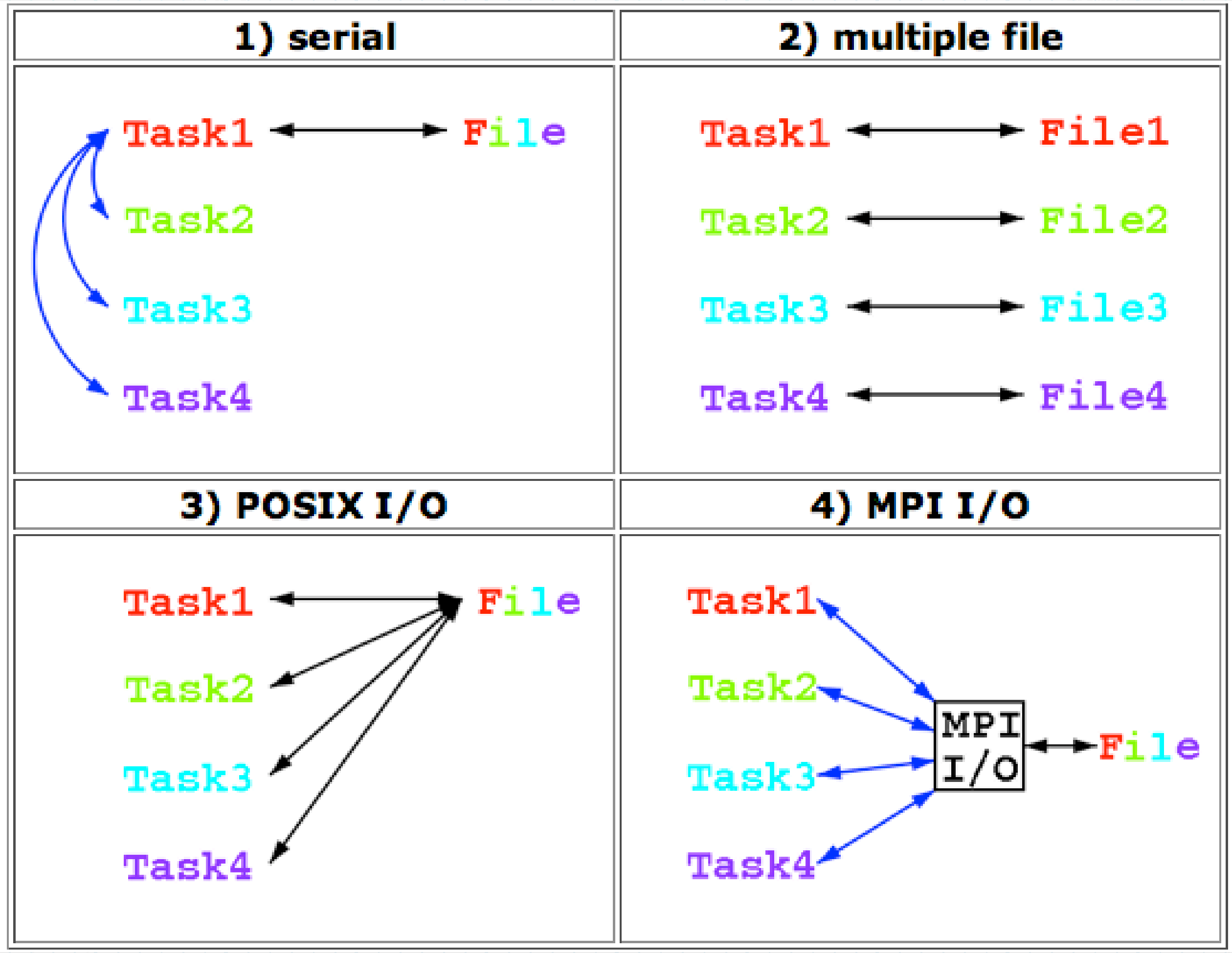
Parallel I/O

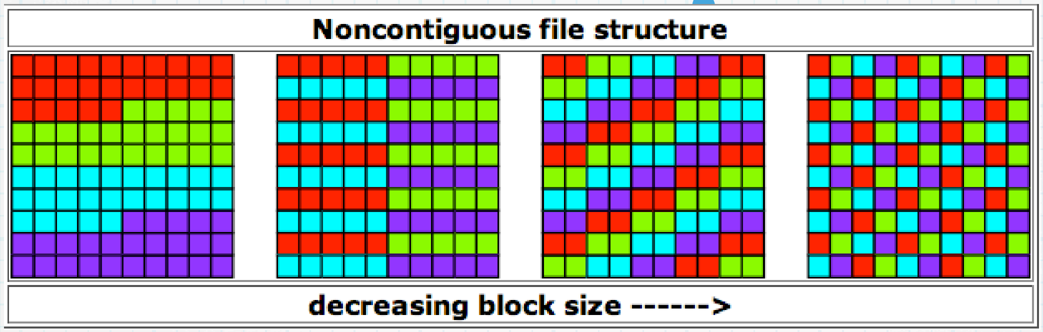
- I/O is already the chief potential bottleneck for serial code - even more so for parallel code.
- Fortran i/o is 'record'-oriented. There is no 'decomposition' of an i/o record that is analagous to data decomposition in a program.

Approaches to Parallel I/O

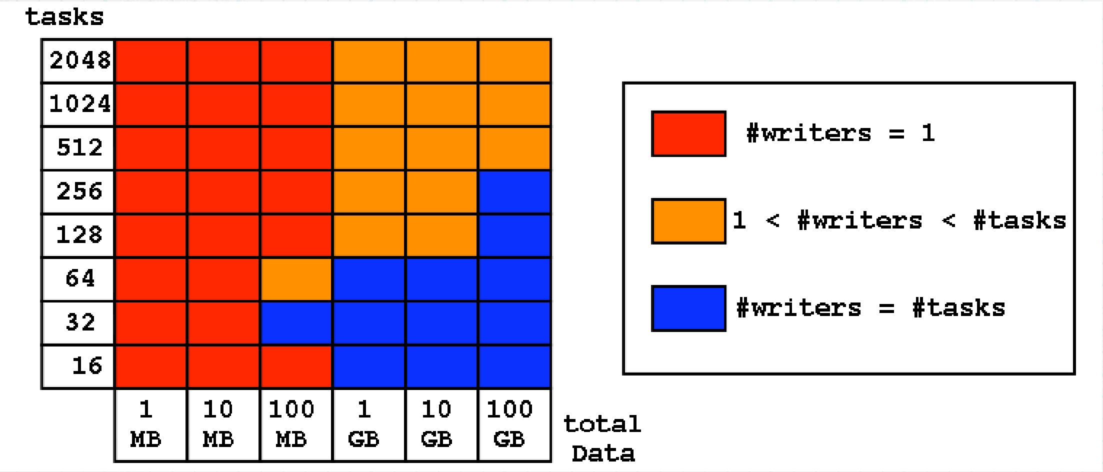
from http://www.nersc.gov/news/reports/technical/seaborg_scaling/io.php



Additional Complications



Can either transpose the data to be contiguous, or perform more, smaller writes.



tasks	1 MB	10 MB	100 MB	1 GB	10 GB	100 GB
2048	🟥	🟥	🟥	🟧	🟧	🟧
1024	🟥	🟥	🟥	🟧	🟧	🟧
512	🟥	🟥	🟥	🟧	🟧	🟧
256	🟥	🟥	🟥	🟧	🟧	🟦
128	🟥	🟥	🟥	🟧	🟧	🟦
64	🟥	🟥	🟧	🟦	🟦	🟦
32	🟥	🟥	🟦	🟦	🟦	🟦
16	🟥	🟥	🟥	🟦	🟦	🟦

total Data

🟥 #writers = 1

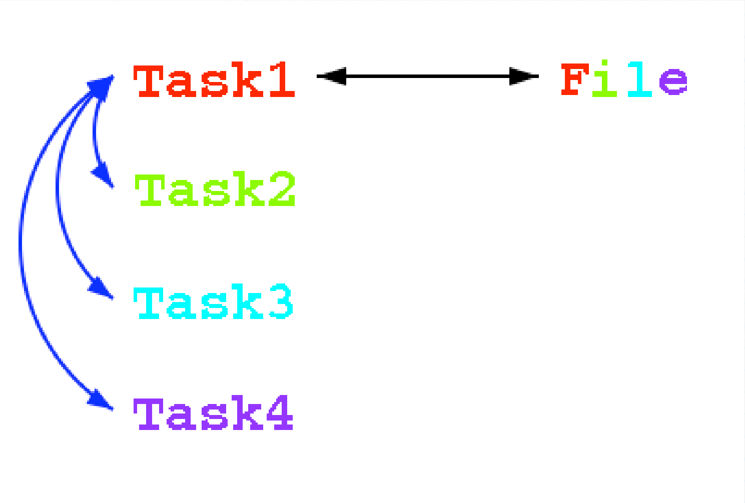
🟧 1 < #writers < #tasks

🟦 #writers = #tasks

Most systems have limits on resources available for i/o that lead to trade-offs affecting i/o strategy.

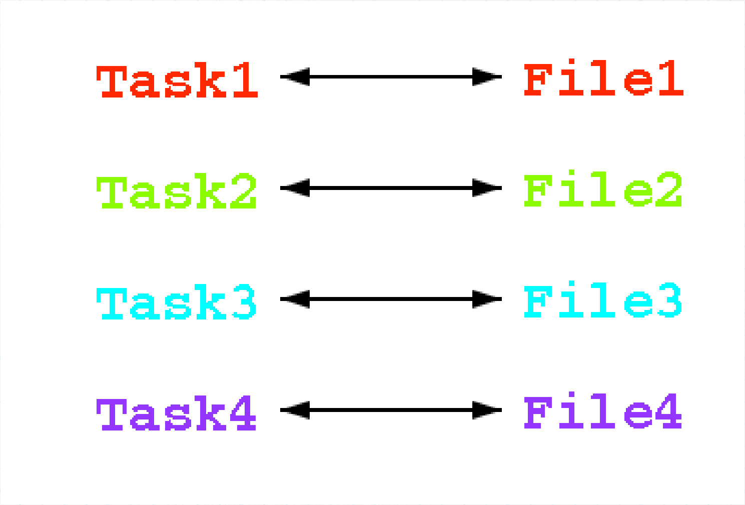
Serial I/O

- No scaling at all.
- May require excessive memory allocation on task1.
- Only used one i/o channel.
- Need gather/scatter mpi code to move the data.



multiple file I/O

- Simplest to code, no mpi communication needed.
- Scales well to a limit.
- Post-processing is needed to reconstruct a global file for visualization.
- If files will be read again, can't be used for a different data decomposition.



Task1 ↔ File1

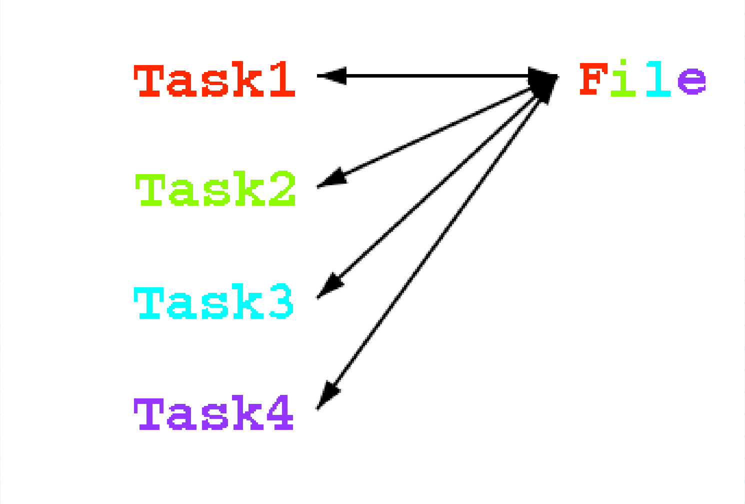
Task2 ↔ File2

Task3 ↔ File3

Task4 ↔ File4

Posix I/O

- Permits overlapping i/o access to a single file asynchonously - not available in fortran, but direct-access i/o is a poor imitation.
- Writing scales negatively, reading ok.
- May saturate i/o channels.
- Efficient i/o this way may not be convenient to visualize.



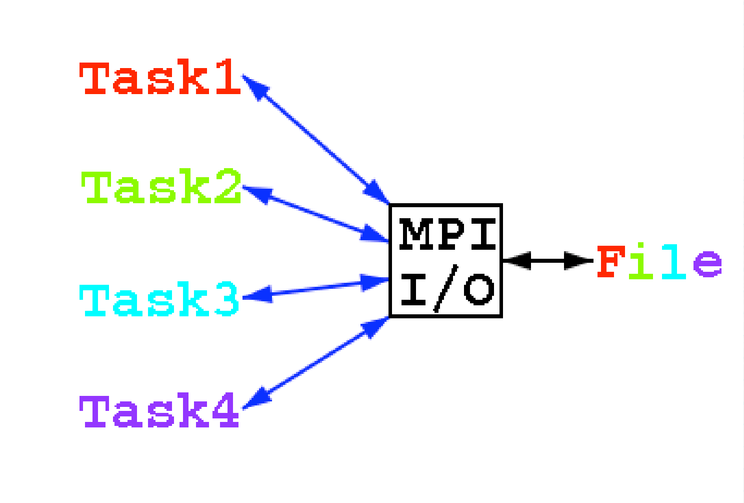
bugs5 example

```
open(4, file=/restarts/model_restart_ocean, &
     form='unformatted', access='DIRECT', recl=lrecl)
if(my_task.eq.0)                               &
   write(4,rec=1,iostat=ioerr)nsecond_dyn, nocean, max_ig_sfc,
lrecl

do ksdm = 1,nsdm_sfc
  do j = 1,jjm_sfc
    do i = 1,iim_sfc
       write(4,rec=1+link_hr_sea( i,j,ksdm), iostat=ioerr )   &
                   sst(i,j,ksdm), iz(i,j,ksdm)
     enddo
   enddo
enddo
close(4)
```

MPI - I/O

- Part of mpi standard
- provides a variety of i/o mechanisms - asynchronous; strided access
- Coordinates the data communication and i/o.
- Can be tuned for optimization.
- The only effective way to do i/o on large numbers of processors.



http://www.nersc.gov/nusers/resources/software/libs/io/mpiio.html

Parallel NetCDF (pnetcdf)

http://www-unix.mcs.anl.gov/parallel-netcdf/

pnetcdf is a netcdf-like interface to mpi-i/o that reads and writes netcdf files.

netcdf

```
call gather_fine(phis, temqp)
call reduce_to_vector_global(temqp, temvec)
if(my_task == 0) then
  status = nf_create( 'gatm_serial.g2.nc',NF_CLOBBER, ncfid)
  status = nf_put_att_int(ncfid, NF_GLOBAL, "total_grid_size",   &
                          nf_int, 1, max_ig)
  status = nf_def_dim (ncfid, "grid_cells", max_ig, gridcellsID)
  tmpdim = (/gridcellsID/)
  status = nf_def_var (ncfid, 'zs', NF_FLOAT, 1, tmpdim, qpvid)
  status = nf_put_att_text (ncfid, qpvid, 'title',               &
                           20, 'Surface elevation    ')
  status = nf_put_att_text (ncfid, qpvid, 'units', 1, 'm')
  status = nf_enddef (ncfid)
  status = nf_put_var_real (ncfid, qpvid, temvec)
  status = nf_close(ncfid)
endif
```

pnetcdf

```
status = nfmpi_create ( mpi_comm_atmos, "gatm_parallel.g2.nc",     &
                        nf_clobber, mpi_info_null, ncidp )
status = nfmpi_put_att_int(ncidp, NF_GLOBAL, "total_grid_size",    &
                           nf_int, 1_mpi_offset_kind, max_ig)
status = nfmpi_def_dim (ncidp, "grid_cells", clen, gridcellsID)
status = nfmpi_inq_dimid (ncidp, "grid_cells", tmpdim(1))
status = nfmpi_def_var (ncidp, 'zs', NF_FLOAT, 1, tmpdim, qpvid)
status = nfmpi_put_att_text (ncidp, qpvid, 'title',                &
                          20_mpi_offset_kind, 'Surface elevation
')
status = nfmpi_put_att_text (ncidp, qpvid, 'units',                &
                             1_mpi_offset_kind, 'm')
status = nfmpi_enddef (ncidp)
do ksdm = 1,nsdm
  do j = 2,jjm-1
    start(1) = grid_center_index(2,j,ksdm)
    count(1) = iim-2
    status = nfmpi_put_vara_real_all (ncidp, qpvid, start, count, &
                                      phis(2,j,ksdm) ,count(1))
  enddo
enddo
status = nfmpi_begin_indep_data(ncidp)
do ksdm = 1,nsdm
  if(polygon_type(1,jjm-1,ksdm) == 3._dbl_kind) then
    start(1) = 1
    count(1) = 1
    status = nfmpi_put_vara_real (ncidp, qpvid, start, count,      &
                                  phis(1,jjm-1,ksdm) ,count(1))
  endif
  if(polygon_type(iim-1,1,ksdm) == 4._dbl_kind) then
    start(1) = 2
    count(1) = 1
    status = nfmpi_put_vara_real (ncidp, qpvid, start, count,      &
                                  phis(iim-1,1,ksdm) ,count(1))
  endif
enddo
status = nfmpi_end_indep_data(ncidp)
status = nfmpi_close ( ncidp )
```