cuAutotools

1.0

Generated by Doxygen 1.8.13

Contents

Chapter 1

File Index

1.1 File List

Here is a list of all files with brief descriptions:

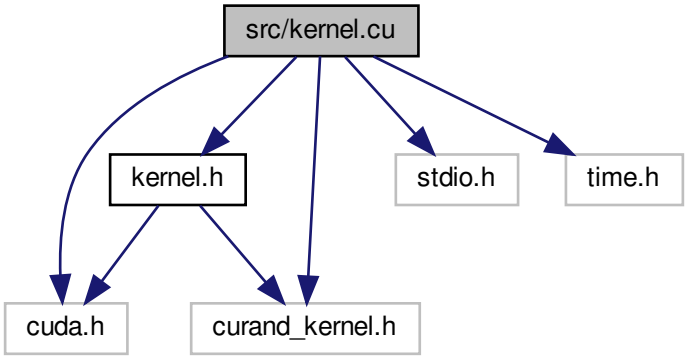
Chapter 2

File Documentation

2.1 src/kernel.cu File Reference

```
#include "kernel.h"
#include <stdio.h>
#include <time.h>
#include <cuda.h>
#include <curand_kernel.h>
```

Include dependency graph for kernel.cu:



Functions

- __global__ void setup_prng (curandState *state, unsigned long long seed, unsigned int qtd)

  Starts the PRNG for each curandState with seed and a different starting id.
- __global__ void create_random_data (curandState *state, unsigned int qtd_states, float *data, unsigned int qtd_data)

  Create Random qtd_data random floats and store the average on the data array.
- __global__ void dummy_kernel (void)

A simple dummy kernel that just prints a Hello World.

- int call_rand_kernel (void)

  Calls a kernel to create 4.0 MB of random float numbers and print the average of those numbers.
- void call_dummy_kernel (void)

  Call a dummy kernel that simply prints a Hello World message from the GPU.
- void start_device (void)

  Start the device with id 0.
- void reset_device (void)

  Reset the started device and clear it for usage.

2.1.1 Function Documentation

2.1.1.1 call_dummy_kernel()

```
void call_dummy_kernel (
            void  )
```

Call a dummy kernel that simply prints a Hello World message from the GPU.

2.1.1.2 call_rand_kernel()

```
int call_rand_kernel (
            void  )
```

Calls a kernel to create 4.0 MB of random float numbers and print the average of those numbers.

This works by creating the same amount of curandState (default to XORXOW generator) and calling the kernel 1000 times. On each kernel we create the average call and store them on the data array passed. Once all kernels are finished we create the average of it.

Note

Honestly this is overkill and a completelly bogus scenario, we could create for instance 512 generators, create the random data on each thread and just return the value created, and then finding the average. But since this is just to show how to run a more complex kernel, we'll leave it like this.

Returns

0 on success, -1 on error

2.1.1.3 create_random_data()

```
__global__ void create_random_data (
            curandState * state,
            unsigned int qtd_states,
            float * data,
            unsigned int qtd_data )
```

Create Random qtd_data random floats and store the average on the data array.

Parameters

state	The already initialized array of curandState states
qtd_states	The amount of states (this must be the same size of data)
data	A float array initialized on the GPU to store the results
qtd_data	How may samples we will create

2.1.1.4 dummy_kernel()

```
__global__ void dummy_kernel (
             void  )
```

A simple dummy kernel that just prints a Hello World.

2.1.1.5 reset_device()

```
void reset_device (
             void  )
```

Reset the started device and clear it for usage.

2.1.1.6 setup_prng()

```
__global__ void setup_prng (
             curandState * state,
             unsigned long long seed,
             unsigned int qtd )
```

Starts the PRNG for each curandState with seed and a different starting id.

Parameters

state	A pointer to a device memory containing an array of curandState values
seed	The seed, this should be different on each execution
qtd	The size of the state array

2.1.1.7 start_device()

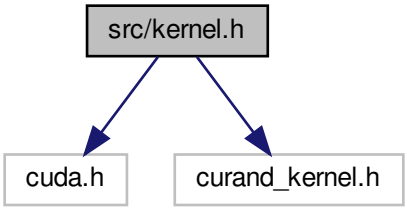
```
void start_device (
             void  )
```

Start the device with id 0.

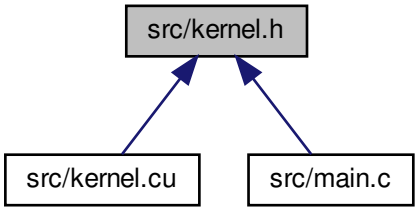
2.2 src/kernel.h File Reference

```
#include <cuda.h>
#include <curand_kernel.h>
```

Include dependency graph for kernel.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define CUDA_CALL(call)

Functions

- __global__ void setup_prng (curandState *state, unsigned long long seed, unsigned int qtd)

  Starts the PRNG for each curandState with seed and a different starting id.
- __global__ void create_random_data (curandState *state, unsigned int qtd_states, float *data, unsigned int qtd_data)

  Create Random qtd_data random floats and store the average on the data array.

- __global__ void dummy_kernel (void)

  A simple dummy kernel that just prints a Hello World.
- void call_dummy_kernel (void)

  Call a dummy kernel that simply prints a Hello World message from the GPU.
- int call_rand_kernel (void)

  Calls a kernel to create 4.0 MB of random float numbers and print the average of those numbers.
- void start_device (void)

  Start the device with id 0.
- void reset_device (void)

  Reset the started device and clear it for usage.

2.2.1 Macro Definition Documentation

2.2.1.1 CUDA_CALL

```
#define CUDA_CALL(
               call )
```

Value:

```
{                                                                                   \
            const cudaError_t error=call;                                           \
            if(error != cudaSuccess){                                               \
                        printf("Error: %s:%d, ", __FILE__, __LINE__);               \
                        printf("code:%d, reason: %s\n", error, cudaGetErrorString(error));  \
                        exit(1);                                                    \
                    }                                                               \
}
```

2.2.2 Function Documentation

2.2.2.1 call_dummy_kernel()

```
void call_dummy_kernel (
             void  )
```

Call a dummy kernel that simply prints a Hello World message from the GPU.

2.2.2.2 call_rand_kernel()

```
int call_rand_kernel (
            void  )
```

Calls a kernel to create 4.0 MB of random float numbers and print the average of those numbers.

This works by creating the same amount of curandState (default to XORXOW generator) and calling the kernel 1000 times. On each kernel we create the average call and store them on the data array passed. Once all kernels are finished we create the average of it.

Note

Honestly this is overkill and a completelly bogus scenario, we could create for instance 512 generators, create the random data on each thread and just return the value created, and then finding the average. But since this is just to show how to run a more complex kernel, we'll leave it like this.

Returns

0 on success, -1 on error

2.2.2.3 create_random_data()

```
__global__ void create_random_data (
             curandState * state,
             unsigned int qtd_states,
             float * data,
             unsigned int qtd_data )
```

Create Random qtd_data random floats and store the average on the data array.

Parameters

state	The already initialized array of curandState states
qtd_states	The amount of states (this must be the same size of data)
data	A float array initialized on the GPU to store the results
qtd_data	How may samples we will create

2.2.2.4 dummy_kernel()

```
__global__ void dummy_kernel (
             void  )
```

A simple dummy kernel that just prints a Hello World.

2.2.2.5 reset_device()

```
void reset_device (
             void  )
```

Reset the started device and clear it for usage.

2.2.2.6 setup_prng()

```
__global__ void setup_prng (
             curandState * state,
             unsigned long long seed,
             unsigned int qtd )
```

Starts the PRNG for each curandState with seed and a different starting id.

Parameters

state	A pointer to a device memory containing an array of curandState values
seed	The seed, this should be different on each execution
qtd	The size of the state array

2.2.2.7 start_device()

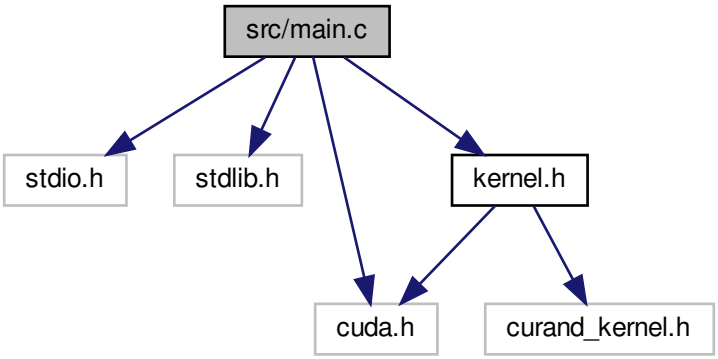
```
void start_device (
             void  )
```

Start the device with id 0.

2.3 src/main.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <cuda.h>
#include "kernel.h"
```

Include dependency graph for main.c:



Functions

- int main (int argc, char **argv)

2.3.1 Function Documentation

2.3.1.1 main()

```
int main (
            int argc,
            char ** argv )
```

Index