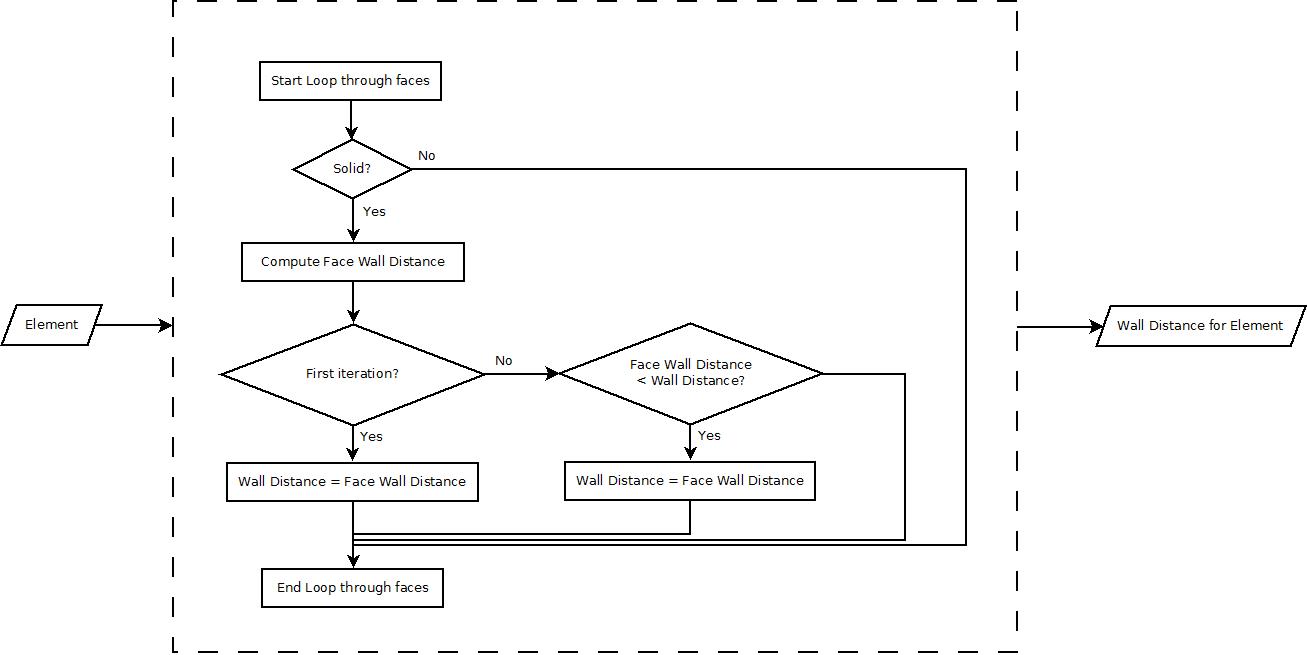
**Brute-force Serial implementation block diagram:**



A serial brute-force implementation for calculating the wall distance for an element is shown in Figure x. For each element, a large number of faces would have to be evaluated to check if the face is solid, to calculate its distance from the wall, and then to update the wall distance if this face has the least distance. The opportunity for parallelism arises in being able to perform the checks and wall distance calculation for each element separately.

For each element:

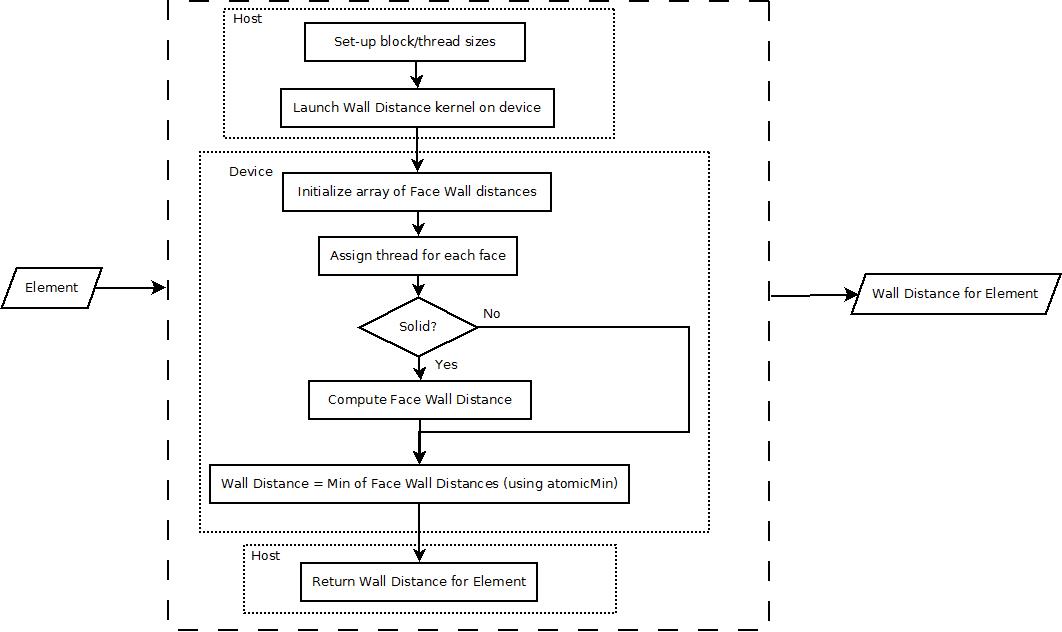
1. Wall distance from a solid face will be calculated
2. The minimum wall distance will be updated if the face wall distance is lower than the current minimum wall distance

The work and step complexities for the serial approach with n elements and m solid faces can be approximated as:

W ~ n\*m = O(mn) – for each element, each solid face will have to be evaluated. So, total number of operations is O(mn).

S ~ m = O(m) – all the elements can be evaluated in parallel, but the distance from each solid face to the element is evaluated in sequence. So, the longest chain of sequential dependencies is O(m).

**Brute-force Parallel implementation#1 block diagram:**

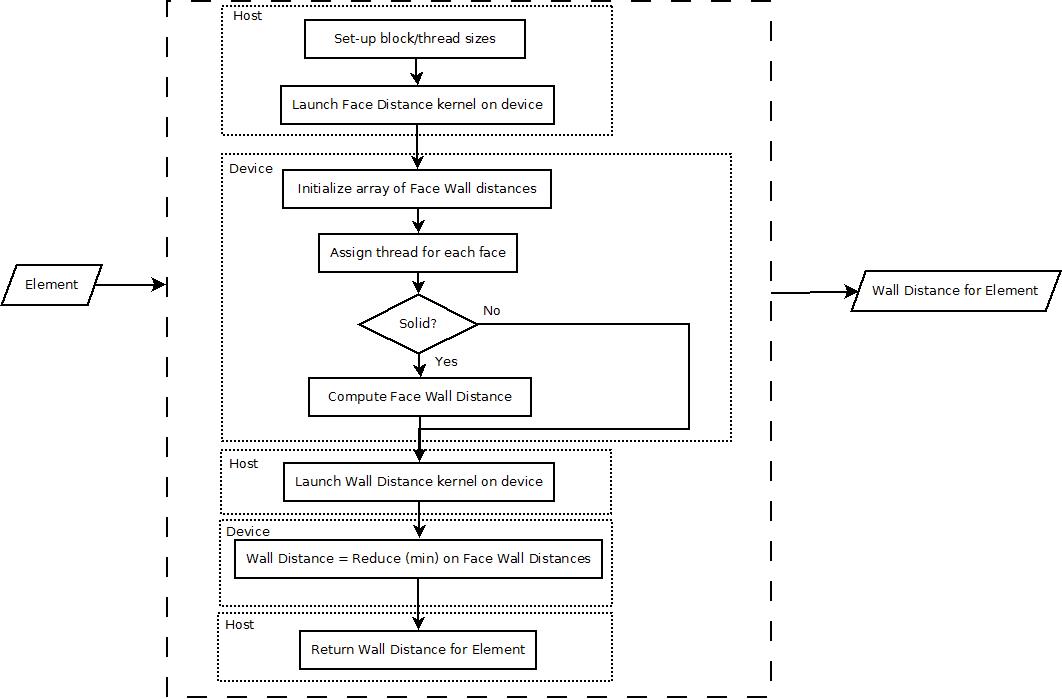


2 simple parallel implementations for the brute-force method will be explored. The first parallel implementation for calculating the wall distance for an element is shown in Figure x. For each element, the host sets up block and thread sizes, and then launches a device kernel. The device kernel will assign a thread for each face, and then compute the wall distance for that face. The minimum wall distance operation will be computed using atomic min so that only one thread is reading and updating the overall wall distance at a given time.

The work complexity for this parallel implementation is slightly higher than the serial as there is additional overhead for assigning threads and initializing face wall distance in the parallel implementation. Overall, the work complexity is similar to the serial implementation: W ~ n\*m = O(mn).

Assuming infinite threads, the wall distances for each solid face can be calculated in parallel in one step. However, the atomic min operation will limit the amount of parallelism that can be performed, as each thread may have to wait a long time before updating the minimum wall distance. Though the step complexity for this parallel implementation can be improved by dividing the atomic operations to be within blocks, and then combining the results from the blocks, it will not be significantly better than the serial implementation: S ~ m = O(m).

**Brute-force Parallel implementation#2 block diagram:**



The second parallel implementation for calculating the wall distance for an element is shown in Figure x. For each element, the host sets up block and thread sizes, and then launches device kernels. The first kernel will assign a thread for each face, and then compute the wall distance for that face (same as the first parallel implementation). However, after this kernel is executed, a reduce (minimum) operation will be performed to calculate the overall wall distance for the element, instead of using an atomic operation.

The work complexity for this parallel implementation is similar to the first parallel (and serial) implementation: W ~ n\*m = O(mn).

However, the step complexity is much lower. Assuming infinite threads, the wall distances for each face can be calculated in parallel in one step, and the reduce operation to compute the minimum wall distance has a step complexity of log m. Therefore, step complexity for this parallel implementation is: S(n) ~ log m = O(log m).

For example, if we have 65,536 solid faces, the second parallel implementation can compute the wall distance in 16 steps, compared to 65,536 steps in the serial implementation.