Representing Execution Variations in Parallel Communication Protocols

Peter Sobe

Dresden University of Applied Sciences
Faculty of Computer Science and Mathematics

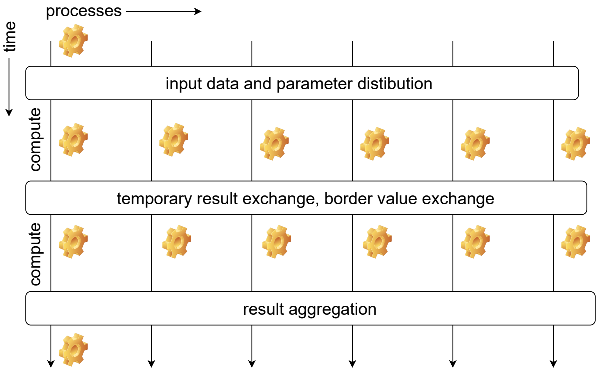
29th PARS Workshop, Aachen, September 2023

Contents

1. Parallel Communication Protocols
2. Expressing Variations of Message Transfer
3. Allreduce Communication Scheme
4. Summary

Parallel Communication Protocols

Communication phases as specific parts of parallel executions

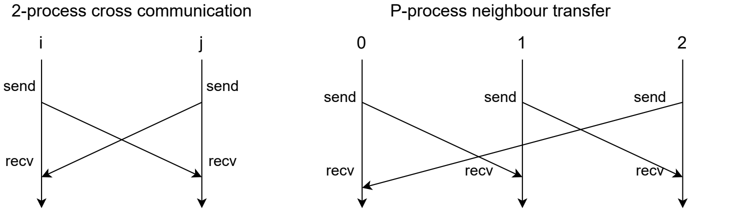


- well-known communication patterns (protocols)
- designed to transfer messages in parallel and to run fast
- adapted, evaluated and compared for different assumptions (network properties, programming interface)

Parallel Communication Protocols

Crossing messages

- exploit the bidirectional nature of communication links
- are good for parallelism
- purely present in a 2-process cross communication and also in a ring-wise neighbour transfer (P>2 processes).



However, it depends on other factors whether messages are transferred in parallel, transferred consecutively or not at all due to a deadlock.

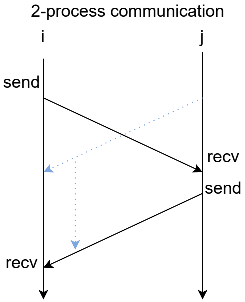
Parallel Communication Protocols

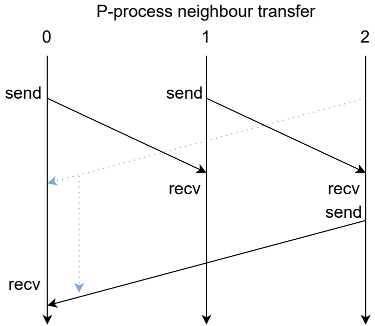
Resolving cross communication

- cross communication cause a deadlock when communication operations block
- cross communication can be resolved by separating message transfer in several rounds.
- rounds are executed in consecutive order, independently of blocking or non-blocking communication.
- the P process neighbour exchange requires one extra round in the protocol description (program) and possibly gets serialized stepwise in further rounds by resolving dependencies at execution time.

Parallel Communication Protocols

Protocols with resolved cross communication

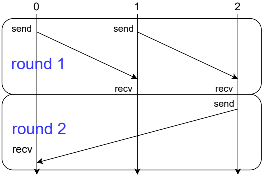




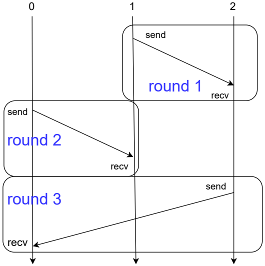
Parallel Communication Protocols

Example: $P = 3$ process neighbour transfer

assuming non-blocking transfer,
the protocol runs in 2 rounds:
round 0: $0 \to 1$, $1 \to 2$ in parallel
round 1: $2 \to 0$



When message transfer blocks,
the protocol runs in 3 rounds:
round 0: $1 \to 2$
round 1: $0 \to 1$
round 2: $2 \to 0$



Expressing Variations of Message Transfer

Protocols with the same general concept can be implemented in different ways

- exploiting or avoiding cross communication
- using synchronous send/recv or asynchronous operations,

. . . and produce different executions at run time

- depending on blocking behaviour of communication network

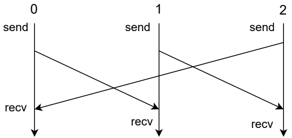
Questions:

- can this be expressed in graphical representations of the protocols, without only showing a selected variant?
- is there any systematic classification of variants and their circumstances?

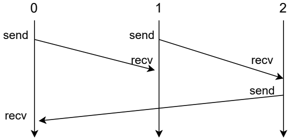
Expressing Variations of Message Transfer

Classification:

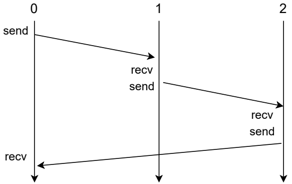
Type 1 (least restictive, deadlock-prone):
Relying on buffering and non-blocking msgs,
even in case of cross communication



Type 2 (less restrictive, deadlock safe):
Resolved cyclic dependencies,
but taking advantage from non-blocking
transfers, rendezvous not required



Type 3 (most restrictive):
Fully predefined message order,
rendezvous forced by implementation
(parallel transfers still possible)



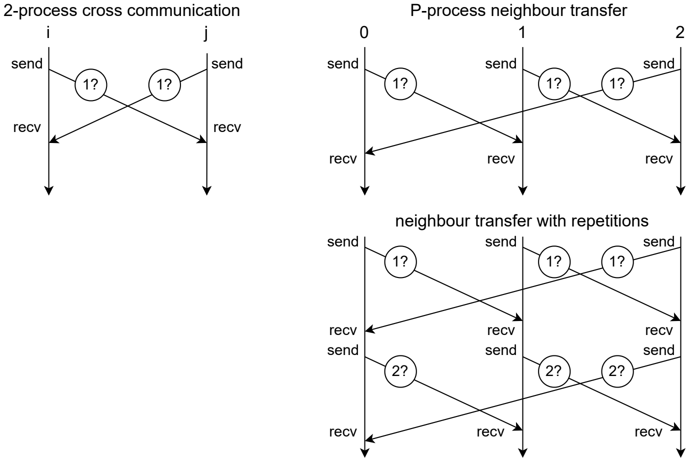
Expressing Variations of Message Transfer

Graphical Representation

- Messages are tagged with round numbers, in case it can be decided
- Otherwise messages are tagged with tupels, representing the earliest round and the latest round in an execution
- Type 1 protocols (deadlock prone) tagged additionally with '?' as postfix of the round number - it means that messages are possibly never transferred

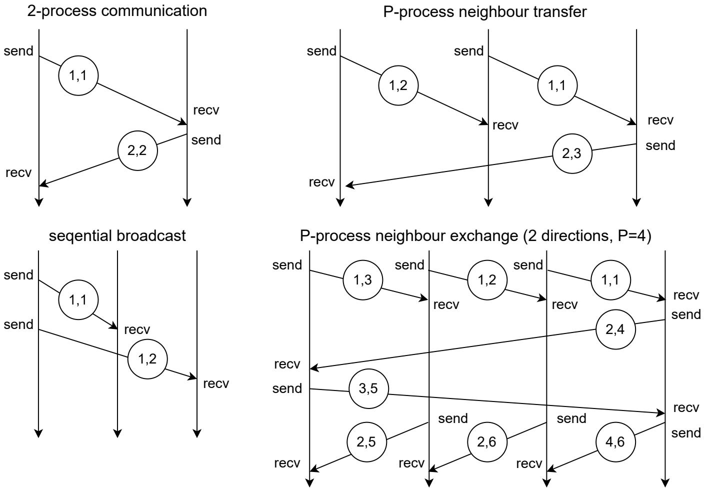
Graphical Representation

Type 1 protocols: tagging with round number and ‘?’



Graphical Representation

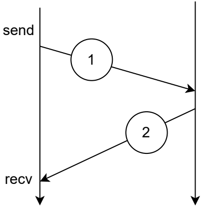
Type 2 protocols: tagging with earliest and latest possible round



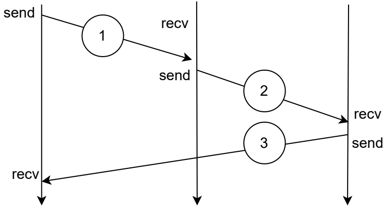
Graphical Representation

Type 3 protocols: tagging with round number

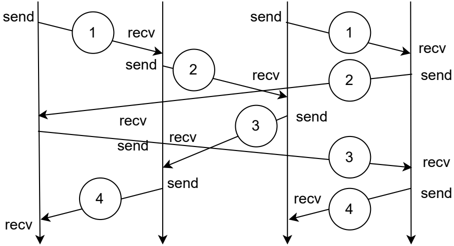
2-process communication



P-process neighbour transfer



P-process neighbour exchange (2 directions, P=4)



Allreduce Communication Scheme

we apply the type schema and the graphical representation to the allreduce communication scheme (MPI_Allreduce)

1st step:

- handling reduce and broadcast separately

2nd step:

- folding reduce and broadcast into another

Questions:

- Does the representation scheme allows a better understanding?
- Is the folded protocol beneficial over a subsequent execution of reduce and broadcast?

Allreduce Communication Scheme

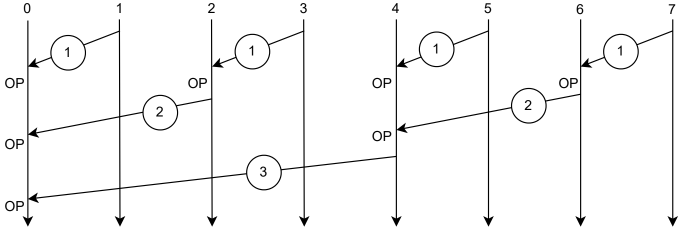
```
epsilon=1.0e-8;
global_delta = 1;  // start value

while (global_delta > epsilon)
{ local_delta = 0.0;
  exchange_overlaps(U, upper, lower);
  for (int row = r.start_row; row <= r.end_row; row++)
  {
    int i = row - start_row;
    for (int col = start_column; col <= end_column; col++)
    {
      int j = col - range.start_column;
      double Uold = U[i][j];
      if (i>0 && i < end_row - start_row)
        U[i][j] = 0.25 * (U[i][j + 1] + U[i][j - 1] + U[i - 1][j] + U[i + 1][j]);
      else
      { if (i==0) // upmost row of local area
          U[i][j] = 0.25 * (U[i][j + 1] + U[i][j - 1] + upper[j] + U[i + 1][j]);
        else // last row
          U[i][j] = 0.25 * (U[i][j + 1] + U[i][j - 1] + U[i - 1][j] + lower[j]);
      }
      local_delta = max (local_delta, fabs (Uold - U[i][j]));
    }
  }
  // ########################################################################
  MPI_Allreduce(&local_delta, &global_delta, 1, MPI_DOUBLE, MPI_MAX, comm );
  // ########################################################################
}
```

Allreduce Communication Scheme

reduce

- reduce gathers values from all processes and calculates a result value that arrives at a distinguished process (root process).
- one common operation (OP), for example a sum.
- type 3 protocol: parallelism present, acyclic data dependencies, acyclic communication patterns

graphical representation of a reduce operation for p=8 processes and the root process 0:

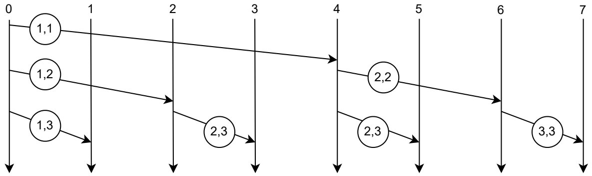


Allreduce Communication Scheme

broadcast

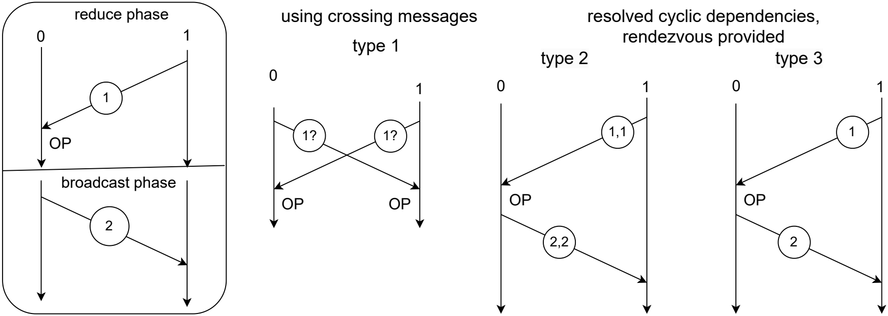
- a broadcast distributes a value (also entire data structures in a serialised form) onto a group of processes.
- parallel broadcast algorithm: recursive doubling
- type 3 protocol: parallelism present, acyclic data dependencies, acyclic communication patterns
- type 2 protocol: possible due to earlier send operations, w/o waiting for rendezvous

recursive doubling for p=8 processes:



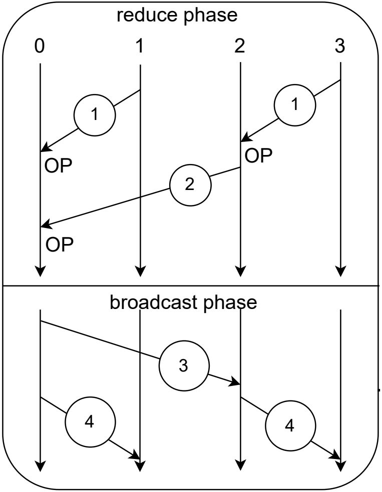
Folding of Reduction and Broadcast

Folding reduce and broadcast for p=2:



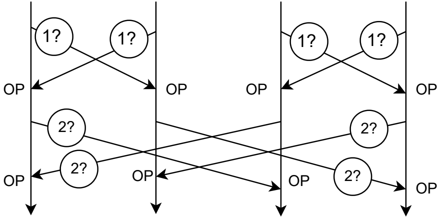
Folding of Reduction and Broadcast

Folding reduce and broadcast for p=4:



using crossing messages

type 1

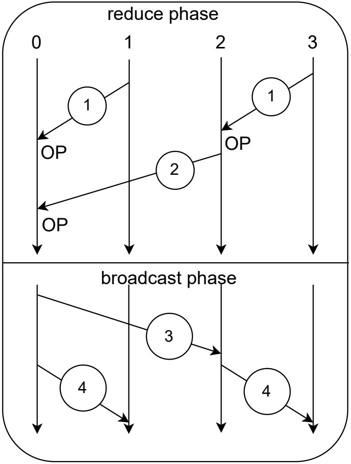


unmodified sequence of reduce and broadcast:

type 3 protocol

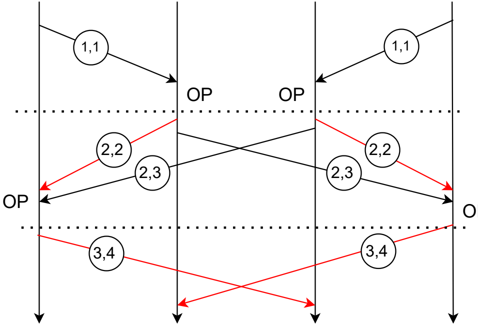
Folding of Reduction and Broadcast

Folding reduce and broadcast for p=4:



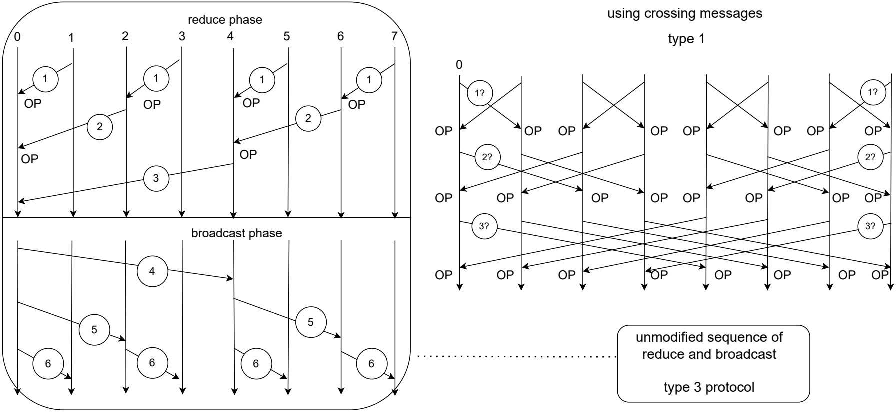
resolving cross communications and overlapping
with next round

type 2



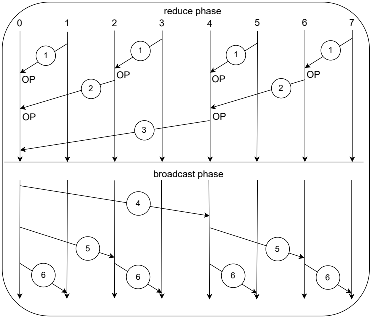
Folding of Reduction and Broadcast

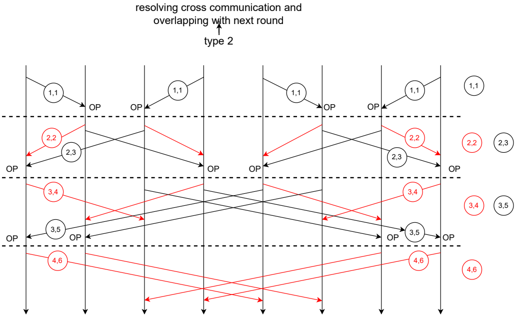
Folding reduce and broadcast for p=8:



Folding of Reduction and Broadcast

Folding reduce and broadcast for p=8:





Folding of Reduction and Broadcast

Comparison with p as number of processes
(p is power of 2)

	reduce	broadcast	folded allreduce type 1	type2	type 3
protocol rounds	$log_2(p)$	$log_2(p)$	$log_2(p)$	min: $log_2(p) + 1$ max: $2 \cdot log_2(p)$	$2 \cdot log_2(p)$
messages	$p - 1$	$p - 1$	$p \cdot log_2(p)$	$p \cdot log_2(p)$	$2 \cdot (p - 1)$
OPs	$p - 1$	0	$p \cdot log_2(p)$	$\frac{p}{2} \cdot log_2(p)$	$p - 1$

Summary

- We studied the parallelism of message transfers during communication phases and how this can be illustrated.
- Classification of protocol types:
  type1 (least restrictive, deadlock-prone),
  type2 (less restrictive, deadlock safe),
  type3 (most restrictive, rendezvous forced)
- Message numbering principle, correlated to protocol rounds
- In particular the allreduce communication pattern was studied
- Allreduce: reduction and broadcast phases are fold together, this allows more parallelism than a sequence of reduction and broadcast (but also more messages and more OPs)