

AD
2008

Adjoinable MPI idioms & source transformation



collaborators: Hascoët, Hill, Hovland, Naumann, Utke see also *On Fully Automating Differentiation of Message-Passing HPC Codes* preprint ANL/MCS-P1530-0808

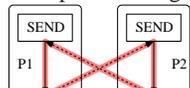
the problem: numerical models use the *message passing*

interface (MPI) for communication in parallel execution and need MPI calls transformed to compute adjoints

- send/rcv modes
- collective operations
- wildcards
- correctness/efficiency of adjoint communication
- barriers
- data flow analysis through MPI calls

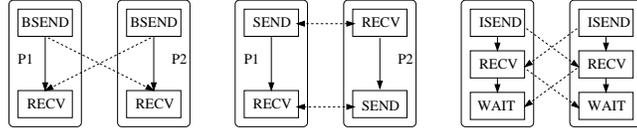
correctness \Rightarrow send / rcv modes:

- correct communications (data, endpoints) ?
- no deadlocks? - modeled with communication graphs
- example: exchange between P1 and P2



... has a cycle (involving comm.edges)

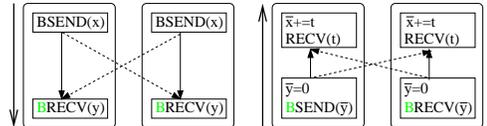
- break with buffered sends, reordering, non-blocking sends, ...



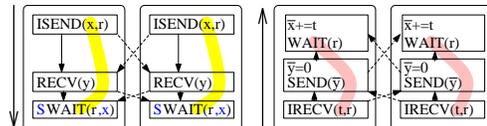
- adjoining plain send/rcv calls:
 - easy adjoint: $send \mapsto rcv$ and $rcv \mapsto send$
 - if forward communication graph acyclic, so is the adjoint; look at the communication graph with reversed edges (Paul)
 - difficult to statically analyze send/rcv pairs; e.g. consider set of all possible dynamic comm. graphs
 - with wildcards: record actual sources/tags on receive and send with recorded tag to recorded source in the adjoint sweep
 - hypothesis: no forward deadlock \equiv no cycle in current dynamic comm. graph \Rightarrow no cycle in inverted dynamic comm. graph \equiv no adjoint deadlock
- send modes: `mpi_[i][b][s|r]send`
- receive modes: `mpi_[i]rcv`
- variants to ensure correctness while reducing wait periods

some communication patterns collaboration:

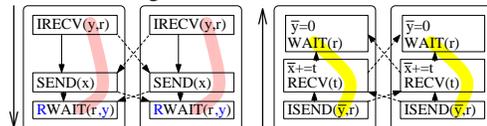
- for buffered/synchronous send - need to mark the rcv



- combination with non-blocking sends (promise to not read or write buffer)



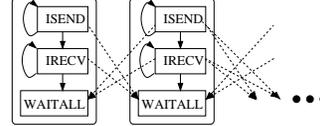
- or non-blocking receives



- buffering takes time and system can run out of space,
- avoid deadlocks via reordering (difficult; order can be dynamic; imposed waits \Rightarrow efficiency loss)
- least order imposed by non-blocking calls with collective `mpi_wait_all`
- specific modes require context information
- can be supplied by distinct interfaces + extra arguments
- alternative: pragma-identified communication channels

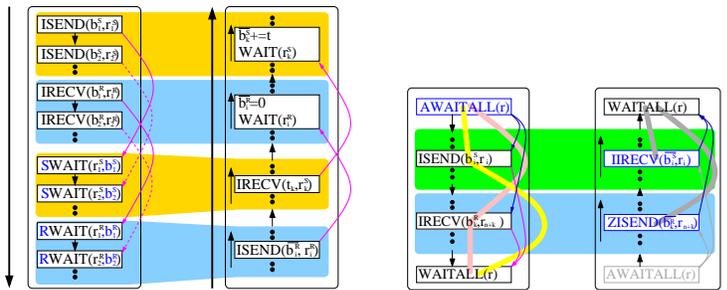
example — exchange of boundary layers:

- all communicate with their resp. neighbors (e.g. East/West)



- problem: the comm. edges are no longer 1 on 1 (multiple in-edges !)

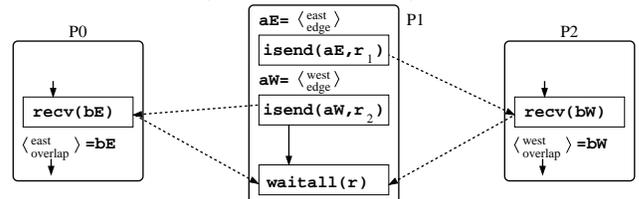
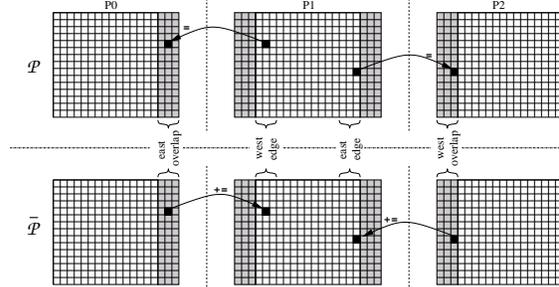
some solution options:



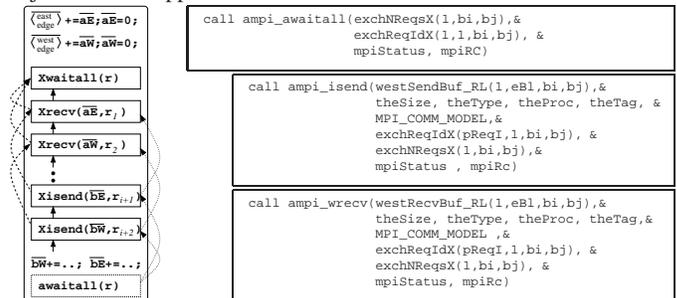
- right: splits collective operation
- left: symmetrize pattern; retains more efficient pattern ☺
- integrate buffer increment & zeroing into MPI wrappers
- require new promises for AWAITALL

prototype implementation in OpenAD:

- uses wrapper routines
- data flow analysis with stubs via global variable
- alternative MPI-CFG based analysis (Strout & Kreaseck)
- MITgcm data exchange example:



- adjoint with wrapper-based communication reversal:



Conclusion: Can adjoint many typical MPI patterns — but need modified symmetric communication patterns via set of special MPI wrappers or a pragma equivalent