

## Additional documentation on the implementation of kernels under the RacEr architecture

This work is an addition to the documentation based on the RacEr architecture developed by Vividspark from a Manycore architecture.

### API RacEr

#### Racer Function in kernel file

RacEr Kernel Function	Definition
<code>RacEr_set_tile_x_y()</code>	Sets RacEr_X and RacEr_Y to the X and Y coordinate of the tile.
<code>RacEr_x_y_to_id(int x, int y)</code>	Calculates tile's flat id using its (x,y) coordinates.
<code>RacEr_id_to_x(int x)</code>	Calculates tile's x coordinate using its flat id.
<code>RacEr_id_to_y(int y)</code>	Calculates tile's y coordinate using its flat id.
<code>RacEr_remote_ptr (int x, int y, void *addr)</code>	Forms a remote address by taking in x and y coordinates of remote tile and the address to local variable. Used in RacEr_remote_store and RacEr_remote_load.
<code>RacEr_remote_store (int x, int y, void *addr, int val)</code>	Stores val to the local address addr in the memory space of the tile at (x,y)
<code>RacEr_remote_store_uint8 (int x, int y, void *addr, unsigned char val)</code>	Stores the 1-byte val to the local address addr in the memory space of the tile at (x,y)
<code>RacEr_remote_store_uint16 (int x, int y, void *addr, unsigned short val)</code>	Stores the 2-byte val to the local address addr in the memory space of the tile at (x,y)
<code>RacEr_remote_load (int x, int y, void *addr)</code>	Loads from the local address addr in the memory space of the tile at (x,y)
<code>RacEr_dram_ptr (void *addr)</code>	Forms a pointer to an element on the DRAM attached to the bottom of tile's column using the local address.
<code>RacEr_dram_load (void *addr, val)</code>	Loads from DRAM into val by using RacEr_dram_ptr.
<code>RacEr_dram_store (void *addr, val)</code>	Stores val into dram by using RacEr_dram_ptr.
<code>RacEr_tilegroup_ptr (void *addr, int index)</code>	Takes in the local address of tilegroup-shared memory, and the array index. Calculates the coordinates of the tile holding that index, and returns a RacEr_remote_ptr to that element.

<code>RacEr_tilegroup_load (void *addr, int index, val)</code>	Loads from tilegroup-shared memory into val by taking in its local address and index, using RacEr_tilegroup_ptr.
<code>RacEr_tilegroup_store (void *addr, int index)</code>	Stores local val to tilegroup-shared memory by taking in its local address and index, using RacEr_tilegroup_ptr.
<code>RacEr_remote_control_store (int x, int y, void *addr, val)</code>	Remote stores the value into the instruction memory of tile (x,y) using local address.
<code>RacEr_remote_freeze (int x, int y)</code>	Starts the execution of tile (x,y) using RacEr_remote_control_store.
<code>RacEr_remote_unfreeze (int x, int y)</code>	Stops the execution of tile (x,y) using RacEr_remote_control_store.
<code>INIT_TILE_GROUP_BARRIER (ROW_BARRIER_NAME, COL_BARRIER_NAME, int x_cord_start, int x_cord_end, int y_cord_start, int y_cord_end)</code>	Initializes parameters for a barrier instruction for all tiles within tilegroup using the start and end coordinates of the tilegroup.
<code>RacEr_tile_group_barrier (ROW_BARRIER_NAME, COL_BARRIER_NAME)</code>	Synchronizes all tiles within the tilegroup by taking in the row and column barrier names generated by INIT_TILE_GROUP_BARRIER .
<code>RacEr_wait_while(int cond)</code>	Wait for condition to be true
<code>poll_range(int range, unsigned char *ptr_value)</code>	Check if no 0 value in ptr_value
<code>RacEr_print_float(posit f)</code>	Print posit value

## Racer Function in main file

Racer main function	Definition
<code>RacEr_mc_device_malloc</code> (RacEr_mc_device_t *device, int size, void *src)	allocate memory size on targeted device accessible with the src
<code>RacEr_mc_device_memcpy</code> (RacEr_mc_device_t *device, void *dst, void *src, int size, hb_mc_memcpy_kind kind)	Copy memory size from src to dst on the device . You must specify how : HB_MC_MEMCPY_TO_DEVICE or HB_MC_MEMCPY_TO_HOST
<code>RacEr_mc_device_memset</code> (RacEr_mc_device_t *device, void *src, int value, int size)	Set memory size for the src to value on the device
<code>RacEr_printf</code> (format, ...)	print
<code>RacEr_mc_kernel_enqueue</code> (RacEr_mc_device_t *device, RacEr_mc_dimension_t grid_dim, RacEr_mc_dimension_t tg_dim, char *function_name, int function_argc, int *function_argv)	Enqueue function with a function_name on device . Specify how the grid_dim and tg_dim . Pass the function parameter throw function_argc and function_argv in a  <code>int kernel_args[] = {function_arg, ...}</code>
<code>RacEr_mc_device_tile_groups_execute</code> (RacEr_mc_device_t *device)	Execute the function enqueue on the device
<code>RacEr_mc_device_init</code> (RacEr_mc_device_t *device, char *name, int device_id)	initialise the device with the name on a specific device_id
<code>RacEr_mc_device_program_init</code> (RacEr_mc_device_t *device, char *bin_path, char *allocator, int device_id)	initialise the program on the device with the bin_path program. It requires an allocator and the device_id
<code>argp_parse</code> (&argp_path, int argc, char *argv, 0, 0, struct arguments_path *args)	Function to parse args from Command line and store it in args which is an arguments_path type
<code>RacEr_mc_device_finish</code> (RacEr_mc_device_t *device)	Terminate the simulation on the device
<code>RacEr_mc_device_free</code> (RacEr_mc_device_t *device, void *src)	free allocated memory on device

## RacEr Type

RacEr Macro	Definition
<pre>RacEr_mc_dimension_t tg_dim = { .x = x, .y = y } RacEr_mc_dimension_t grid_dim = { .x = value / block_size_x, .y = value / block_size_y }</pre>	2-dimension type for <code>tg_dim</code> and <code>grid_dim</code>
<pre>enum hb_mc_memcpy_kind { HB_MC_MEMCPY_TO_DEVICE, HB_MC_MEMCPY_TO_HOST }</pre>	Enum to define the copy mode
<pre>RacEr_mc_device_t device</pre>	Device type
<pre>struct arguments_path args = {name, path}</pre>	Struct to store function <code>name</code> and <code>path</code>

## RacEr Macro

RacEr Macro	Definition
<pre>#define ALLOC_NAME "default_allocator"</pre>	Definition of the default_allocator
<pre>#define HB_MC_SUCCESS      (0) #define HB_MC_FAIL        (-1) #define HB_MC_TIMEOUT      (-2) #define HB_MC_UNINITIALIZED (-3) #define HB_MC_INVALID      (-4) #define HB_MC_INITIALIZED_TWICE (-4) // same as invalid #define HB_MC_NOMEM        (-5) #define HB_MC_NOIMPL       (-6) #define HB_MC_NOTFOUND     (-7) #define HB_MC_BUSY         (-8) #define HB_MC_UNALIGNED    (-9)</pre>	Errno macro
<pre>#define RacEr_TILE_GROUP_X_DIM RacEr_tiles_X int start_x = __RacEr_tile_group_id_x * block_size_x int end_x = start_x + block_size_x</pre>	Get a thread action field

## **Compilation**

### **Needed files**

- a
- b

### **Commands**

# Migration of Cuda code to RacEr

## Vector reduce kernel example

### RacEr

```
#include "RacEr_manycore.h"
#include "RacEr_set_tile_x_y.h"

#define RacEr_TILE_GROUP_X_DIM RacEr_tiles_X
#define RacEr_TILE_GROUP_Y_DIM RacEr_tiles_Y
#include "RacEr_tile_group_barrier.h"
INIT_TILE_GROUP_BARRIER (r_barrier, c_barrier, 0, RacEr_tiles_X - 1, 0,
                          RacEr_tiles_Y - 1);

int __attribute__((noinline))
kernel_float_vec_dotprod (posit *A, posit *B, int block_size_x)
{
    int start_x = block_size_x
                  * (__RacEr_tile_group_id_y * __RacEr_grid_dim_x
                     + __RacEr_tile_group_id_x);
    posit A_a = 0.0, B_b = 0.0;
    for (int iter_x = __RacEr_id; iter_x < block_size_x;
         iter_x += RacEr_tiles_X * RacEr_tiles_Y)
    {
        B_b = A[iter_x] + B_b;
    }
    B[__RacEr_id] = B_b;

    // RacEr_tile_group_barrier (&r_barrier, &c_barrier);
    // for (int i = 0; i < RacEr_tiles_X * RacEr_tiles_Y; i++)
    // {
    //     A_a = A_a + B[i];
    // }
    RacEr_tile_group_barrier (&r_barrier, &c_barrier);
    B[__RacEr_id] = A_a;

    return 0;
}
```

### CUDA

```
__global__ void reduce(float *input, float *output, int n) {
    extern __shared__ float sharedData[];
    unsigned int tid = threadIdx.x;
    unsigned int i = blockIdx.x * blockDim.x * 2 + threadIdx.x;

    if (i < n) {
        sharedData[tid] = input[i] + (i + blockDim.x < n ? input[i + blockDim.x] : 0);
    } else {
        sharedData[tid] = 0;
    }
    __syncthreads();

    for (unsigned int s = blockDim.x / 2; s > 0; s >>= 1) {
        if (tid < s) {
            sharedData[tid] += sharedData[tid + s];
        }
        __syncthreads();
    }

    if (tid == 0) {
        output[blockIdx.x] = sharedData[0];
    }
}
```

## Vector reduce main example

### RacEr

```
#include "cuda_tests.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/types.h>

#define ALLOC_NAME "default_allocator"

int
kernel_float_vec_dotprod (int argc, char *argv[])
{
    char *bin_path, *test_name;
    struct arguments_path args = { NULL, NULL };

    argp_parse (&argp_path, argc, argv, 0, 0, &args);
    bin_path = args.path;
    test_name = args.name;

    int rc, mismatch = 0, n = 1000, size = sizeof (float) * n;

    RacEr_mc_device_t device;
    rc = RacEr_mc_device_init (&device, test_name, 0);
    rc = RacEr_mc_device_program_init (&device, bin_path, ALLOC_NAME, 0);

    float a[n], b[n];

    RacEr_mc_eva_t a_device, b_device;
    rc = RacEr_mc_device_malloc (&device, size, &a_device);
    rc = RacEr_mc_device_malloc (&device, size, &b_device);

    for (int i = 0; i < n; i++)
    {
        a[i] = (float)drand48 (); // Large
    }

    void *dst = (void *)((intptr_t)a_device);
    void *src = (void *)&a[0];

    rc = RacEr_mc_device_memcpy (&device, dst, src, size,
                                HB_MC_MEMCPY_TO_DEVICE);
    uint32_t block_size_x = n;
    RacEr_mc_dimension_t tg_dim = { .x = 2, .y = 2 };
    RacEr_mc_dimension_t grid_dim = { .x = 1, .y = 1 };

    int cuda_argv[3] = { a_device, b_device, block_size_x };
    rc = RacEr_mc_kernel_enqueue (&device, grid_dim, tg_dim,
                                   "kernel_float_vec_dotprod", 3, cuda_argv);

    rc = RacEr_mc_device_tile_groups_execute (&device);

    src = (void *)((intptr_t)b_device);
    dst = (void *)&b[0];
    rc = RacEr_mc_device_memcpy (&device, dst, src, size, HB_MC_MEMCPY_TO_HOST);
}

int
main (int argc, char *argv[])
{
    RacEr_pr_test_info ("Vector reduce \n");
    kernel_float_vec_dotprod (argc, argv);
    return 0;
}
```

### CUDA



```
#include <cuda_runtime.h>
#include <cuda_runtime.h>

int
main (int argc, char *argv[])
{
    float *f1, result *d_input, d_output;

    int n = 1000;
    f1 = (float *)malloc (sizeof (float) * n);

    for (int i = 0; i < n; i++)
        f[i] = (float)drand48 ();

    int numBlocks = (n + BLOCK_SIZE - 1) / BLOCK_SIZE;

    cudaMalloc (&d_input, n * sizeof (float));
    cudaMalloc (&d_output, numBlocks * sizeof (float));

    cudaMemcpy (d_input, h_input, n * sizeof (int), cudaMemcpyHostToDevice);

    reduce<<<numBlocks, BLOCK_SIZE, BLOCK_SIZE * sizeof (float)>>> (
        d_input, d_output, n);

    while (numBlocks > 1)
    {
        n = numBlocks;
        numBlocks = (n + BLOCK_SIZE - 1) / BLOCK_SIZE;
        reduce<<<numBlocks, BLOCK_SIZE, BLOCK_SIZE * sizeof (float)>>> (
            d_output, d_output, n);
    }

    cudaMemcpy (&result, d_output, sizeof (float), cudaMemcpyDeviceToHost);

    cudaFree (d_input);
    cudaFree (d_output);

    return 0;
}
```