Design Report

A20402686

Conway's Game of Life-

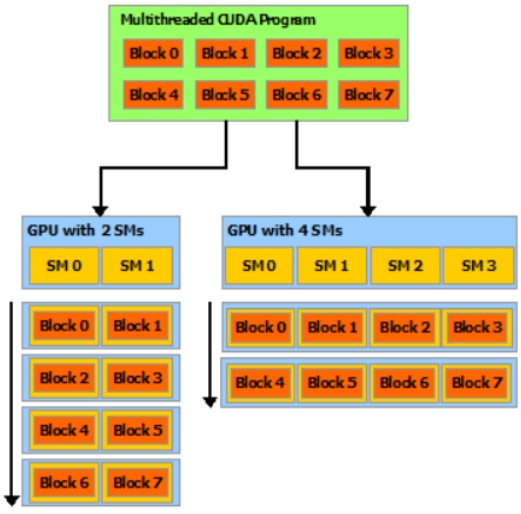
This is a zero-player game, meaning that its evolution is determined by its initial state, requiring no further input. One interacts with the Game of Life by creating an initial configuration and observing how it evolves, or, for advanced "players", by creating patterns with particular properties. The Game has been reprogrammed multiple times in various coding languages. The pattern can be evolved in many ways. There are many different types of patterns in the Game of Life like still life, oscillators, glider. This problem can be implemented parallelly using CUDA.

What is CUDA?

CUDA stands for Compute Unified Device Architecture. CUDA is a parallel computing platform and programming model that enables dramatic increases in computing performance by harnessing the power of the graphics processing unit (GPU).

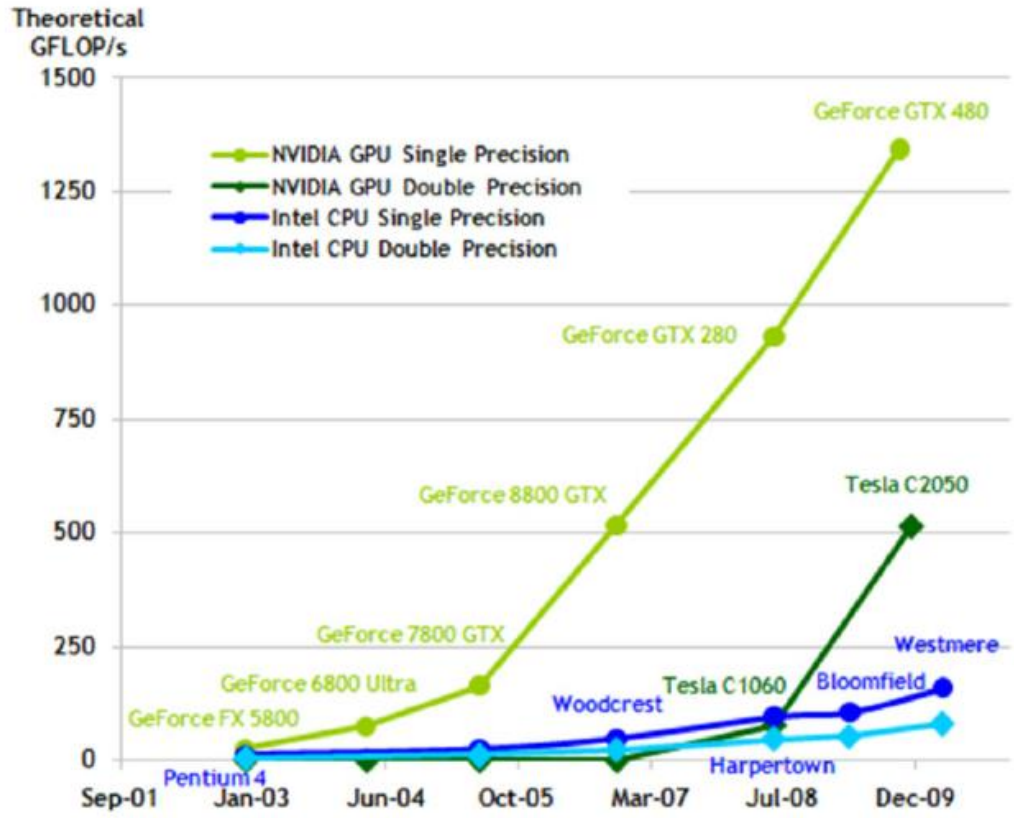
CUDA has been widely deployed through thousands of applications and published research papers, and supported by an installed base of hundreds of millions of CUDA-enabled GPUs in notebooks, workstations, compute clusters and supercomputers. Applications used in astronomy, biology, chemistry, physics, data mining, manufacturing, finance, and other computationally intense fields are increasing using CUDA to deliver the benefits of GPU acceleration. Using CUDA, for data parallel applications, accelerations of more than two orders of magnitude have been seen. CUDA supports Windows, Linux and Mac OS.

The three key abstractions - a hierarchy of thread groups, shared memories, and barrier synchronization - preserves language expressivity by allowing threads to cooperate when solving each sub-problem, and at the same time enables automatic scalability. Indeed, each block of threads can be scheduled on any of the available multiprocessors within a GPU, in any order, concurrently or sequentially, so that a compiled CUDA program can execute on any number of multiprocessors.



Why GPU?

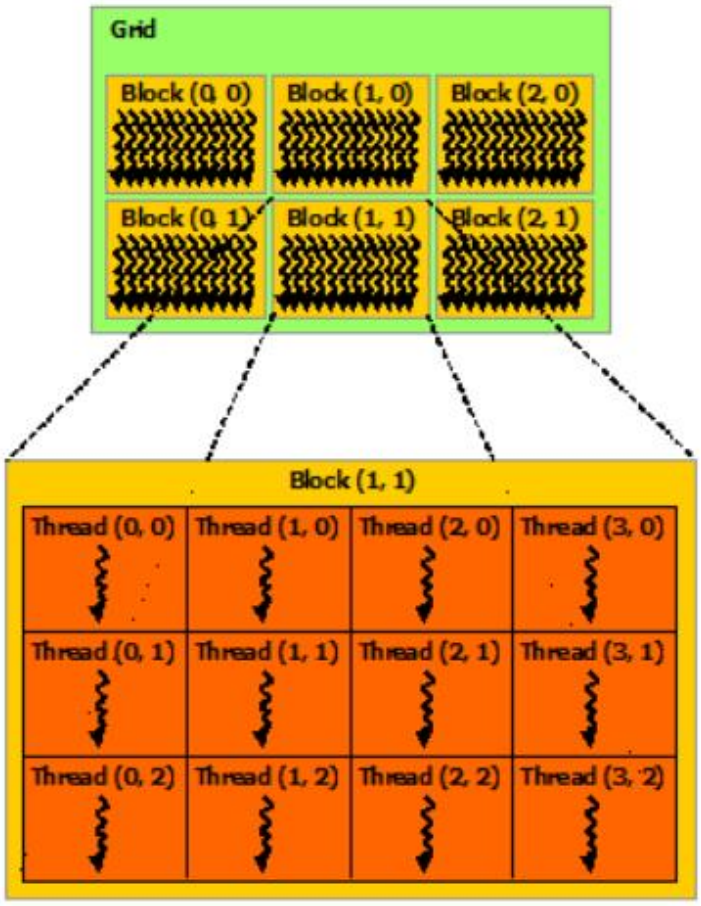
market demand for real time, high-definition 3D graphics, the programmable Graphic Processor Unit or GPU has evolved into a highly parallel, multithreaded, manycore processor with tremendous computational horsepower and very high memory bandwidth.



Working of blocks and threads-

There is a limit to the number of threads per block, since all threads of a block are expected to reside on the same processor core and must share the limited memory resources of that core. On current GPUs, a thread block may contain up to 1024 threads. Total number of threads is equal to the number of threads per block times the number of blocks.

Blocks are organized into a one-dimensional, two-dimensional, or three-dimensional *grid* of thread blocks as shown in figure below. The number of thread blocks in a grid is usually dictated by the size of the data being processed or the number of processors in the system, which it can greatly exceed.


Grid
Block (0, 0)
Block (1, 0)
Block (2, 0)
Block (0, 1)
Block (1, 1)
Block (2, 1)
Block (1, 1)
Thread (0, 0)
Thread (1, 0)
Thread (2, 0)
Thread (3, 0)
Thread (0, 1)
Thread (1, 1)
Thread (2, 1)
Thread (3, 1)
Thread (0, 2)
Thread (1, 2)
Thread (2, 2)
Thread (3, 2)

Algorithm –

- Declare and Initialize matrix on host with random numbers with 0 or 1.
- Declare matrix to be sent to device
- Allocate the memory for device matrix using cudaMalloc
- Copy allocated memory to device using cudaMemcpy() function
- Call the kernel with the information of number of blocks and number of threads per block
    - Calculate the Id of row and column of matrix using thread Id, block Id and block dimension.
    - According to row and column number, determine the index of element of matrix on which thread will compute.
    - Check the neighbors of that element and perform following steps-
        - Any live cell with fewer than two live neighbors dies, as if caused by under-population.
        - Any live cell with two or three live neighbors lives on to the next generation
        - Any live cell with more than three live neighbors dies, as if by overcrowding.
        - Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.
- Copy result back to host using cudaMemcpy() function.
- Print the result matrix.

Explanation of algorithm-

While solving this problem parallelly using CUDA. While calling kernel function, number of blocks and number of threads per block are passed. For example, if there are 100 elements in matrix, it is feasible to divide the matrix into 4 blocks of 25 elements and use 25 threads per blocks (25*4 = 100) so that every thread can work independently on single element. The same logic is used in this problem of Conway's game of life. After passing matrix of elements (0 or 1) to kernel, each thread performs the steps (calculation of element index, checking of neighbors' state and changing the current elements state) mentioned in above algorithm on single element concurrently so that total time can be reduced.

Each thread works on element assigned to it parallelly and store the new state of corresponding element to matrix at the same position. At the end, this updated matrix is sent back to host. Here, each thread works parallelly and independent of each other. Hence correctness of code is guaranteed.

In the device function, **__syncthreads()** function is used to synchronize all the threads within the block so that every thread finish its work parallelly.