Charm++ - Bug #2045
Chare array broadcast messages don't appear to be freed by the runtime
01/21/2019 02:16 PM - Sam White

Status: In Progress
Priority: Normal
Assignee: Juan Galvez
Category: 
Target version: 

Description
Need to debug further, but it seems like doing many broadcasts we run out of memory eventually due to the messages not being freed.

History
#1 - 01/21/2019 02:31 PM - Sam White
This should only affect parameter marshalled entry methods and [nokeep] broadcasts, where the user doesn't take ownership of the message object.

This patch made CmiReference atomic somewhat recently, and could possibly be affecting the broadcast code:
https://charm.cs.illinois.edu/gerrit/#/c/charm/+/4108/

#2 - 01/21/2019 03:46 PM - Juan Galvez
I haven't looked at the array broadcast internals to know what the issue is, but it's possible that this works this way to support anytime migrations. If PEs need to be able to store messages for a time because some chare might arrive that has not received the broadcast message, then the solution to the problem might not be obvious. Obviously you would have to purge the messages after a point, but when? If this is the case, this is likely to affect any type of messages, not just marshalled. If the user is supposed to take ownership of the message, the array manager needs to keep the original copy of the message to be able to give it to other array elements.

#3 - 01/21/2019 04:05 PM - Sam White
That's true. AMPI (and possibly other applications) sets `_isAnytimeMigration = false;` and `_isStaticInsertion = true;` inside an [initnode] routine, which is supposed to be used by the runtime to detect if the array has "stable locations". If you grep for "stableLocations" in src/ck-core/ you'll see some code that is supposed to conditionally either do "spring cleaning" of broadcast messages every so often (currently 1 min, which seems arbitrary) or delete broadcast messages after finishing delivering them on each PE.

My guess is that your benchmark does not set the above values or use CkArrayOptions to specify that it has "stable locations", so you are getting the spring cleaning behavior and your benchmark takes less than a minute to run so it never gets there? If that's the case, we can decrease the spring cleaning time interval some, at least. Or maybe there's really something broken with spring cleaning and/or the stable locations code path...

#4 - 01/21/2019 08:44 PM - Juan Galvez
I'm just running with default values, so spring cleaning option seems to be the default. I didn't know about the other options.

I ran another test, where the broadcast volume is lower, and mem usage stops growing at approximately 120 seconds into the program. It doesn't free all the messages (just the oldest ones) so mem usage doesn't decrease at all at once, it just basically stops growing. Weirdly if 1 minute is the default I don't know why it stops growing at 2 minutes.

Anyway, I don't like timers since they are completely arbitrary and one value for the timer might work well for some application but not another. Still, there might not be a better solution. If timers is the way to go, it might be possible to use a pretty small value as default (a few seconds?), and have the runtime abort with a message if a migrating chare misses a message. Message could say something like: "increase timer with this command line..."
option”. I bet most applications wouldn’t even encounter this message.

I don’t have any real use case where spring cleaning behavior is problematic (i.e. where the rate of broadcasts is so high as to consume lots of memory). I encountered the issue because of a bug in my code. But since spring cleaning is the default, I think it’s best to avoid the unnecessary memory consumption for long periods of time since we shouldn’t assume anything about other users’ code or memory constraints.

#5 - 01/21/2019 08:58 PM - Sam White
Yeah, I agree with everything you said. Timer defaults are always hacky, but 1 or 5 seconds seems better than 1 minute to me (plus adding an abort message). Can you also confirm that if you set the stableLocations option to true, the memory of broadcast messages is always freed?

We can also discuss changing the default to be no anytime migration in Charm++ v7.0. (Plus you could make it the default in CharmPy earlier if you wanted)

#6 - 01/21/2019 08:58 PM - Sam White
- Status changed from New to In Progress

#7 - 01/21/2019 09:02 PM - Juan Galvez
One case that occurs to me that could be problematic is a typical implementation of Stochastic Gradient Descent commonly used in Deep Learning. The parallel implementation is basically a loop where each worker does some computation, and then AllReduce happens so that workers communicate the averaged model parameters to other workers. And this happen for X number of iterations until convergence is reached. The problem is that model sizes can be several hundred MB, and if the allreduce is implemented as reduction followed by broadcast, that means each iteration there is a broadcast of several hundred MBs.

#8 - 01/21/2019 09:09 PM - Sam White
Yes, and in that scenario you really want the zero copy API for broadcasts and reductions in order to avoid intermediate copies of the data as much as possible.

#9 - 01/22/2019 10:12 AM - Jim Phillips
Perhaps spring cleaning could be triggered by the number of accumulated broadcasts (and/or amount of memory used by accumulated broadcasts) to a single array rather than by elapsed time. Or, rather than broadcasting to all nodes, you build a node-adapted tree of array elements and periodically update the tree to correspond to the node topology.

#10 - 01/24/2019 02:21 PM - Sam White
I agree with Jim’s suggestion for moving away from a timer-based spring cleaning mechanism. I don’t think the documentation currently mentions that you can set ‘_isAnytimeMigration = false;’ and ‘_isStaticInsertion = true;’ inside an [initnode] routine or by setting CkArrayOptions for those options.

In Charm v7.0 we are proposing to make no anytime migration and static insertion the defaults, with opt-in-ability for anytime migration and dynamic insertion.