Send buffer access considered harmful

Alexander Supalov

Intel Corporation

Contents

Executive summary

- **Vote #45 and #46 down**
  - #45 and #46 are based on questionable motivation
  - #45 and #46 disallow several promising optimizations
  - #45 and #46 disadvantage little-endian platforms
  - #46 is syntactically questionable
  - #46 makes the damage irreversible

Summary of objections

- #45 send access proposal restricts freedom of implementation. This statement is based upon historic precedent and several realistic, forward looking scenarios.
- #46 const buffer proposal depends on #45 and makes it irreversible thru a syntactic change.
  - Since #45 is questionable, #46 is questionable by inference. Moreover, #46 has a syntactic weaknesses of its own.
  - Inversely, if #46 is the desired end of the inference chain, weakness of #46 weakens motivation for introducing #45.
- The argument in favor of #45 and #46 is the desire to make a few existing and future incorrect MPI programs valid. These programs are yet to be identified in any substantial number.
  - Whether or not the unrelated MPI-3 MPI_INIT_ASSERTED proposal is going to help to revert #45 later remains to be seen. Accepting #46 now will definitely eliminate that option.

Objection (1/6) Questionable motivation

- #45 and #46 are motivated by “user friendliness” over implementation freedom. This means:
  - Fixing a couple of unidentified existing programs. What are they? How many are out there? Please be specific.
  - If user friendliness is the king, should we also try to make the following sequence work? It is much more popular than send buffer access:

    MPI_Send(self, ...)

    MPI_Recv(self, ...)
  - If a program really needs several MPI_Sends over the same buffer, maybe it should simply use MPI_Bcast?

##45, 46 mean special disadvantage for the ubiquitous little-endian HPC platforms

Objection (2/6) In-place send buffer transformations

- #45 disallows many reversible in-place send buffer conversions. Examples identified so far include:
  - Byte swapping. ***This was actually done in the past***.
  - Little-endian to big-endian conversion on the sender
    - Converting native format to network order
    - Implementation of the Interoperable MPI standard (see http://www.nist.gov/impi)
    - Implementation of the external32 data representation for file I/O
  - In-place compression and/or encryption done by the CPU

##45, 46 mean special disadvantage for the ubiquitous little-endian HPC platforms

Objection (3/6)
Send buffer remapping to receiver

- #45 disallows the following shared memory optimization
  - In the MPI_Isend, MPI maps the send buffer into the address space of the receiving process.
  - In the matching MPI_Recv, the receiving process makes a copy of the mapped send buffer into the receive buffer.
  - Once the copy is complete, the send buffer is mapped back into the sender address space.
  - Remapping can be applied to page-aligned buffers or page-aligned parts of the buffer
  - Note: retaining the remapped send buffer in the sender space will infringe on IBM's patent #**7,392,256**

##45, 46 effectively exclude memory remapping from the implementor's arsenal

Objection (4/6) Fortran language implications

- #45 makes multiple copy-in/copy-out necessary in Fortran
  - More than one send with a nontrivial and large buffer section will potentially generate an additional copy-in buffer.
  - This applies mostly to blocking send in multithreaded environment, as non-blocking sends have a known problem of their own with the copy-in/copy-out.

##45, 46 increase memory usage in Fortran

Objection (5/6)
C language implications

- #46 introduces ambiguity into the syntactic meaning of the send buffer
  - A const modifier on the MPI_BOTTOM and/or a derived data type, possibly with holes in it, does not seem to square well with C language sequence association rules.

#46 has questionable syntactic meaning in C

Objection (6/6) MPI_INIT_ASSERTED won't help

- #46 is irreversible
  - The const modifier from #46 makes #45 irreversible.
  - Even the MPI_INIT_ASSERTED will be useless, should we later decide to introduce an assertion like MPI_NO_SEND_BUFFER_READ_ACCESS.

#46 makes the damage irreversible

Conclusion

- **Vote #45 and #46 down**
  - #45 and #46 are based on questionable motivation
  - #45 and #46 disallow several promising optimizations
  - #45 and #46 disadvantage little-endian platforms
  - #46 is syntactically questionable
  - #46 makes the damage irreversible
- All this for the sake of a few broken programs?
- **This is akin to putting someone on the death row without material evidence.**
- **Are we beyond reasonable doubt here?**