Notes on sip\_barrier implementation

BAS 2/28/14

Barriers from the point of view of a SIAL program essentially ensure that

1. All modifications to arrays before the barrier are visible to all workers after the barrier.
2. All workers have reached the barrier before any of them continue.

From the point of view of the SIP, a barrier is a termination detection algorithm. In general, termination detection algorithms detect that 1. all processes have terminated, there are no messages in transit, and that the global state used is consistent.

For the SIP, a barrier can be passed after detecting

1. all workers have reached the barrier
2. all servers have handled all pending requests
3. there are no messages (except for control messages implementing the barrier itself) in transit
4. the view of the combined global state is consistent

First characterize behavior of the workers and servers w.r.t. termination and message passing.

Messages:

1. get: worker sends msg to server and posts ireceive, server replies with block.
2. put: worker sends two messages to server, posts ireceive for acknowledgement. The server replies with acknowledgement.
3. set\_persistent and restore\_persistent: a worker sends a message to a server, posts ireceive for acknowledgement s. The server replies with an acknowledgement. Note that every server who holds any part of the indicated array must be sent a message—the easiest way is to let the master do it.

Process state to detect

1. Worker: at barrier, no unsatisfied gets, or other unacknowledged messages
2. Server: no pending messages. A pending message implies an unsatisfied get or some unacknowledged message, thus no unsatisfied gets or unacknowledged messages in the system imply no pending messages.
3. Network: no messages in transit. A message in transit implies an unsatisfied get or some unacknowledged message, thus no unsatisfied gets or unacknowledged messages in the system imply no messages in transit.

At a barrier:

Workers:

1. Check for unsatisfied gets and waits for them to be satisfied if necessary (Note that if there are unsatisfied gets at a barrier, the get is requesting a block that wasn’t needed, so the program should be examined.)
2. Wait for all remaining messages to be acknowledged.
3. Perform an MPI\_barrier on the communicator formed from all the workers.

Server: No action required.

Lemma: The global state is consistent.

Proof: An inconsistent state requires that a message sent after the barrier is received before the barrier. MPI barriers require that all workers reach a barrier before any worker can send a message after the barrier.

Dynamic check for correctness:

Workers and servers maintain a barrier\_counter whose value is included in each message.

Workers: increment barrier\_counter each time a sip\_barrier is passed. If a message is received with a different from the worker’s own, this is an error.

Servers: if a message is received with a value smaller than the server’s own barrier\_count, this is an error. If a message is received with a value larger than the server’s own, replace the server’s barrier\_count with the one included in the message.

Smarter Servers

A different approach would be to let servers run a simple interpreter that proceeded through the code determines which gets, puts, set\_persistent, and set\_restore should be received before the next barrier. When a barrier is reached, wait for and handle any expected message that has not yet arrived. Participate in MPI\_barrier that includes all workers and servers.