#349: Dynamic Window Displacements

RMA WG Plenary Presentation
June 6, 2013

https://svn.mpi-forum.org/trac/mpi-forum-web/ticket/349

What's the problem?

- Dynamic windows require arithmetic on MPI_Aint values that represent addresses
  - Indexing into an array
  - Accessing a field in a struct

- Arithmetic on MPI_Aints that represent addresses cannot be done portably
  - MPI_Aints are signed integers, but addresses ain't (are not)
  - Value is relative to MPI_BOTTOM, which varies across processes and language bindings (e.g., F*****N)
    - MPI_BOTTOM is address of a variable in a common block
  - Arithmetic can overflow!
  - Arithmetic overflow behavior is not specified

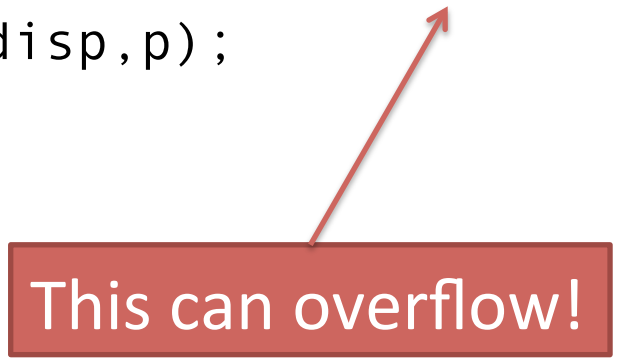
Dynamic Window Displacement Arithmetic Example

Rank p

```
double array[1000];
MPI_Aint array_disp;
…
MPI_Win_attach(array);
MPI_Get_address(array,
    &array_disp);
MPI_Bcast(&array_disp,p);
```

Rank q

```
MPI_Aint array_disp;
…
MPI_Bcast(&array_disp,p);
…
MPI_Get(array_disp +
    i*sizeof(double), …);
```



Proposed Solution

- Add a function that can “safely” add an address and a displacement

- Implementation:
  - Must correctly handle overflows
  - Just add them, if system is two’s complement

Proposed Text (draft)

```
MPI_Aint MPI_Aint_add(MPI_Aint base, MPI_Aint disp)
```

This function produces a new MPI_Aint value that is equivalent to the sum of the *base* and *disp* arguments, where *base* represents an address and *disp* represents a signed integer displacement. The value of *base* may be relative to a non-zero value of MPI_BOTTOM that is unknown at the process performing the call to MPI_Aint_add. The addition is performed in a manner that results in the correct MPI_Aint representation of the output address, as if the process that originally produced *base* had called:

```
MPI_Get_address((char *) base + disp, &out_addr)
```

Straw Polls

- How would you vote for this proposal?
  - (14) yes, (0) no, (4) abstain
- Can we handle this as an erratum?
  - (9) yes, (0) no, (10) abstain