Accelerating DODGE BUBBLE using CUDA

CS6023: GPU programming course project (JAN-MAY 2020)

Sundar Raman P, EE17B069

Problem Statement & Results:

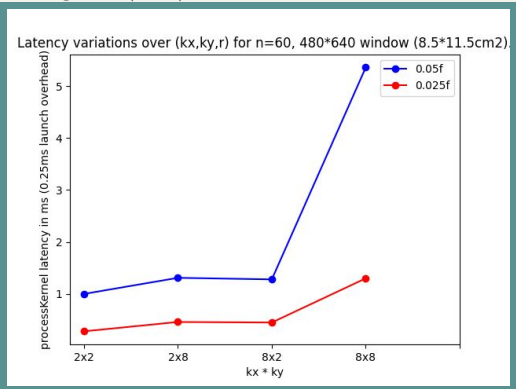
- Accelerate a 2D game where balls bounce into each other inside a window, using CUDA.

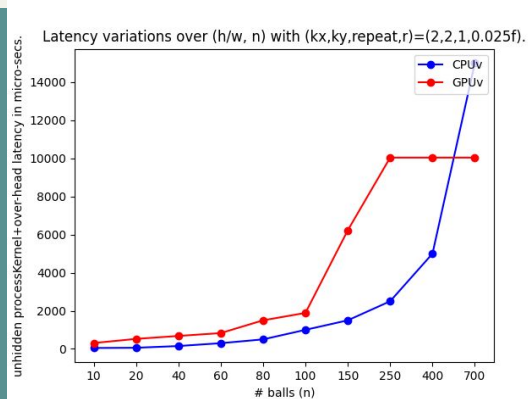
Key Idea: Split the window into square tiles, assign balls to tiles to perform *path* computations.

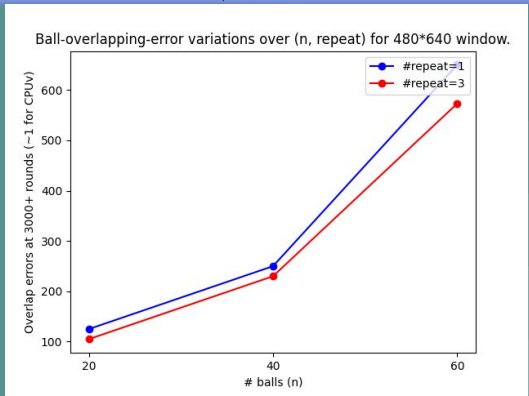
```
#procedure input: #balls (n); (kx, ky); ball radius (r)
device_kernel():
    i,j=tile co-ordinate based on thread ids
    if not_halo:
        worklist <- worklist_new  #stage1
        balls <- balls_new
        for b in worklist(i,j):
            for n in eight neighbouring_tiles of (i,j):
                if overlap(n,b):
                    if b is 0:
                        die()
                    naive_collider(n,b)  #updates position, velocity of 'balls'
    if first_thread_of_a_block:
        atomicInc(*counter, max_counter);  #init: counter=0, max_counter=total # blocks-1
    __syncthreads(); while(*counter);  #global barrier
    for b in worklist(i,j):  #stage2
        balls_new.position <- balls.position + (p==0)? keyboard_moves: balls.velocity
        balls_new.velocity <- balls.velocity
        reflect 'b' if it is at/outside boundary
        i_new, j_new <- new_tile_of_ball(b)
        atomically: worklist_new[i_new,j_new] = b

main():
    balls[0:n] <- initBalls()  #randomized positions, velocities
    glutInit(); glewInit()  #display, keyboard, timer, view-port inits
    while(True):
        if window_is_reshaped:
            tiles <- split window into rectangles based on (kx, ky, r)
            worklist <- assign balls to tiles based on their position
        device kernel(tiles assigned to GPU cores from threads->blocks in 2D) in stream1
        if approp. keyboard_input: convolution_kernel() in stream2 to modify background
        glutDraw(balls)
```

Pseudo-code







Data

Challenges faced - Solution:

- Parallelizing an inherently (exactly) ‘global’ problem - 2 approximate collision schemes!
- Approximately non-overlapping displays - Old, new; 2 stages & a simple global barrier.
- Passing ‘ball’ handling roles to neighbours - Update ‘*worklist*’ Atomically! (systolics)
- Old ‘graphics.h’ library - OpenGL (glew, glut) with modern APIs, interops, h/w archs., etc.
- Variety of background image data (RGBa) - Texture, depth binded to frame buffers
- Circles in openGL - Circles using triangle fans
- Dull background - Launch a (heavy) convolution kernel in a different stream
- (Background) memory-access bottleneck - 2D Texture buffer
- (Balls) memory-access bottleneck - Balls defined as Structure Of Arrays instead of AoS
- Persistent kernel issue - Multiple kernel launches in a stream without cudaDeviceSync()
- Lots of memory transfers between device, host - Pinned memory using cudaHostAlloc()
- nvprof profiler - nsight visual profiler
- Thread divergence due to conditionals - Ternary operator
- Consistency of implementation - Track ‘*energy*’ variation over time (-0.00001/3000 rounds)
- Get one ball to respond to user inputs - Keyboard handler
- Naive game playing agents - Devised simple-rule based agents
- Accessing common memory within a block - Profuse use of shared memory
- Nested for-loop structure inside kernel - Cache-efficient structuring of them

Imp. ref.