Association will consider defining a small set of unified API calls and header files that enforce these semantics.

3.4 MRAPI Synchronization Primitives

The MRAPI synchronization primitives include *mutexes*, *semaphores*, and *reader/writer locks*.

Mutexes are intended to be simple binary semaphores for exclusive locks. Semaphores allow for counting locks. The reader/writer locks can be used to implement shared (reader) and exclusive (writer) locking. Mutexes are intended to support very fast, close to the hardware implementations, while semaphores and reader/writer locks provide more flexibility to the application programmer at the expense of some performance.

MRAPI provides blocking and non-blocking functions for obtaining locks on the synchronization primitives. The non-blocking functions should be used in conjunction with the `mrapi_test()`, `mrapi_wait()` and `mrapi_wait_any()` functions to determine when the request has completed. There is no need for non-blocking semantics for unlock because it always happens immediately.

All of the synchronization primitives are supported across MRAPI domains by default but this may have a performance impact (e.g., chip-to-chip will necessarily be slower). Sharing across domains can be disabled by setting the `MRAPI_DOMAIN_SHARED` attribute of a synchronization primitive to `MRAPI_FALSE` (default is `MRAPI_TRUE`).

3.4.1 MRAPI Mutexes

The basic semantics of MRAPI mutexes can be summarized as follows: MRAPI mutexes are binary, they support recursion (but that is not the default), and they are intended to be the closest match to underlying HW acceleration in many systems. Recursive locking is allowed if the locking node already owns the lock, and if the mutex attributes have been set up to allow recursion. Recursive locking means that once a mutex is locked, it can be locked again by the lock owner before unlock is called. For each lock, a unique lock key is returned. This lock key must be provided when the mutex is unlocked. The implementation uses the keys to match the order of the lock/unlock calls. Recursive locking is disabled by default. Note that individual mutex attributes may vary, but they must be set before mutex creation and cannot be altered later.

3.4.2 MRAPI Semaphores

Semaphores are differentiated from mutexes in that they support counting locks. Therefore semaphores are differentiable in terms of performance (mutexes are binary and some hardware has HW acceleration for this, and semaphores are more rich functionally but may have slower performance) as well as features.

3.4.3 MRAPI Reader/Writer locks

The MRAPI reader/writer locks provide a convenient mechanism for optimized access to critical sections that are not always intended to modify shared data. These primitives support multiple read-only accessors at any given time, or one exclusive accessor. This supports the RWL (Reader/Writer locks) software pattern that is commonly used for cases where there are more readers than writers. In order to guarantee fairness, MRAPI implementations must enforce serialization of requests such that that no new read lock will be granted while a blocking write lock request is pending.

3.4.4 MRAPI Atomic Instructions (ABB Extension)

The MRAPI atomic instructions access hardware-specific CPU operations in a portable fashion (currently Windows and Linux), extending the capability through shared memory across multiple address spaces. These primitives support read and write access to operating system dependent atomic data types. An optional argument to each call passes a barrier structure that must be initialized for nodes in non-Windows deployments that communicate across processes. There is no additional computational overhead where the barrier is not needed.

3.5 MRAPI Memory

MRAPI supports two different notions of memory: *shared memory* and *remote memory*. Shared memory is provided in MRAPI to support applications that are deployed on hardware with physically shared memory with hardware managed cache coherency (*coherent shared memory*), but which cannot rely on a single operating system to provide a coherent shared memory allocation facility. It is