Skip to main content

- Home
- Topics
- Reference
- Glossary
- Help
- Notebook

Virtual Workshop

Welcome guest
Log in (Globus)
Log in (other)
Try the quiz before you start
MPI Collective Communications
Introduction Goals Prerequisites
Characteristics Three Types of Routines Barrier Synchronization Data Movement • Broadcast • Gather and Scatter • Gather/Scatter Effect • Gatherv and Scatterv • Allgather • All to All Global Computing • Reduce • Scan • Operations and Example • Allreduce Mini-Exercise Nonblocking Routines • Nonblocking Example Performance Issues • Two Ways to Broadcast • Two Ways to Scatter Application Example • Scatter vs. Scatterv • Scatterv Syntax
Exercise Quiz
Short survey

MPI Collective Communications: Scatter vs. Scatterv

Let's compare scatter and scatterv in a bit more detail. First, we should recognize that they share some aims in how they move data.

Common purposes of scatter and scatterv:

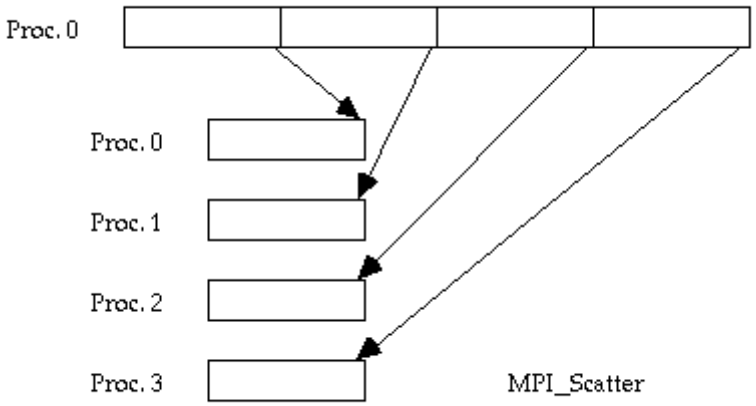
- To send out different chunks of data to different processes
- *Not* to act like a broadcast, in which the same data goes to all

Recall that the goal of a scatter is to distribute different chunks of data to different tasks—in other words, you want to *split up* your data among the processes. Recall also that a scatter is not the same as a broadcast, since in a broadcast, everyone is sent the *same* chunk of data.

Scatter

Requirements of basic scatter:

- Contiguous data
- Uniform message size

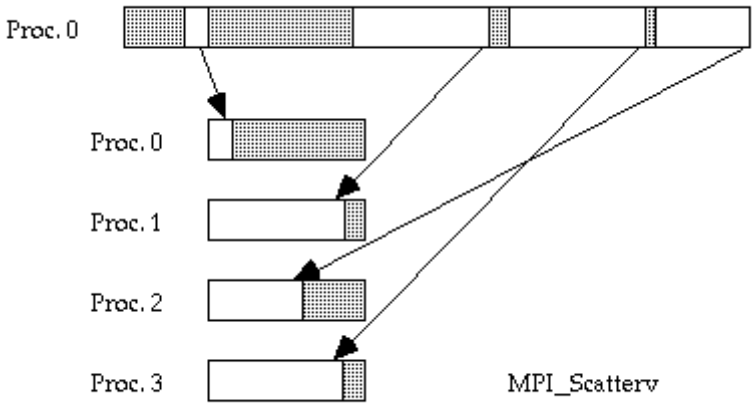


As the figure shows, the basic MPI_Scatter requires that the sender's data are stored in contiguous memory addresses and that the chunks are uniform in size. Thus, the first N data items are sent to the first process in the communicator, the next N items go to the second process, and so on. However, for some applications this might be overly restrictive: what if some processes need fewer than N items, for instance? Should you just pad out the sending buffer with unnecessary data? The answer is no, you should use MPI_Scatterv instead.

Scatterv

Extra capabilities in scatterv:

- Gaps are allowed between messages in source data
  (but the individual messages must still be contiguous)
- Irregular message sizes are allowed
- Data can be distributed to processes in any order



MPI_Scatterv gives you extra capabilities that are most easily described by comparing this figure to the previous one. First of all, you can see that varying numbers of items are allowed to be sent to the different processors. What's more, the first item in each chunk doesn't have to be positioned at some regular spacing away from the first item of the previous chunk. And the chunks don't even have to be stored in the correct sequence!

This is the flexibility that was alluded to earlier. However, MPI has not done away with all the restrictions, because the chunks themselves must be contiguous, and they must not overlap with one another. (Well… if you really, really want to, you can insert skips into a chunk by creating an MPI derived datatype that has a built-in

stride. But that trick may carry with it a severe penalty in efficiency; furthermore, the chunks still can't overlap, because MPI can't count in fractions of a type.)

A safer and more performant alternative, depending on the problem, might be to have a second scatterv send overlapping regions to a second receive buffer. Or in other cases, the redundant reads might be handled by point-to-point communication. Each problem is different, and while MPI often provides a simple and safe solution to problems, getting the optimal solution is not always easy and may even depend on the hardware environment one is using. That being said, MPI routines have generally already been well optimized, so think carefully before trying to use multiple MPI calls in place of a single MPI call.

<= previous next =>