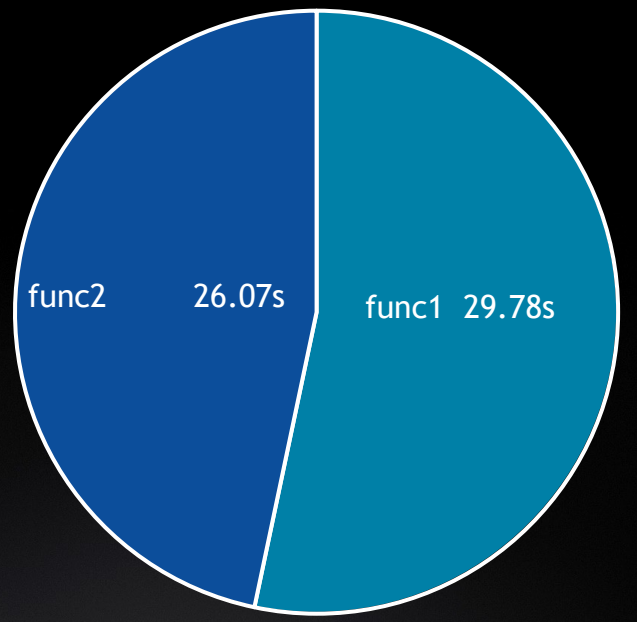
Obtain detailed information about how the code ran.

This can include information such as:

- Total runtime
- Runtime of individual routines
- Hardware counters

Identify the portions of code that took the longest to run. We want to focus on these “hotspots” when parallelizing.

Hotspot Analysis



PROFILING SEQUENTIAL CODE

Using Command Line Interface (CLI)

NVIDIA Nsight Systems CLI provides

- Simple interface to collect data
- Can be copied to any system and analysed later
- Profiles both serial and parallel code
- For more info enter `nsys --help` on the terminal

To profile a serial application with NVIDIA Nsight Systems, we use NVIDIA Tools Extension (NVTX) API functions in addition to collecting backtraces while sampling.

PROFILING SEQUENTIAL CODE

NVIDIA Tools Extension API (NVTX) library

What is it?

- A C-based Application Programming Interface (API) for annotating events
- Can be easily integrated to the application
- Can be used with NVIDIA Nsight Systems

Why?

- Allows manual instrumentation of the application
- Allows additional information for profiling (e.g: tracing of CPU events and time ranges)

How?

- Import the header only C library `nvToolsExt.h`
- Wrap the code region or a specific function with `nvtxRangePushA()` and `nvtxRangePop()`

```c
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include "laplace2d.h"
#include <nvtx3/nvToolsExt.h>

int main(int argc, char** argv)
{
    ...

    nvtxRangePushA("init");
    initialize(A, Anew, m, n);
    nvtxRangePop();

    ...

    nvtxRangePushA("while");
    while ( error > tol && iter < iter_max )
    {
        nvtxRangePushA("calc");
        error = calcNext(A, Anew, m, n);
        nvtxRangePop();

        nvtxRangePushA("swap");
        swap(A, Anew, m, n);
        nvtxRangePop();

        ...
    }
    nvtxRangePop();

    ...
}
```

-t	Selects the APIs to be traced (nvtx in this example)
--status	if true, generates summary of statistics after the collection
-b	Selects the backtrace method to use while sampling. The option dwarf uses DWARF's CFI (Call Frame Information)
--force-overwrite	if true, overwrites the existing results
-o	sets the output (qdrep) filename

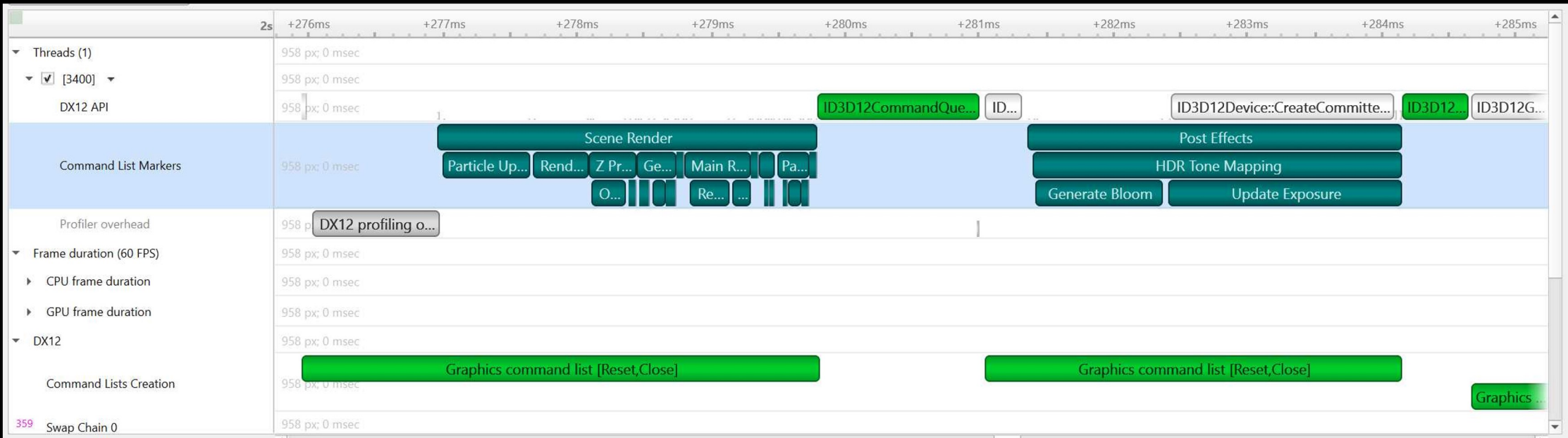
```
nsys profile -t nvtx --status=true ./laplace-seq
```

```
Processing events...
Capturing symbol files...
Saving intermediate "/home/mozhgank/Code/openacc-training-materials/labs/module4/English/C/solutions/parallel/laplace-seq.qdstrm" file to disk...

Importing [==========================================================100%]
Saved report file to "/home/mozhgank/Code/openacc-training-materials/labs/module4/English/C/solutions/parallel/laplace-seq.qdrep"
Exporting 70802 events: [=================================================100%]

Exported successfully to
/home/mozhgank/Code/openacc-training-materials/labs/module4/English/C/solutions/parallel/laplace-seq.sqlite

Generating NVTX Push-Pop Range Statistics...
NVTX Push-Pop Range Statistics (nanoseconds)

Time(%)      Total Time   Instances         Average         Minimum         Maximum  Range
-------  --------------  ----------  --------------  --------------  --------------  --------
   49.9     55754497966           1  55754497966.0     55754497966     55754497966  while
   26.5     29577817696        1000     29577817.7        29092956        65008545  calc
   23.4     26163892482        1000     26163892.5        25761418        60129514  swap
    0.1       137489808           1    137489808.0       137489808       137489808  init
```

The qdrep file is for GUI

USER ANNOTATIONS APIS FOR CPU & GPU
NVTX, OPENGL, VULKAN, AND DIRECT3D PERFORMANCE MARKERS

EXAMPLE: VISUAL MOLECULAR DYNAMICS (VMD) ALGORITHMS VISUALIZED WITH NVTX ON CPU



Events View | Process [16084] ModelViewer.exe (1 of 1 thread)

Filter... | All | Search...

#	Name	Duration	TID	Start
686	ID3D12GraphicsCommandList::BeginEvent	5.900 µs	3400	2.27697s
687	CPU Marker Start (5) Particle Update	200 ns	3400	2.277s
690	ID3D12GraphicsCommandList::EndEvent	200 ns	3400	2.27765s
691	CPU Marker Start (5) RenderLightShadows	200 ns	3400	2.27768s
694	ID3D12GraphicsCommandList::EndEvent	100 ns	3400	2.27807s
695	CPU Marker Start (5) Z PrePass	200 ns	3400	2.2781s
696	ID3D12GraphicsCommandList::BeginEvent	200 ns	3400	2.2781s
697	CPU Marker Start (5) Opaque	1 ns	3400	2.27812s
698	ID3D12GraphicsCommandList::BeginEvent	1 ns	3400	2.27812s
699	CPU Marker End (5)	200 ns	3400	2.27836s
700	ID3D12GraphicsCommandList::EndEvent	200 ns	3400	2.27836s
701	CPU Marker Start (5) Cutout	100 ns	3400	2.27839s
704	ID3D12GraphicsCommandList::EndEvent	100 ns	3400	2.27842s
705	CPU Marker End (5)	100 ns	3400	2.27843s
706	ID3D12GraphicsCommandList::EndEvent	100 ns	3400	2.27843s
707	CPU Marker Start (5) Generate SSAO	100 ns	3400	2.27846s

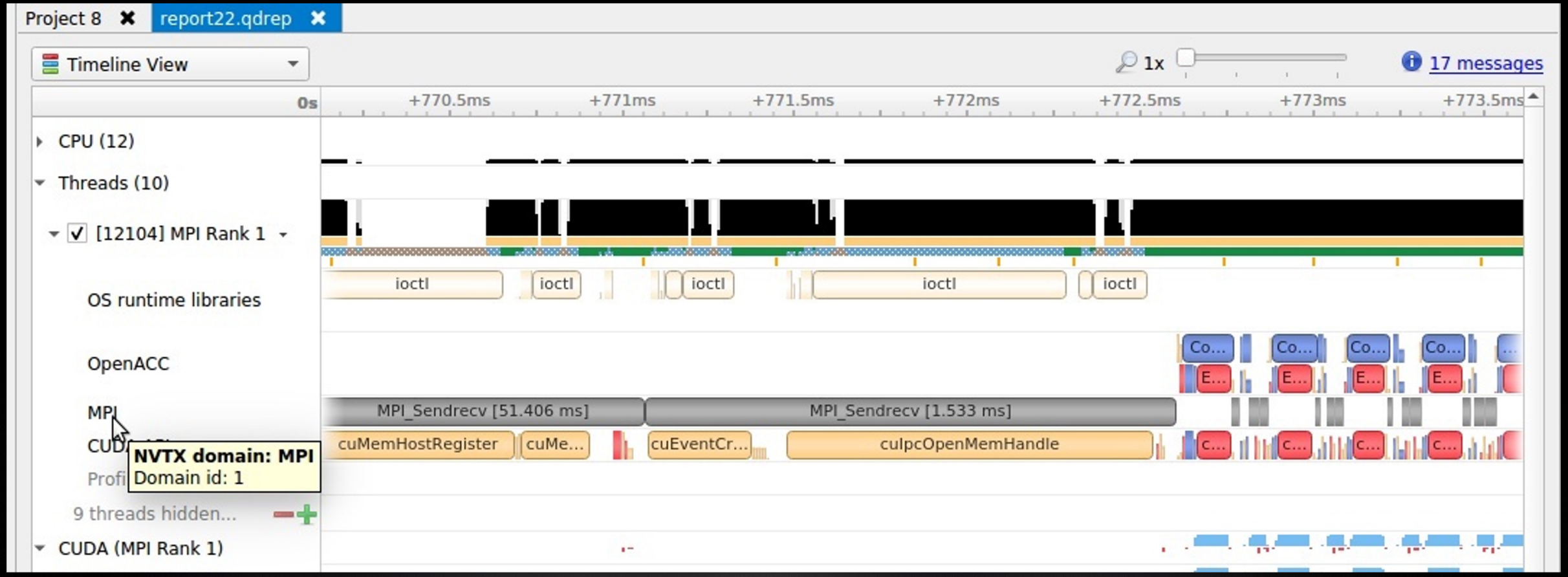
CPU Marker Start (5) Opaque

Start: 2.27812s
End: 2.27812s
Duration: 1 ns
Thread: 3400

EVENT TABLE

NVIDIA

MPI & OPENACC TRACE



REFERENCES

https://developer.nvidia.com/nsight-systems

https://developer.nvidia.com/nsight-systems

- Register NVIDIA developer
- Download and Install Nsight Systems on Windows (Your computer)

More Information:

https://docs.nvidia.com/nsight-systems

https://developer.nvidia.com/hpc-sdk

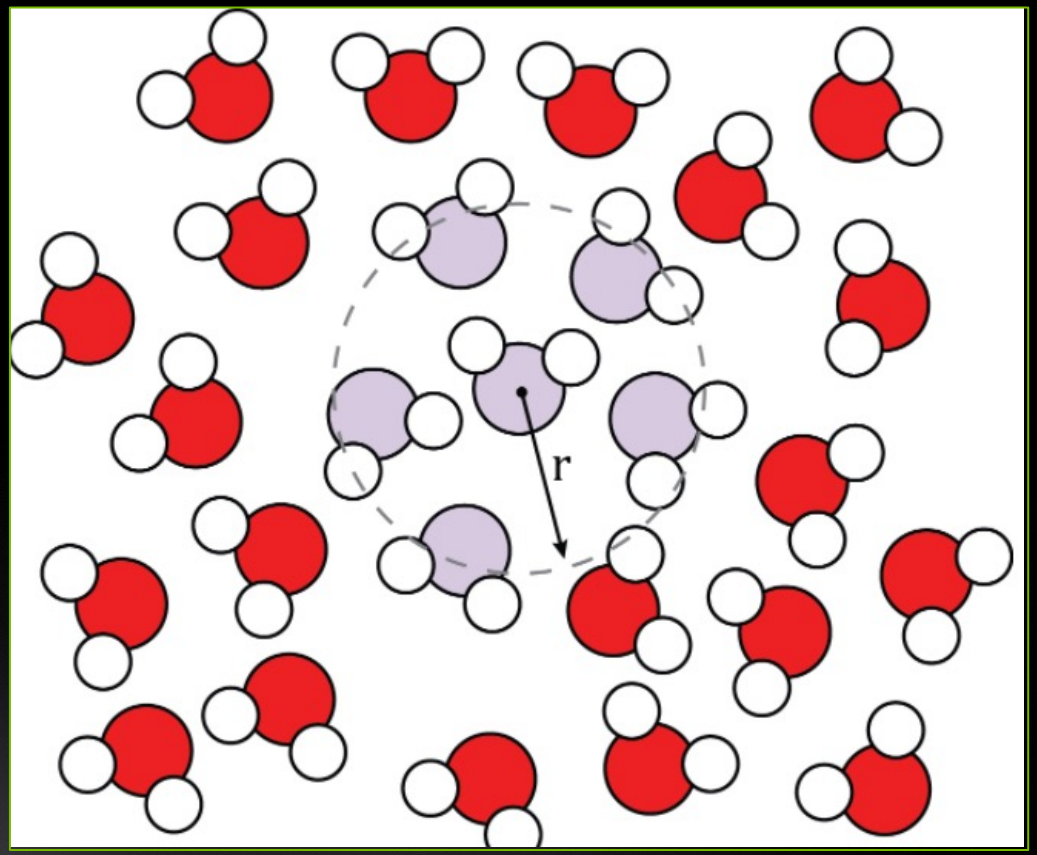
BOOTCAMP MINI-APP

APPLICATION

Molecular Simulation

RDF

The radial distribution function (RDF) denoted in equations by g(r) defines the probability of finding a particle at a distance r from another tagged particle.



RDF

Pseudo Code - C

```
for (int frame=0;frame<nconf;frame++) {
        for(int id1=0;id1<numatm;id1++) {
                for(int id2=0;id2<numatm;id2++) {
                        did1 = frame*numatm+id1
                        did2 = frame*numatm+id2
                        dx=d_x[did1]-d_x[did2];
                        dy=d_y[did1]-d_y[did2];
                        dz=d_z[did1]-d_z[did2];
                        r=sqrtf(dx*dx+dy*dy+dz*dz);

                        if (r<cut) {
                                ig2=(int)(r/del);
                                d_g2[ig2] = d_g2[ig2] +1 ;
                        }
                }
        }
}
```

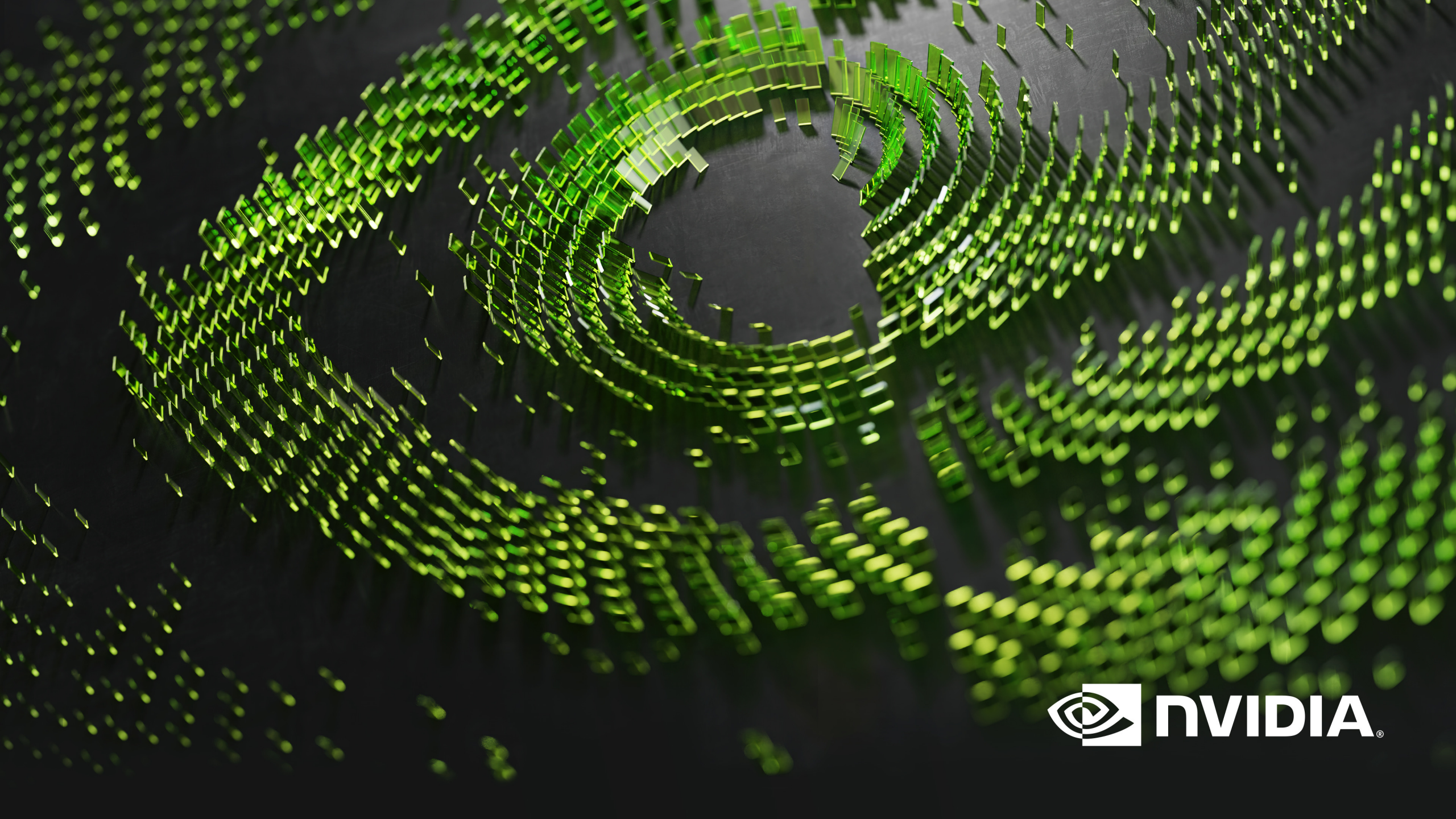
- Across Frames
- Find Distance
- Reduction

RDF

Pseudo Code - Fortran

```
do iconf=1,nframes
   if (mod(iconf,1).eq.0) print*,iconf

   do i=1,natoms
      do j=1,natoms
            dx=x(iconf,i)-x(iconf,j)
            dy=y(iconf,i)-y(iconf,j)
            dz=z(iconf,i)-z(iconf,j)

            r=dsqrt(dx**2+dy**2+dz**2)
            if(r<cut)then
                  g(ind)=g(ind)+1.0d0
            endif
      enddo
   enddo
enddo
```

- Across Frames
- Find Distance
- Reduction


NVIDIA