Basic Data Type

Jump to bottom

zoran edited this page on Sep 9 · 1 revision

Introduction

In the c programming language, data types are declarations for memory locations or variables that determine the characteristics of the data that may be stored and the methods (operations) of processing that are permitted involving them.

Data types are important. Arithmetic operations usually require the operands to share the same data type to produce a meaningful result. In Python, programmers tend to ignore data types since compiler will convert the data type for them, and this also happens in language C.

However, data types are extremely important in GPU parallel computing because compilers will **not** convert data types when one transmits data between CPU and GPU. One have to carefully check that the data declarations in both C and Python conform to each other, otherwise data will be lost and thread/block indexes will be in a mess.

Built-in Data Types

We care built-in data types most since in our course we will not create compound data type. (No structure, no class)

Here are main basic data types:

Type	Explanations
char	Smallest addressable unit of the machine that can contain basic character set. It is an integer type. Actual type can be either signed or unsigned. Usually It contains 8 bits.
signed char	Of the same size as char, but guaranteed to be signed. Capable of containing at least the [−127, +127] range.

Type	Explanations
unsigned char	Of the same size as char, but guaranteed to be unsigned. Contains at least the [0, 255] range.
short int	Short signed integer type. Capable of containing at least the [−32,767, +32,767] range; thus, it is at least 16 bits in size. The negative value is −32767 (not −32768) due to the one's-complement and sign-magnitude representations allowed by the standard, though the two's-complement representation is much more common.
unsigned short	Short unsigned integer type. Contains at least the [0, 65,535] range.
int	Basic signed integer type. Capable of containing at least the [−32,767, +32,767] range; thus, it is at least 16 bits in size.
unsigned int	Basic unsigned integer type. Contains at least the [0, 65,535] range.
long int	Long signed integer type. Capable of containing at least the [−2,147,483,647, +2,147,483,647] range; thus, it is at least 32 bits in size.
unsigned long	Long unsigned integer type. Capable of containing at least the [0, 4,294,967,295] range.
long long int	Long long signed integer type. Capable of containing at least the [−9,223,372,036,854,775,807, +9,223,372,036,854,775,807] range;thus, it is at least 64 bits in size.
unsigned long long	Long long unsigned integer type. Contains at least the [0, +18,446,744,073,709,551,615] range.
float	Real floating-point type, usually referred to as a single-precision floating-point type. Actual properties unspecified (except minimum limits), however on most systems this is the IEEE 754 single-precision binary floating-point format (32 bits). This format is required by the optional Annex F "IEC 60559 floating-point arithmetic".
double	Real floating-point type, usually referred to as a double-precision floating-point type. Actual properties unspecified (except minimum limits), however on most systems this is the IEEE 754 double-precision binary floating-point format (64 bits). This format is required by the optional Annex F "IEC 60559 floating-point arithmetic".
long double	Real floating-point type, usually mapped to an extended precision floating-point number format. Actual properties unspecified. It can be either x86 extended-precision floating-point format (80 bits, but typically 96 bits or 128

Type	Explanations
	bits in memory with padding bytes), the non-IEEE "double-double" (128 bits), IEEE 754 quadruple-precision floating-point format (128 bits), or the same as double. See the article on long double for details.

Data Type Match

Parallel computing requires exact data type matching. That is, for example, when a variable in GPU is declared as `int`, the CPU has to transmit a exact `int` data with the same bits length to it. `short int`, `long int`, `unsigned int` data will **not** work.

However, the bits length of data type is not strictly defined in C standard. Take `int` for example, it could be 32 bits, or in modern CPU, it could be 64 bits. In GPU, usually it is 32 bits. For `float`, it is usually 32 bits in both CPU and GPU in C, but it is 64 bits in Python. It is unnecessary to remember all this details, the important thing is to make sure the data type is the same. Always use `sizeof` function to check the number of bits of a built-in data type before you run your program.

What if data type is not matched?

If bit length is mismatched, then the threads have to be re-indexed. Take `int` again for example. If a variable in GPU is defined as `int` (say it's 32 bits), but CPU transmit `long int` data (say it's 64 bits) to it, then two threads represent this data instead of one, with one thread represents the lower 32 bits and other represents higher 32 bits. However, this is danger because the number of threads may not be enough to hold the `long int` data (after all, you told GPU it is a `int`) and leading to 'out of bounds' error. If a variable in GPU is defined as `long int` but CPU transmit `int` data to it, then a thread will represent two variables, with the higher 32 bits repenting one variable and lower 32 bits representing another.

If unsigned/signed is mismatched, then the data has to be re-organized. Take `int` for example, if CPU transmits a `int` to an `unsigned int`, then you have to negative each bit in that variable and then plus 1 (if singed number follows one's complement). Similar transform is needed if CPU transmits a `unsigned int` to a `int`.

However, you may find a way to correct `int` mismatching, but for `float` mismatch, it is much more complicated, let aloe complete mismatch (e.g., `int` matches with `float`).

Pages 18

E4750 Course Wiki Home

1. Home Page
2. Tutorials

- Google Cloud
  - Google Cloud VM Setup
  - GUI Installation
- Code
  - Python
  - PyCUDA Tutorial
  - Indexing in CUDA & OpenCL

3. Concepts and Additional How-Tos

- CUDA Profiling
- CUDA Cores v. Threads
- Data Types
- Timing execution in PyOpenCL

4. Assignments (distributed from the Code section)
   1. Assignment 1
   2. Assignment 2

Clone this wiki locally

```
https://github.com/eecse4750/e4750_2021Fall_students_repo.wiki.git
```