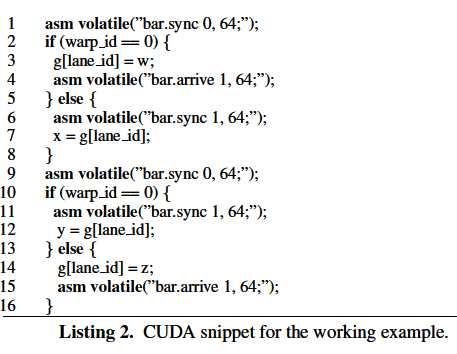
**Summary:**

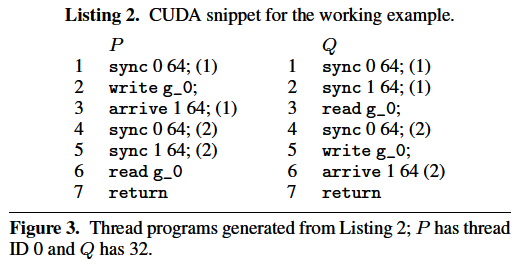
**4. Algorithm**

**4.2 Property checking.**

To determine if the given the CUDA program for the CTA T is well synchronized or not the authors finally present the algorithm in this section. Following is the CUDA program which the algorithm tests, which includes the code for two warps with thread ids 0 and 32.



They also present the thread programs for the two warps and name them P and Q. Following is the Thread program code.

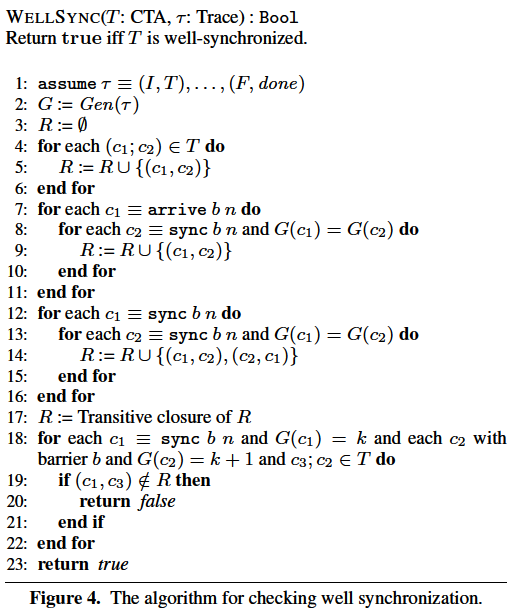


They clearly explain that each memory location g[j] is treated as the individual variable in the shared memory. They statically generate a program trace τ out of all the programs for the sake of representation. The execution of this particular trace can end either in deadlock or in it can be a well synchronized program and for all the threads can end in done and return.

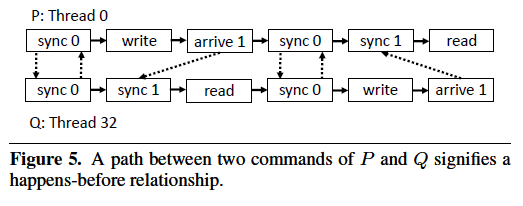
To begin with each of the threads in all the warps must perform sync on barrier 0 so does the warp P followed by which P writes the shared variable, registers at barrier 1 and performs a sync on barrier 0 recycling the barrier. From here the warp Q starts its execution by performing sync on the barrier 0 followed by which it syncs on the barrier 1 then reads the shared variable and syncs on the barrier 0 hence recycling it. After that Q writes the shared variable registers on the barrier 1(again recycling the barrier) and then returns. From here P continues performs sync on barrier 1 followed by reading the shared variable and then returning.

They use the trace to assign the generation ids to the barriers that are given in the parenthesis. For example, all threads have to sync at barrier 0 in the beginning of the program so the generation id for those commands in each warp in our thread programs is 0. We can also see that the sync is performed on the same barrier at fourth step of the warps, at this command the generation ids are set to 2 for both the commands indicating the that the barriers have been recycled once.

To ensure the well synchronization property they statically check that traces assign same generations to the commands Gen(τ). They use the generation for this trace to get the happens before relation. The algorithm is as follows.



For the given trace τ, the algorithm begins with the empty happens before relation R. The relation R is populated with all the possible pairs of commands c1 and c2 that belong to same warp in the given (either P or Q) guaranteed to run sequentially. Next the edges corresponding to the inter thread happens before relation are added. If c1 is a arrive operation on the barrier b and c2 is a sync operation on barrier b and they are in same generation then add (c1, c2) to the relation (for example add the command 3 in P and command 2 of Q must be added). If c1 and c2 are sync operations on barrier b in the same generation then add the pairs (c1, c2) and (c2, c1) to the relation R(For example command 1 of P and command 1 of Q and as a second pair command 1 of Q and command 1 of P). These steps can be observed from the happens before graph also as show in the following image.



In the graph the dashed edges represent the inter warp dependencies and the solid edges represent the intra warp dependencies. The cycles in this graph represent that the commands involved execute simultaneously.

Now, transitive closure of the relation R is computed and a check is made if for all the successive generations of the same barrier there exists a happens before relationship or not. This checking is done by considering the command c1 which is sync on barrier b with generation k any command c2 with generation k+1 on barrier b and command c3 if there does not exist the pair (c1, c3) then the well synchronization property is not satisfied and the algorithm returns false else true.