

# Advanced Message-Passing Programming

---

User-Defined Reduction Operations

# Reusing this material



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

<http://creativecommons.org/licenses/by-nc-sa/4.0/>

This means you are free to copy and redistribute the material and adapt and build on the material under the following terms: You must give appropriate credit, provide a link to the license and indicate if changes were made. If you adapt or build on the material you must distribute your work under the same license as the original.

Note that this presentation contains images owned by others. Please seek their permission before reusing these images.

# Motivation

- Reductions are very highly optimised
- But ...
  - you may want to reduce a non-basic type, e.g. a C structure
  - you may want to implement an operation that is not pre-defined
- Examples
  - summing a compound structure with integer and double members
  - sorting an array

# Procedure (i) (for C)

- Define an MPI datatype for the new object
- E.g for:

```
struct compound{  
    int ival;  
    double dval;  
};
```

- use `MPI_Type_create_struct()` to create an `MPI_COMPOUND`
- then `MPI_Type_commit(&MPI_COMPOUND)` ;

# Procedure (ii)

- Define a function to do the reduction
- Must have this format (e.g. for addition):

```
void my_op (void *invec, void *inoutvec, int *len,  
            MPI_Datatype *datatype)  
{  
    ...  
    for (int i = 0; i < *len; i++)  
        inoutvec[i] = inoutvec[i] + invec[i];  
}
```

# Procedure (iii)

- Register the new function with MPI

```
MPI_Op MPI_SUM_COMPOUND;  
int commute = 1;    //  $a + b = b + a$ 
```

```
MPI_Op_create(my_op, commute, &MPI_SUM_COMPOUND);
```

- Can now use it in reduction call

```
MPI_Allreduce(input, output, n,  
              MPI_COMPOUND, MPI_SUM_COMPOUND, comm)
```

# Sorting

- Imagine each MPI process has a sorted list of values
  - e.g. the top 10 values in an array
- Want to find the top 10 across all processes
  - involves merging the sorted lists
- Is this a valid reduction operation?
- If so, how can it be done

# Summary

- Reduction operations have some restrictions
  - e.g. constant size input and output
- However, despite this there are many occasions where registering your own reduction operation is useful
  - for elegance (working with non-basic types)
  - for performance
- Simple compound example illustrates both elegance *and* performance