Com 4521 Parallel Computing with GPUs: Lab 01

Spring Semester 2019

Dr Mozhgan Kabiri Chimeh

Lab Assistants: Robert Chisholm, John Charlton

Department of Computer Science, University of Sheffield

Learning Outcomes

- Understand how to compile and execute a simple C program in Visual Studio
- Understand error outputs and how to fix them
- Understand how to link an external C library
- Understand file and string manipulations
- Understand how to manually allocate and free memory
- Understand how to read from a binary file
- Understand how to implement a linked list data structure in C.

Amendments

- 07/02/2018 – Exercise 2.5 clarified the form of the linear generator.

Lab Register

The lab register must be completed by every student following the completion of the exercises. You should complete this when you have completed the lab including reviewing the solutions. You are not expected to complete this during the lab class but you should complete it by the end of the teaching week.

Lab Register Link: https://goo.gl/0r73gD

Exercise 1

Add a new Windows 32 console project to the solution called '`Lab01_Exercise01`' by right clicking on the solution and selecting `Add->New Project...` To select it as the project which we would like to execute right click on the project and select '`Set as Startup Project`'. Add a new source file called '`exercise1.c`' and copy the contents from our hello world example. You can remove the hello world print command. Make a copy of the provided '`random.h`' file and place it in your projects source directory. Add the `random.h` header file to the project by right clicking on the project in the solution explorer and selecting `Add->Existing item`.

We are going to create a program to create a list of normalised random integers.

1.1 Create a pre-processor definition called `NUM_VALUES` and assign it the value of `250`.
1.2 Declare a global signed 32-bit integer **array** called '`value`' using your pre-processor definition to define the array size.
1.3 Define a local unsigned 32-bit integer variable called '`sum`' in the main function capable of holding only positive values and initialise it to `0`.

1.4 Define a local variable `i` in the main function using a data type which can hold values in the range of `0-255`, initialise it to `0`.
*Note: If you **declare** your variables and set their values (i.e. **define** them) on separate lines then you will need to ensure that both your declaration and definition appear before any expressions (e.g. `sum=0;`).*

1.5 We need to make a call to a function declared in a header file `random.h`, include the header file and call `init_random` in the `main` function.

1.6 Write a simple for loop (using the integer `i` as a counter) in the range of `0` and `NUM_VALUES`. Within the loop make a call to the function `random_ushort` and save the value in the values array at index `i`. Within the loop create a print statement to the console which outputs in a single line the value of `i` and the value you have stored in the array. We can use this to debug the output.

1.7 The `random_ushort` function contains an implicit cast from `int` to `unsigned short`. Modify this so that it uses an explicit cast. This won't change the program but is good practice.

1.8 Modify your loop by commenting out the debug statement and summing the value into the variable '`sum`'. Output the `sum` value after the loop has returned. What is the sum? It should be `4125024`. Add a new local variable '`average`' using an appropriate data type. Calculate and store the average value of the random numbers.

1.9 Normalise the random numbers by subtracting the average. Calculate the minimum and maximum values. A ternary `if` operator

```
(conditional expression)? expression_t : expression_f ;
```

can be used to return either `expression_t` if the conditional is true or `expression_f` is the condition is false. E.g. `int x = (a == 0)? 123 : 567;` Use this shorthand notion in calculating the min and max values. Print the average, min and max values along with the sum. You should get the following values.

```
Sum=4125024
Average=16500
Min=-16247
Max=16221
```

Exercise 2

We are now going to extend the previous exercise by implementing a better random function. Add a new Windows 32 console project to the solution called '`Lab01_Exercise02`'. Make sure that the folder location is the root of your solution (and not one of the previous projects). Set the new project as the start-up project, so when you run the program it won't run the previous project. Make a copy of the source and header file from the previous exercise (rename `exercise1.c` to `exercise2.c`) and move them the new projects source folder. Add them to the project by selecting `Add->Existing Items`.

The problem with the existing rand function is that it only returns values in the range of `0-32767` (the positive range of a signed short) despite returning a 32 bit integer. This is due Microsoft preserving backwards compatibility with code utilising the function when it was first implemented (and when 16 bit integers were more common). This is a "feature" of the msvc runtime. You will find in Linux that rand returns a full 32 bit number.

2.1 Let us start by separating the random function definitions into a header and separate source module. Create a new file in the project called `random.c`. Move the `init_random` and `random_ushort` function definitions into the new source module. Move the inclusion of `stdlib.h` to the new source module and include `random.h` in `random.c`. The `random.h` file should now only contain the seed. Build the application. The compiler should give a warning that the two functions are undefined.

2.2 Modify `random.h` by adding appropriate function declarations. If you don't include the `extern` keyword then it will be implicitly defined by the compiler (as all globals are `extern` by default). It is good practice to include it. The project should now build without errors.

2.3 We are now going to implement simple linear congruential generator (wiki link). This works by advancing a seed by a simple multiplication and addition. Define the parameters `RANDOM_A` and `RANDOM_C` in `random.h` using a pre-processor macro. Set their values to `1103515245` and `12345` respectively.

2.4 Create a 32 bit unsigned global integer variable called '`rseed`' in `random.c`. Modify `init_random` to set the value of `rseed` to `RAND_SEED`.

2.5 Create a function declaration (in `random.h`) and definition (in `random.c`) for a function `random_uint` returning an unsigned int. Implement a linear generator using the equation

$$x_{(n+1)} = A*x_{(n)} + C$$

where `x` can use the variable '`rseed`' (with an initial value x_0 = `RAND_SEED` as per exercise 2.4), and `A` and `C` are your parameters.

2.6 Replace the call to `random_ushort` with a call to `random_uint` in `exercise2.c`. Our variable sum is now too small to hold the summed values. Modify it to a 64 bit unsigned integer and ensure it is printed to the console correctly. Modify the type of the `values` array to a 64 bit signed integer also. You will also need to modify the calculation of the average, min and max value by adding an explicit cast. Ensure that your `printf` formats are correct for the data types. What is the sum, average, min and max? They should be;

```
Sum=524529029501
Average=2098116118
Min=-2093214053
Max= 2107382890
```

Exercise 3

We are now going to extend the previous exercise by implementing a floating point random function. Add a new Windows 32 console project to the solution called '`Lab01_Exercise03`'. Make sure that the folder location is the root of your solution (and not one of the previous projects). Set the new project as the start-up project. Make a copy of the source and header file from the previous exercise (rename `exercise3.c` to `exercise4.c`) and move them the new projects source folder. Add them to the project by selecting `Add->Existing Items`.

3.1 Add a new function definition and declaration (`random_float`) returning a random float. This should be a value cast from the `random_uint` function. Modify the example so that floating point values are calculated for sum, average, min and max. Ensure that the values are printed with 0 decimal places. What is the sum, average, min and max? They should be;

```
Sum=524529139712
Average=2098116608
Min=-2093214592
Max=2170532608
```

Exercise 4

You are going to create a calculator which takes input from the command line.

Add a new Windows 32 console project to the solution called 'Lab01_Exercise04' by right clicking on the solution and selecting Add->New Project... To select it as the project which we would like to execute right click on the project and select set it as the start-up project". Make a copy of the file exercise4.c (which is provided for you) and place it in your projects source directory. Add the file to the project by selecting Add->Existing Items.

The source file contains the basic structure of a simple command line calculator which will understand the following basic commands "add N", "sub N", "mul N", "div N" and "exit", where N is a floating point value.

4.1 Complete the while loop by adding character sequentially to the buffer.
4.2 Implement a check to ensure that you don't write past the end of the buffers limits. Writing passed the end of an array is called an overflow. When an potential overflow is detected write an error message to stderr using fprintf and then call exit(1) to force the program to terminate early.
4.3 Ensure that once the while loop has exited the buffer is correctly terminated with the string termination character.
4.4 Use the strcmp function to test if the line reads "exit". If it does, then readLine should return 0 otherwise it should return 1. Test the program. It should quit when a user enters "exit" otherwise it should print "Unknown command".
4.5 Modify the while loop in the main function. Check that the line contains a three characters followed by a space. You can use the isalpha function from 'ctype.h' to check that a character is a letter. If the line does not meet this criteria, then output an error "Incorrect command format" to stderr and use continue to begin the loop again.
4.6 Assuming the criteria for 4.5 is met then use sscanf to extract the 3 character command and the floating point value from the buffer to command and a respectively.
    Note: You will need to pass in_value to sscanf prefixed with the & operator. E.g sscanf(…, &in_value).
4.7 Modify the condition false to check the command to see if it is equal to "add"
4.8 Create an else if condition for sub, mul and div. Test your program.
4.9 Add additional conditions using strncmp to test the first two letter of the command. If it is "ad" then output "Did you mean add?" Complete cases for "su", "mu", "di". Test your program.

Exercise 5

We are now going to extend the previous exercise by modifying the calculator so that it can read commands from a file. Add a new Windows 32 console project to the solution called 'Lab01_Exercise05'. Make sure that the folder location is the root of your solution (and not one of the previous projects). Set the new project as the start-up project. Make a copy of the source file from the previous exercise in the new projects source folder. Rename the source file from exercise4.c to exercise5.c. Add the source file to the project by selecting Add->Existing Items.

5.1 Modify the example so that it can read the provided 'commands.calc' file. You will need to implement the following;
5.1.1 Open and closing the file in read only mode

5.1.2 Modify the `readLine` function so that it reads form a file rather than the console. You should check for end of file character (`EOF`) and `return 0` if it is found. *Note: this behaviour requires any .calc files to have a blank line at the end of the file.*

5.1.3 Modify the `main` function. Incorrect commands or misspelt commands should cause a console error and immediate exit. The while loop should be silent (no console output) and only the final sum should be output to the console. The correct answer is `99.0000`.

Exercise 06

The purpose of this exercise is to modify some existing code to use pointers. Some sample code has been provided for you. The example code will read in a binary file which contains 4 records of information on students. The information consists of their forename, surname and average module mark. A `struct` has been defined to hold the student data, the format of this `struct` matches the `struct` used in the program which created the binary files used in the example.

1.1 Compile and execute the program. It should print out the information for 4 student's.

1.2 The `print_student` function is inefficient. It requires passing a structure (by value) which causes all of the data to be duplicated. Amend this so that the structure is passed as a reference. You will need to update both the `print_student` function declaration and definition.

1.3 The `main` function uses a statically defined array to hold our student data. Modify this code so that `students` is a pointer to a student `struct` and then manually allocate enough memory to read in the student records. Don't forget to also `free` the data at the end of the program.

Exercise 07

Copy your previous code from the first exercise into a new project called '`Lab01_Exercise07`'.

The student structure uses a statically defined, fixed length `char` array to hold both the forename and surname. This is OK but potentially wasteful when we deal with large records as much of the `char` array will be empty. The file *`students2.bin`* differs from the file used in the first exercise in that it uses dynamic length `char` arrays to hold strings. Both the forename and surname are written to the binary file in the following format;

```
unsigned int n, char[0], char[1], char[2], …., char[n]; e.g.
5, 'J', 'o', 'h', 'n', '\0'
```

Modify the `struct` definition so that forename and surname are pointers to `char`. Now update the code to read the student data. You will need to use `fread` to read the length of the forename (i.e. `n`). Hint: allocate memory for the forename (of length `n`) and then `fread` the forename, etc. Don't forget to also update your code to ensure that you `free` any memory you have allocated.

Exercise 08

Copy your previous code from the last exercise into a new project called '`Lab01_Exercise08`'.

Both the previous exercises assumed that we knew how many student records were stored in the binary data file. For the next exercise we will update our program to read, store and display an

arbitrary number of records. In order to do this we are going to use a linked list data structure. The *linked_list.h* header file contains very basic implementation of a generic linked list. The header file contains a structure `llitems` which defines a pointer to the previous and next item in the list.

1.1 The implementation of a linked list is incomplete. Complete the function `add_to_linked_list()` by implementing the following;
   1.1.1 Check that the `ll_end` item is in fact the end of the list (the next record should be `NULL`). If it is not the end then the function should return `NULL`.
   1.1.2 Add the item to the end of the linked list updating the old end of the linked list to reflect the addition.
   1.1.3 Return a pointer to the new end of the linked list.

1.2 In order to use the `print_items` function the function pointer `print_callback` must be set to a function with the following declaration;

```
void print_function(void *);
```

   You already have a function `print_student` which could be used but this function accepts a `const` pointer to student structure. Assign to the `print_callback` function pointer, your `print_student` function using an explicit cast. You must be careful about your use of brackets here.

1.3 Update your code to read in *students2.bin* by creating a linked list of student records. You will need a pointer to mark both the start and end of the linked list. To test if your stream is at the end of a file (i.e. it has read the last record) you should check the return value of `fread` (if less than the requested number of items are returned this indicates the end of the file). You should use the `create_linked_list` and `add_to_linked_list` functions. You can use the `free_linked_list` function to `free` your linked list but be careful as this won't free the records which the linked list points to.