IOS SUMMARY PART 3:

SEMANTICS:

3. OPERATIONAL SEMANTICS:

Domain focuses on verification of: Deadlock-freedom, proper named barrier recycling and shared memory data race freedom. Inter-CTA and software level Intra-CTA synchronizations are not handled. The latter can be constructed with atomic primitives but verification of this method is completely unrelated to solving the problem of verification of named barriers.

<https://lwn.net/Articles/695257/> This article explains clearly what atomic primitives are and how the synchronization is achieved.

It is assumed that all CTAs execute the same program and verification of one CTA is enough to establish correctness.

Only shared memory of the GPU is validated for race conditions.

Syntax:

N: Number of threads within a CTA (32-1024)

P : A thread in a CTA.

T: A CTA

P1 || P2|| … PN to denote a CTA with P1,P2,…PN threads.

g : Variables in shared memory location. (64 bit variables)

id : separate identifier of each thread. i is used to range over various threads.

B : Barriers; b: A barrier

Every thread program is considered to be an abstract thread program.

Abstract thread program: Threads are assumed to be programs with instructions requiring synchronization and shared memory accesses.

Any thread Program has the following grammar:

P ::= return|c ;P

A thread program is a sequence of commands which are straight line code. Does not involve non-deterministic loops and conditionals or branches.

Each command is a read/write/synchronization instruction. Read and write commands are treated as no-ops ( In [computer science](https://en.wikipedia.org/wiki/Computer_science), a **NOP**, **no-op**, or **NOOP** (pronounced "no op"; short for **no operation**) is an [assembly language](https://en.wikipedia.org/wiki/Assembly_language) instruction, [programming language](https://en.wikipedia.org/wiki/Programming_language) statement, or [computer protocol](https://en.wikipedia.org/wiki/Protocol_(computing)) command that does nothing. (Wiki)) used to detect data races and do not play a role in explaining semantics of named-barriers.

c ::= read g | write g | arrive b n | sync b n

sync and arrive operations are synchronization operations. b represents the barrier on which this synchronization is performed, n refers to the number of threads which needs to be registered at one generation of a barrier.

Standard barrier or the *syncthread* which is used to achieve block level synchronization i.e. to say that a barrier is safe to use only when all thread in the block reach the barrier. This is example on barrier *0: sync 0 N*

Program points: The point of execution of code: The program point is defined by the command just before it.

Warp-synchronous execution: This is the type of execution where all the threads in the warp execute the same instruction i.e. in a lock-step. This can be achieved by adding a sync command across all the threads after every original command in a program.

3.2: State

Enabled map E: A map which explains if a thread is enabled or is arrived, or blocked or disabled from execution.

A barrier map B: A triplet which consists of two lists and a count. List 1 I: list of blocked threads.

List 2 A: List of arrived threads and the count of number of threads which can use the barrier on configuration.

The synchronization instructions must specify the number of participants as this needs to be configured by the first thread reaching the barrier.

Initially all threads are enabled, ready to execute, no thread has registered at any barrier, and all barriers are unconfigured.

Notations:

[] - Empty list of threads

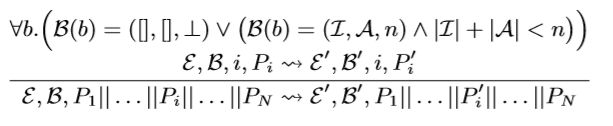
:: - adding a thread to list

| : denotes an unconfigured barrier

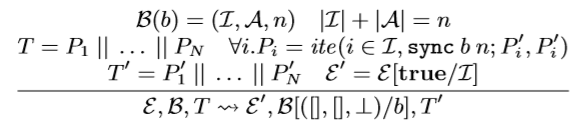
3.3 Semantics:



A CTA / Thread program in one state go to another state after one step.



The above notation suggests that, for all the barriers b either each barrier is unconfigured or needs to register more threads. In such a case any thread is non-deterministically chosen and executed for one step, thus changing its state from Pi to Pi’

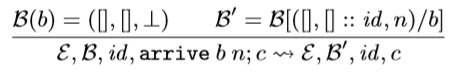


Another state can be when a barrier has encountered all the threads that needs to be encountered and hence needs to unblock all the blocked threads to recycle the barrier, enable the control of blocked threads.

The execution terminates when all threads execute a return.

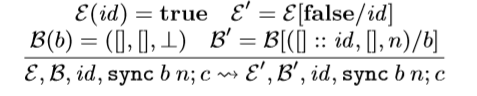


The barriers are reset before starting a new CTA.

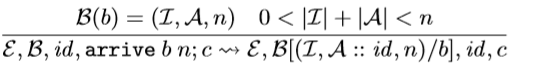


When a new thread arrives at a barrier then it is added to the list of Arrived thread, and also configures the barrier with thread count.

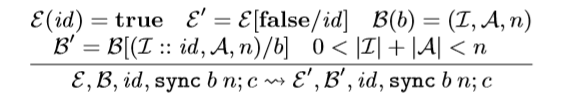
If the first thread is sync then the control and the barrier map are updated. The thread updates E map and is added to the list of blocked thread.



On executing a non-blocking arrive at the barrier the control and the barrier map are updated.



When a sync is encountered, the thread is added to the list of blocked threads and control remains the same.



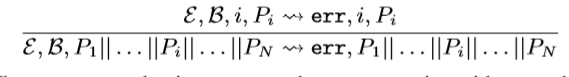
If too many threads are encountered or if more than the registered number of threads encounter a barrier the state is changed to err state.



or



If any thread produces error then the entire CTA terminates in an error state.



These error productions ensure that an execution either reaches done, goes to err, or deadlocks. No other outcome is possible.