# Receive-and-accumulate: optimizing a common application pattern using one-sided ideas in two-sided communication

Jeff R. Hammond[1],

Argonne National Laboratory, Argonne, Illinois
jhammond@alcf.anl.gov

**Abstract.** Both collective and one-sided communication in MPI support the notion of reduction – incoming data is accumulated into the target buffer with well-known operands such as MPI_SUM. On the other hand, point-to-point communication provides only buffer-copy semantics, incoming data can only replace that which is in the target buffer. This is unfortunate, as receive-then-accumulate is a common pattern in application codes. We propose an accumulating receive function, MPI_RECV_ACCUMULATE as one solution to this issue. The benefits of this addition to the MPI standard include: (1) elimination of buffering by the user in favor of pipelining done inside of MPI, (2) symmetry with both collective and one-sided communication, and (3) enabling the exploitation of modern active-message networks. We describe three different implementations of this feature: first, a trivial one with no performance benefits but which can be immediately adopted by any existing implementation; second, a MPICH-oriented implementation that uses both eager and rendezvous protocols where appropriate; and third, an implementation using the IBM PAMI active-message interface. Performance...

## 1 Introduction

cite MPICH [1]

## 2 Justification

## 3 Syntax

The new functions proposed for consideration in MPI are the natural generalizations of MPI_Recv and MPI_Irecv. The MPI_Op argument specifies the reduction operation to be applied to incoming data.

int MPIX_Recv_accumulate(void * buf, int count, MPI_Datatype datatype, MPI_Op op, int source, int tag, MPI_Comm comm, MPI_Status * status)

The nonblocking variant of this function changes the last argument to an MPI request object, by analogy with the regular receive functions.

Restrictions on the usage of these functions are discussed in the next section.

## 4 Semantics

The semantics of Recv_accumulate can be defined by considering a reference implementation using existing MPI functions, which is straightforward for the blocking case (the nonblocking case

```
int MPIX_Recv_accumulate(void * buf, int count, MPI_Datatype datatype,
                         MPI_Op op, int source, int tag, MPI_Comm comm,
                         MPI_Status * status)
{
    void * tmp;
    int rcount, typesize;

    if (op!=MPI_REPLACE) {
        MPI_Type_size(datatype, &typesize);
        MPI_Alloc_mem(count*typesize, MPI_INFO_NULL, &tmp);
        MPI_Recv(tmp, count, datatype, source, tag, comm, status);
        MPI_Get_count(status, datatype, &rcount);
        MPI_Reduce_local(tmp, buf, rcount<count ? rcount : count, datatype, op);
        MPI_Free_mem(tmp);
    }
    else {
        MPI_Recv(buf, count, datatype, source, tag, comm, status);
    }

    return MPI_SUCCESS;
}
```

## 5 Related Work

## 6 Conclusions and Future Work

## Acknowledgments

This work was supported by the U.S. Department of Energy, under Contract DE-AC02-06CH11357.

## References

1. MPICH. http://www.mpich.org/