CSC4005 Tutorial 7

Nov 4, 2021

This tutorial will cover...

- Assignment 3 Clarification
- N-Body Simulation
- MPI: Passing a struct
- Building OpenMP Applications
- OpenMP Basic
- Building CUDA Applications
- CUDA Basic

Illustrations by Pixeltrue on icons8

Assignment 3 Clarification

Illustrations by Pixeltrue on icons8

Assignment 3 Clarification

- No clarification for now. (*as of Nov 4*)
- MPI, OpenMP, pthread, CUDA
  - And the bonus: MPI+OpenMP (10pts)
  - And sequential for free
- See grading criteria

“I attempted but it doesn’t work”

I have a CUDA-capable card on my device

I feel good running forwarded GUI from the server

	Yes	No
Yes	Cool!	Run your program forwarded rom the server
No	Try to self-build it	**Oops...**

No luck from CUDA or X Server?

- TC301 might have PCs with CentOS & CUDA
- Username: cuhksz / Password: P@ss1234
- Shared with CSC/BIM, so do your work early
- Keep your data safe!

- Aliyun for Students can also be a choice
- If you're remote: let's talk to see if there are solutions...

N-Body Simulation

Illustrations by Pixeltrue on icons8

N-Body Simulation

- Computer, Mathemeatics & Physics...
- The demo

Describe the “body”

- For each body:
  - Radius, Weight
  - X pos, Y pos
  - X speed, Y speed
  - X acceleration, Y acceleration

Process

- In the beginning:
  - Generate bodies with the same radius, random position and mass
- In each round:
  - Calculate the acceleration
  - Calculate the speed
  - Calculate the distance (move the body)
  - Handle collision (if any)

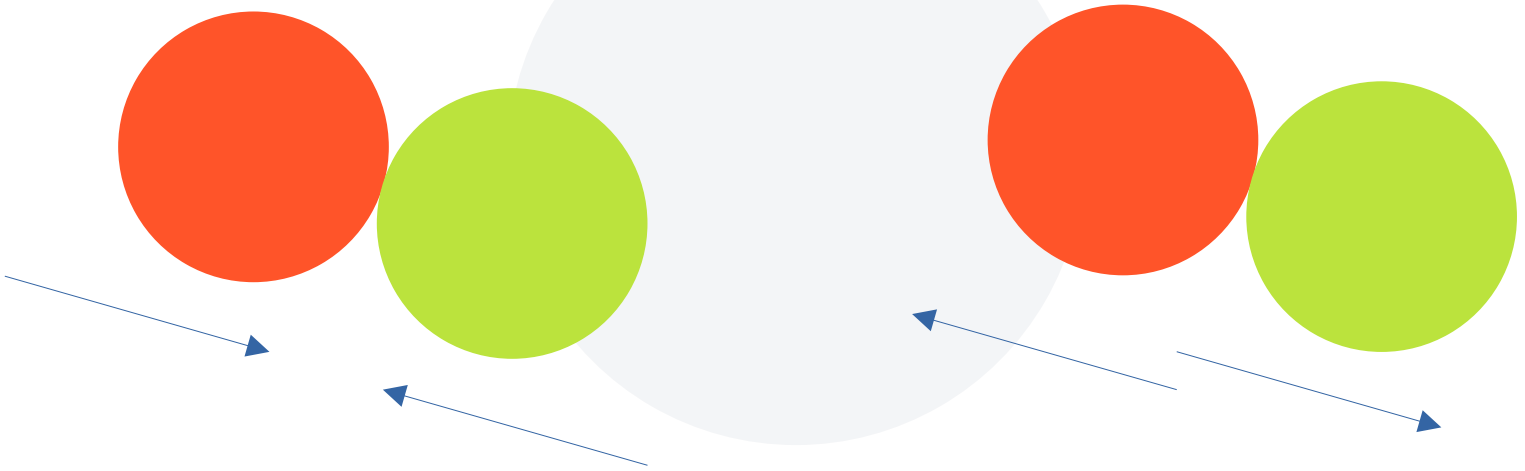
Events

- Forces: Gravity from other balls
- Special cases:
  - Ball & ball collision
  - Ball & wall collision

$$F = G\frac{m_1 m_2}{r^2}$$

Collision

- Ball+Ball / Ball+Wall
- Speed towards the collision: Reversed



Parameters

- Space: The size of the canvas
- Gravity: The so-called 6.67×10^{-11}
  - Feel free to change it to another constant for aesthetics
- Elapse: Animation running speed
- Maximum Mass

MPI: Passing a struct

Illustrations by Pixeltrue on icons8

MPI: Passing Custom Structs

- MPI_INT, MPI_DOUBLE, MPI_FLOAT...
- But how about passing a struct via MPI?

Define a MPI_Datatype

- Create & Commit
- Create:
  - MPI_Type_contiguous (simplist)
  - MPI_Type_vector (still readable)
  - MPI_Type_create_struct (???)
  - and so on

An Easy-to-use Hack:
A struct with multiple fields of the same data type

- Make use of `__attribute__((packed))`
- See mpi-struct.cpp

* Please consult professional C++ users when using contents from this page.

Buliding OpenMP Applications

Illustrations by Pixeltrue on icons8

Building OpenMP

- We'll have you covered with CMake
- But you can also do it yourself
  - e.g.

```
clang++ openmp-sample.cpp -fopenmp -Wall \
  -o ./openmp-sample
```

Check if OpenMP is really there

- OpenMP programs can just build without OpenMP
- Use `-Wall` to see if it's really built with OpenMP

```
openmp-sample.cpp:7: warning: ignoring '#pragma omp parallel' [-Wunknown-pragmas]
    7 | #pragma omp parallel for shared(data) default(none)
      |
```

OpenMP Basic

Illustrations by Pixeltrue on icons8

OpenMP

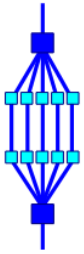
- Shared memory – on the same node
- Use #pragma to give orders

OpenMP

- `#pragma omp parallel (if ...) (default(...) \`
  `(shared(...) private(...))`
- `#pragma omp for (nowait)`
- `#pragma omp single (nowait)`
- `#pragma omp barrier`
- `#pragma omp critical`

(Need to be in a parallel region)

The if/private/shared clauses



if (scalar expression)

- *Only execute in parallel if expression evaluates to true*
- *Otherwise, execute serially*

```
#pragma omp parallel if (n > threshold) \
        shared(n,x,y) private(i)
  {
    #pragma omp for
     for (i=0; i<n; i++)
        x[i] += y[i];
  } /*-- End of parallel region --*/
```

private (list)

- *No storage association with original object*
- *All references are to the local object*
- *Values are undefined on entry and exit*

shared (list)

- *Data is accessible by all threads in the team*
- *All threads access the same address space*

OpenMP

- `#pragma omp parallel (if ...) (default(...) \`
  `(shared(...) private(...))`
- `#pragma omp for (nowait)`
- `#pragma omp single (nowait)`
- `#pragma omp barrier`
- `#pragma omp critical`

Default: Implicit barrier
Nowait: No barrier

See [openmp-single.cpp].

OpenMP

- `#pragma omp parallel (if ...) (default(...) \`
  `(shared(...) private(...))`
- `#pragma omp for (nowait)`
- `#pragma omp single (nowait)`
- `#pragma omp barrier`
- `#pragma omp critical`

See [openmp-barrier/].

OpenMP

- `#pragma omp parallel (if ...) (default(...) \`
  `(shared(...) private(...))`
- `#pragma omp for (nowait)`
- `#pragma omp single (nowait)`
- `#pragma omp barrier`
- `#pragma omp critical`

OpenMP: Several functions

```
#include <omp.h>

...

omp_set_num_threads()

omp_get_num_threads()

omp_get_thread_num()
```

See [openmp-thread-num.cpp].

Building CUDA Applications

Illustrations by Pixeltrue on icons8

Building CUDA

- We'll (also) have you covered with CMake
- But you can also try using nvcc
  - e.g.

```
nvcc main.cu -o ./main \
  -ccbin=/opt/rh/devtoolset-10/root/usr/bin/g++ \
  --relocatable-device-code=true
```

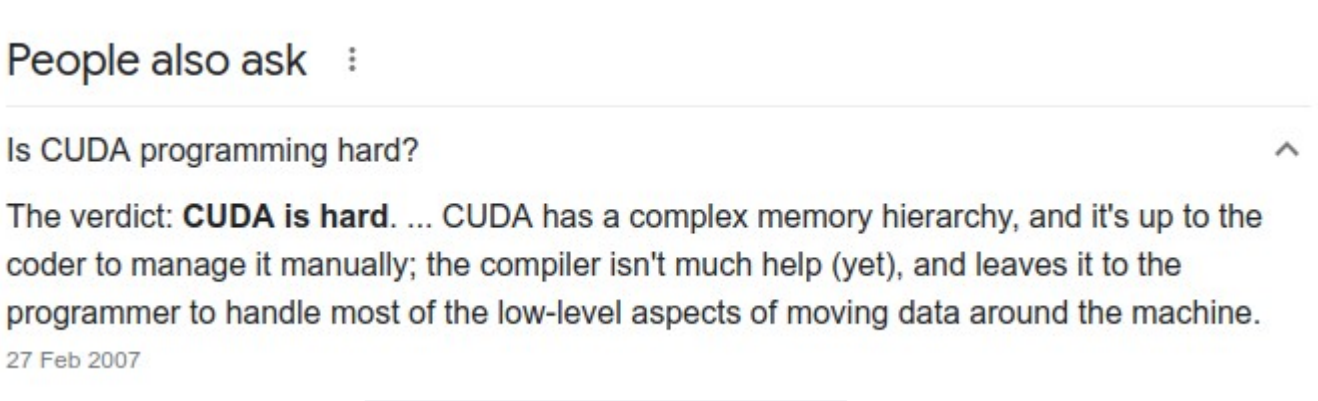
CUDA Basic

Illustrations by Pixeltrue on icons8

* Also check your CSC3150 tutorials on CUDA :)

CUDA

- One suggestion: **Start early!**



CUDA

- CUDA = Compute Unified Device Architecture
- NVIDIA only
- See supported GPU List
  - GeForce, GTX, RTX, Quadro, TITAN...

CUDA Basic Concepts

- Host: Your CPU
- Device: Your GPU
- Kernel, __global__ - Callable from the host
- __device__ - Callable from the device only
- __host__ - Callable from the host only

CUDA: Memory

- cudaMalloc(void** buffer, size_t size)
- cudaFree(void* buffer)
- cudaMemset(void* devPtr, int value, size_t count)
- cudaMemcpy (void* dst, const void* src, size_t count, enum cudaMemcpyKind kind)
  - cudaMemcpyHostToHost
  - cudaMemcpyHostToDevice
  - cudaMemcpyDeviceToHost
  - cudaMemcpyDeviceToDevice

Launching A Kernel: The Easy way

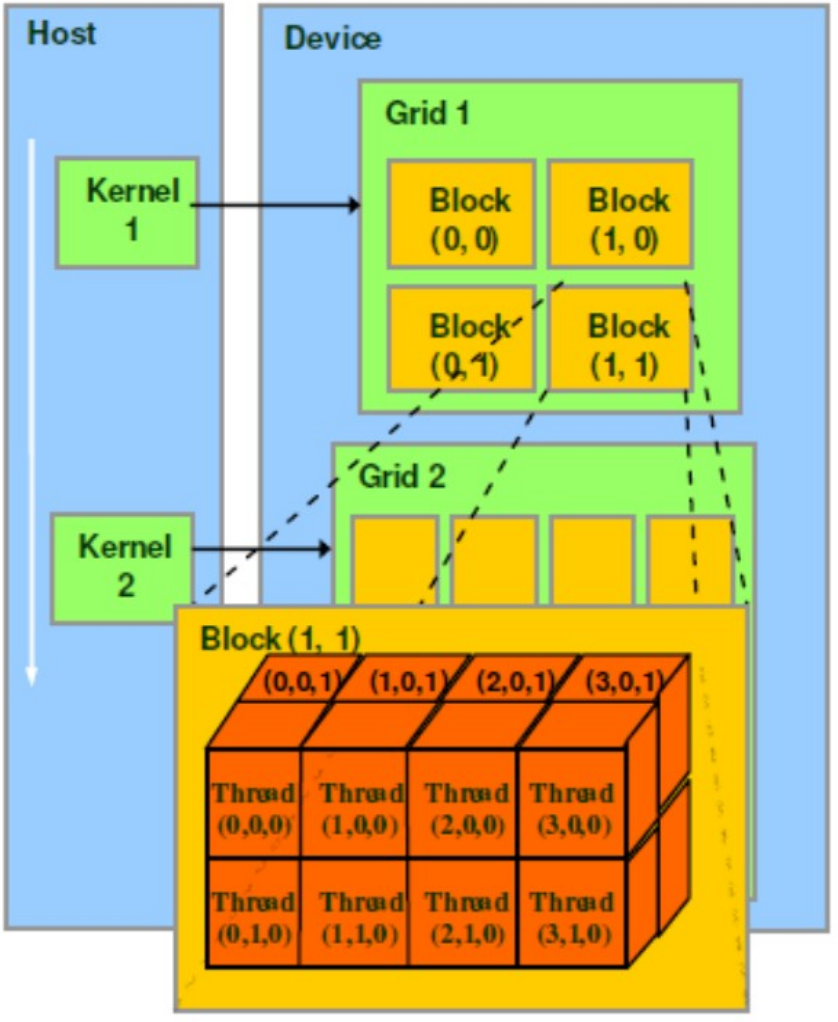
- Write a kernel (__global__)
- Launch N copies of the kernel:
  - `kernal_name<<<N, 1>>>(args);`
- Use `blockIdx.x` to get the ID

Launching A Kernel: The Hard Way

- Write a kernel (__global__)
- Run the kernel:
  - `kernal_name<<<grid_size, block_size>>>(args);`
- But how to set the grid_size / block_size?

Grid, Block, Thread

- Up to 3 dimensions
- All thread in a block can share memory



See [cuda-block-thread.cu].

Device Information

- deviceQuery for you as a test
- Check your CUDA environment or block/grid size

```
salloc -N1 -t3
srun /pvfsmnt/device-query
```

See [cuda-query/].

```
 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "NVIDIA GeForce RTX 2080 Ti"
  CUDA Driver Version / Runtime Version          11.4 / 11.4
  CUDA Capability Major/Minor version number:    7.5
  Total amount of global memory:                 11019 MBytes (11554717696 bytes)
  (068) Multiprocessors, (064) CUDA Cores/MP:    4352 CUDA Cores
  GPU Max Clock rate:                            1620 MHz (1.62 GHz)
  Memory Clock rate:                             7000 Mhz
  Memory Bus Width:                              352-bit
  L2 Cache Size:                                 5767168 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total shared memory per multiprocessor:        65536 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  1024
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 3 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device supports Managed Memory:                Yes
  Device supports Compute Preemption:            Yes
  Supports Cooperative Kernel Launch:            Yes
  Supports MultiDevice Co-op Kernel Launch:      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 175 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 11.4, CUDA Runtime Version = 11.4, NumDevs = 1
Result = PASS
```

Block & Grid Sizes

- Maximum number of threads per block: 1024
- Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
- Max dimension size of a grid size (x,y,z): (2147483647, 65535, 65535)

See Also: OpenMP

- An Overview of OpenMP
  - Light-blue code blocks (`!$omp`) are for Fortran, discard them
- A “Hands-on” Introduction to OpenMP
- OpenMP: Execution Environment Routines

See Also: CUDA

- CSC3150 Tutorials
- CUDA Runtime API
- CUDA Thread Indexing Cheatsheet
- CUDA documentation
- Supported GPU List
- CUDA Programming Guide

That’s All!

- Check out the slides & code on:
  - Blackboard, or
  - https://csc4005-tut-slides.netlify.app/07/, or
  - https://csc4005-tut-slides.pages.dev/07/