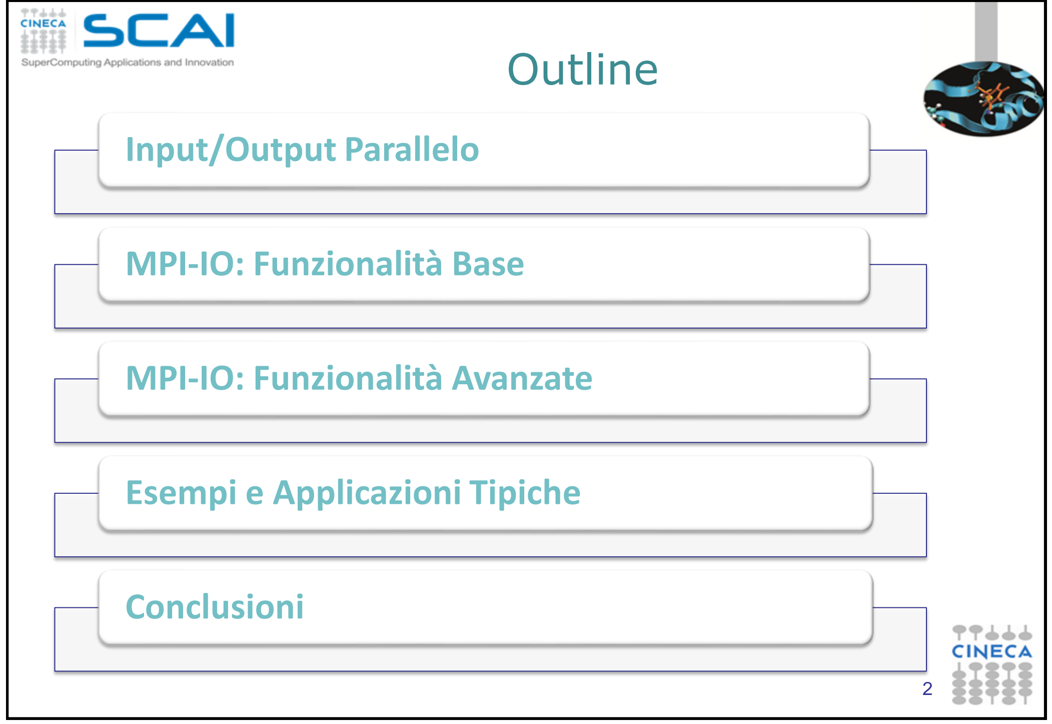
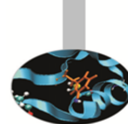
SCAI
SuperComputing Applications and Innovation
I/O Parallelo con MPI
Claudia Truini
c.truini@cineca.it
Luca Ferraro
l.ferraro@cineca.it
Vittorio Ruggiero
v.ruggiero@cineca.it
CINECA


SCAI
SuperComputing Applications and Innovation
Outline
Input/Output Parallelo
MPI-IO: Funzionalità Base
MPI-IO: Funzionalità Avanzate
Esempi e Applicazioni Tipiche
Conclusioni
2
CINECA

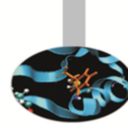
Approcci Tradizionali all' IO Parallelo



- Molte applicazioni parallele, per ragioni storiche, gestivano l'IO in uno dei seguenti modi:
  - **master-slave:** solo un processo gestisce IO e scambia dati con gli altri processi tramite comunicazioni
    - ulteriore memoria per bufferizzare i dati sul master
    - non scalabile (serializzazione operazioni)
  - **accesso disgiunto:** ogni processo gestisce file diversi
    - difficile post-processing e gestione di molti file
    - limitazioni sul restart con numero processi differente
  - **accesso controllato:** ogni processo accede al medesimo file in posizioni disgiunte
    - difficile controllo nei pattern di accesso più comuni
    - error prone
    - rischio di serializzazione (filesystem che non supportano lock multipli)

3

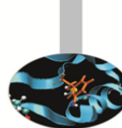
Approccio Moderno all'IO Parallelo



- Nelle moderne applicazioni di supercalcolo, *la gestione dell'IO assume un ruolo critico in fase di disegno e progettazione dell'applicazione*
  - l'aumento delle capacità di calcolo ha aumentato la richiesta di capacità e prestazioni dei sistemi di storage
  - la quantità di dati da salvare è spesso molto grande
  - la latenza di accesso ai dischi rimane ancora molto alta
- soluzioni per non far diventare l'IO un collo di bottiglia per il calcolo
  - file-system paralleli efficienti sono ormai disponibili su tutte le architetture di supercalcolo
  - architetture e librerie software capaci di gestire accessi paralleli in modo efficiente, sfruttando il file-system parallelo

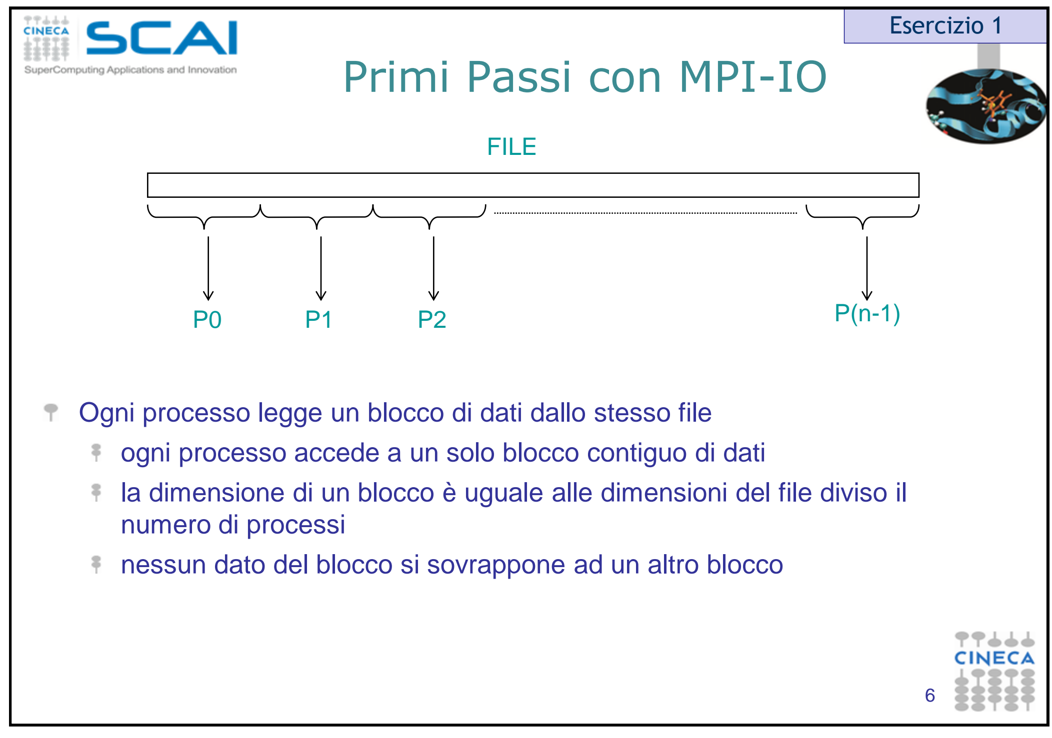
4

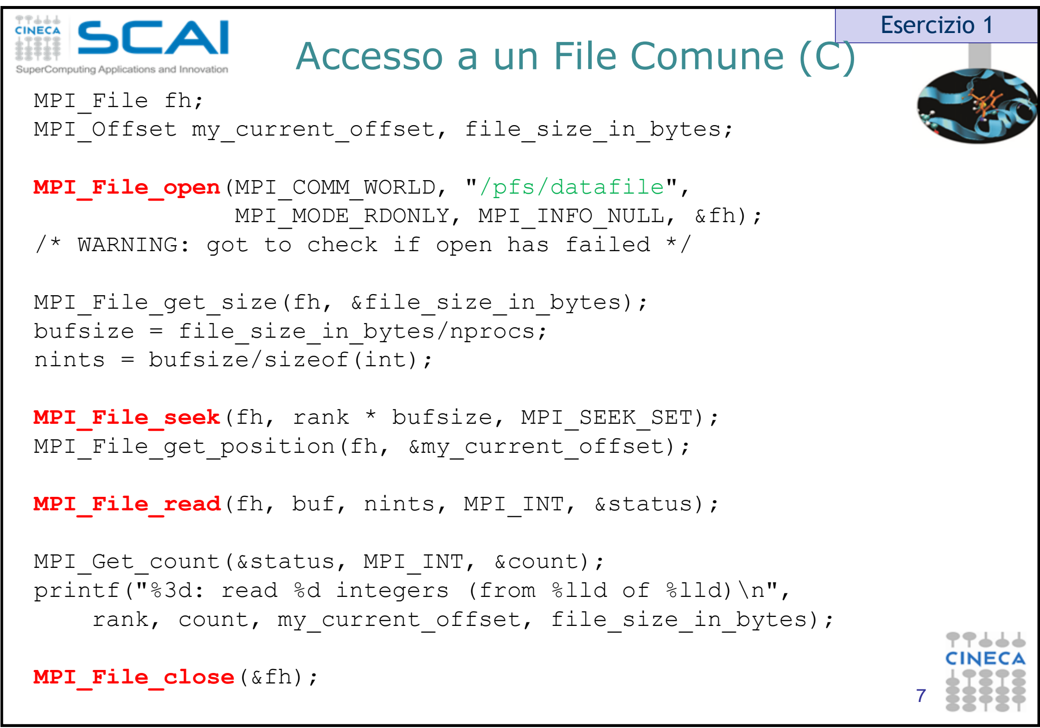
Funzionalità MPI-2.x per IO Parallelo



- MPI-IO: introdotto nello standard MPI-2.x (1997)
  - leggere/scrivere un file è come ricevere/mandare un messaggio da un buffer MPI
  - accesso ottimizzato a dati non contigui
  - operazioni di accesso singole e collettive con comunicatori
  - modalità blocking e non-blocking
  - portabilità dei file (implementation/system independent)
  - good performance nella maggior parte delle implementazioni
- due buone ragioni per cominciare a usarlo:
  - semantica molto semplice
  - performance : 32 processi (4x8) con griglie locali 100002 (dp)
    - MPI-IO: **48**sec vs Traditional-IO: **3570**sec (scrittura di 24Gb)

5





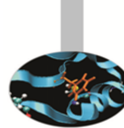
Esercizio 1

Accesso a un File Comune (Fortran)

```
 integer mpi_integer_size
 integer (kind=MPI_OFFSET_KIND) offset
! in F77, see implementation notes (might be integer*8)

  call MPI_FILE_OPEN(MPI_COMM_WORLD, '/pfs/datafile', &
            MPI_MODE_CREATE + MPI_MODE_WRONLY, &
            MPI_INFO_NULL, fh, ierr)
! WARNING: got to check if open has failed

  call MPI_TYPE_SIZE(MPI_INTEGER, mpi_integer_size, ierr)

  offset = rank * nints * mpi_integer_size
  call MPI_FILE_SEEK(fh, offset, MPI_SEEK_SET, ierr)

  call MPI_FILE_WRITE(fh, buf, nints, MPI_INTEGER, status, ierr)

  call MPI_GET_COUNT(status, MPI_INTEGER, count, ierr)

  print *, 'process ', rank, ` wrote ', count, 'integers'

  call MPI_FILE_CLOSE(fh, ierr)
```

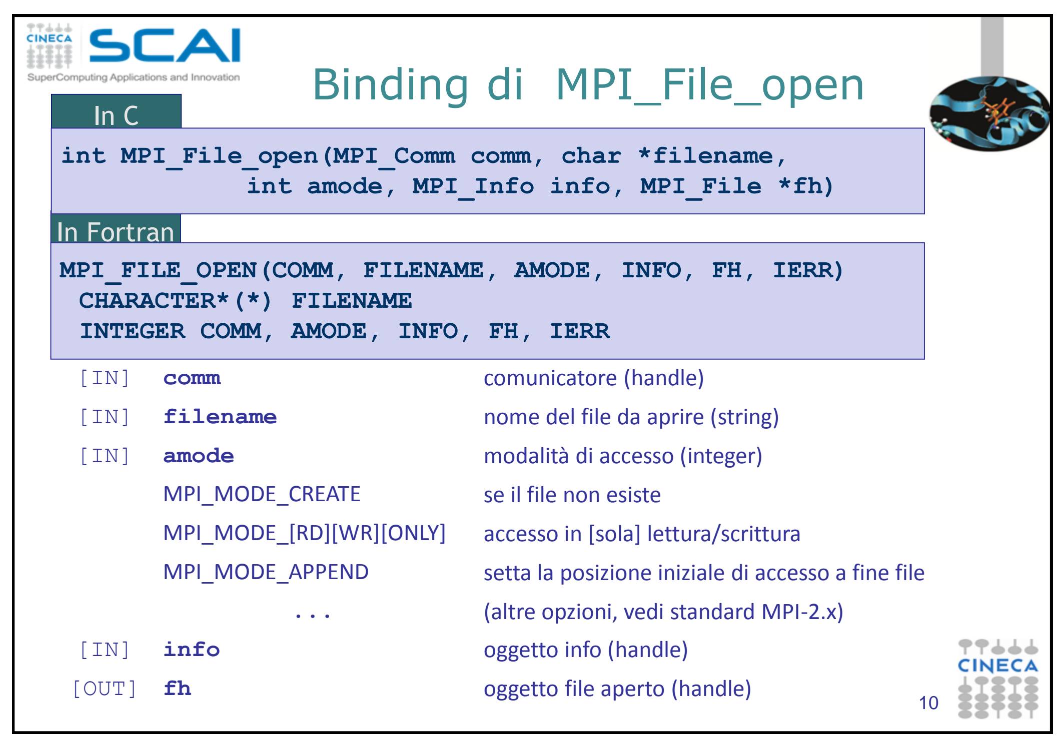
8

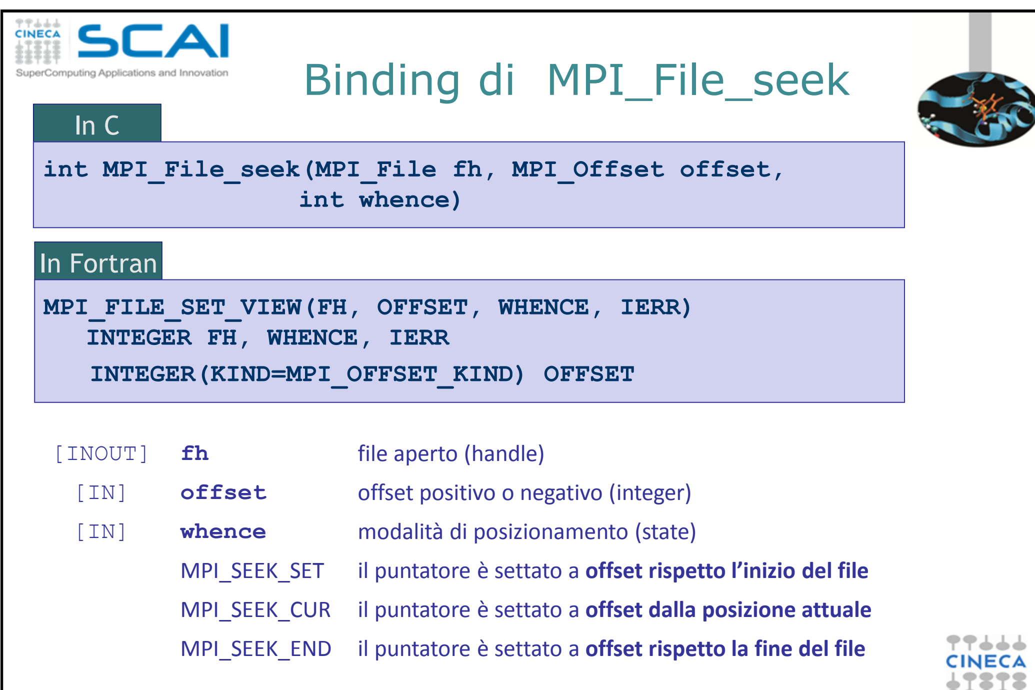
Funzionalità Elementari di MPI-IO

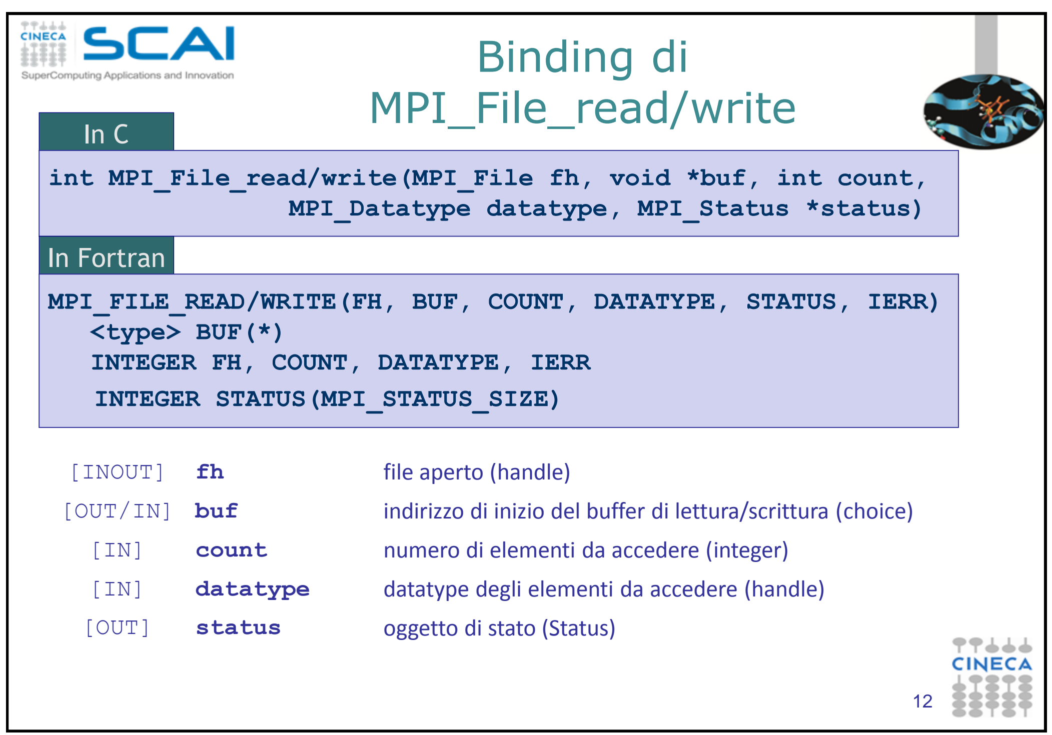


- MPI-IO fornisce funzionalità che ci si aspetta per IO:
  - open, seek, read, write, close (ecc.)
- open/close sono operazioni collettive su medesimo file
  - modalità di accesso multiple (combinabili: |,+)
- read/write sono simili a Send/Recv di dati a/da buffer
  - ogni processo controlla un proprio puntatore locale al file (individual file pointer) tramite operazioni seek,read,write
  - la variabile offset è di tipo speciale ed è espresso in unità elementare (etype) di accesso al file (default in byte)
    - errore tipico: dichiarare offset come intero di default
  - è possibile interrogare lo status di ritorno per verifiche
- chiamate a funzionalità accessorie informative

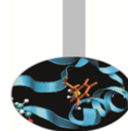
9




SCAI
SuperComputing Applications and Innovation
Binding di MPI_File_seek
In C
int MPI_File_seek(MPI_File fh, MPI_Offset offset,
int whence)
In Fortran
MPI_FILE_SET_VIEW(FH, OFFSET, WHENCE, IERR)
INTEGER FH, WHENCE, IERR
INTEGER(KIND=MPI_OFFSET_KIND) OFFSET
[INOUT] fh file aperto (handle)
[IN] offset offset positivo o negativo (integer)
[IN] whence modalità di posizionamento (state)
MPI_SEEK_SET il puntatore è settato a offset rispetto l'inizio del file
MPI_SEEK_CUR il puntatore è settato a offset dalla posizione attuale
MPI_SEEK_END il puntatore è settato a offset rispetto la fine del file
11
CINECA


SCAI
SuperComputing Applications and Innovation
Binding di
MPI_File_read/write
In C
int MPI_File_read/write(MPI_File fh, void *buf, int count,
MPI_Datatype datatype, MPI_Status *status)
In Fortran
MPI_FILE_READ/WRITE(FH, BUF, COUNT, DATATYPE, STATUS, IERR)
<type> BUF(*)
INTEGER FH, COUNT, DATATYPE, IERR
INTEGER STATUS(MPI_STATUS_SIZE)
[INOUT] fh file aperto (handle)
[OUT/IN] buf indirizzo di inizio del buffer di lettura/scrittura (choice)
[IN] count numero di elementi da accedere (integer)
[IN] datatype datatype degli elementi da accedere (handle)
[OUT] status oggetto di stato (Status)
12
CINECA

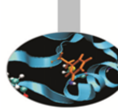
Funzionalità Avanzate di MPI-IO



- Le funzionalità base sarebbero sufficienti per gestire IO
- Poco efficienti nella maggior parte dei casi di interesse
  - gli algoritmi paralleli introducono una distribuzione dei dati non contigua e frammentaria in memoria e di accesso su file
    - ghost cells: riordinamento dati in memoria locale
    - block/cyclic array distributions: associazione di dati locali per processo non contigua su file
  - read/write multiple per segmenti così frammentati porta un enorme degrado di performance (per alta latenza IO)
- MPI-IO consente di accedere ai dati in modo:
  - non contiguo su file: specificando pattern di accesso (fileview)
  - non contiguo in memoria: specificando nuovi datatype
  - collettivo: raggruppando accessi multipli vicini in uno o più accessi singoli (abbattendo le penalità di latenza)

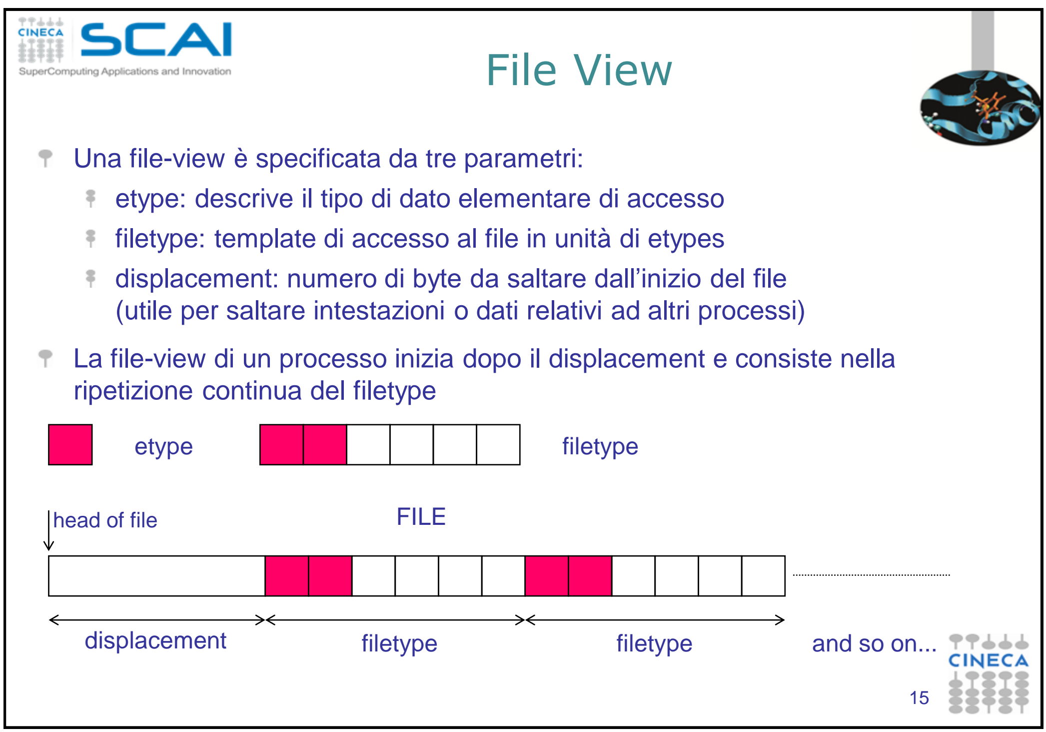
13

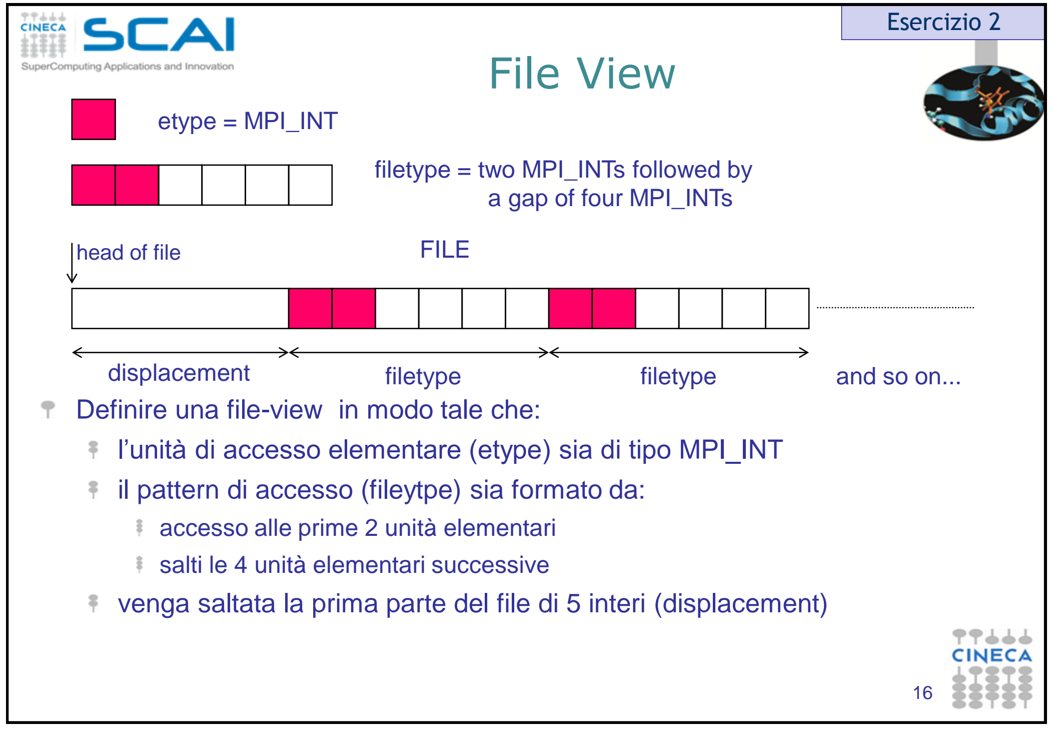
File View

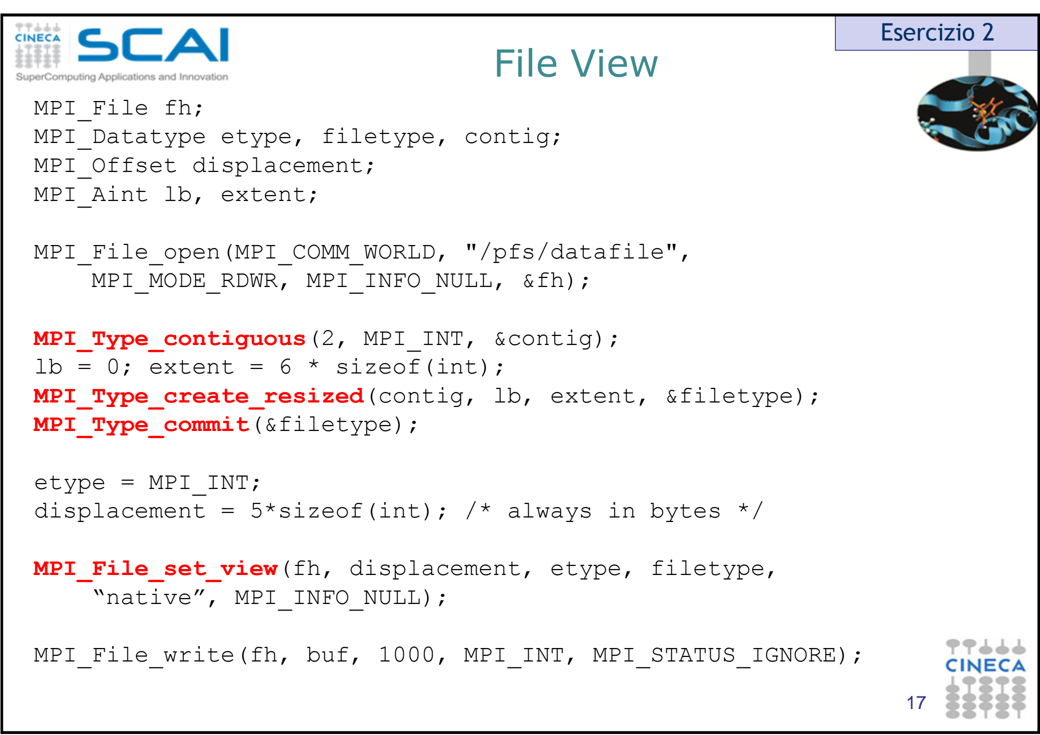


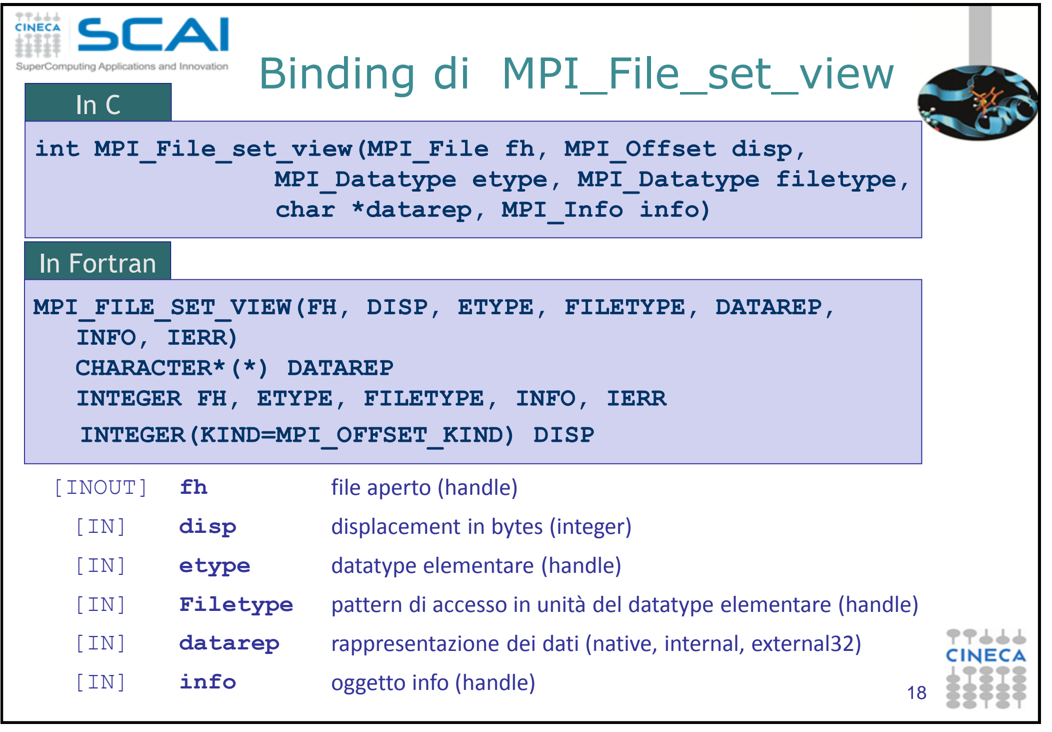
- Una file-view definisce la porzione di dati di un file visibile a un processo
  - read/write possono accedere solo ai dati visibili
  - dati non visibili vengono semplicemente saltati
- Quando un file viene aperto da un gruppo di processi, viene assegnata a tutti una file-view di default
  - file visibile da tutti i processi
  - dati accessibili come stream continuo di MPI_BYTE
- E' possibile definire una file-view differente per ogni processo
  - utile per mappare i dati su processi diversi senza overlap

14

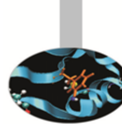

SCAI
SuperComputing Applications and Innovation
File View
Una file-view è specificata da tre parametri:
etype: descrive il tipo di dato elementare di accesso
filetype: template di accesso al file in unità di etypes
displacement: numero di byte da saltare dall'inizio del file (utile per saltare intestazioni o dati relativi ad altri processi)
La file-view di un processo inizia dopo il displacement e consiste nella ripetizione continua del filetype
etype
filetype
head of file
FILE
displacement
filetype
filetype
and so on...
CINECA
15


Esercizio 2
SCAI
SuperComputing Applications and Innovation
File View
etype = MPI_INT
filetype = two MPI_INTs followed by a gap of four MPI_INTs
head of file
FILE
displacement
filetype
filetype
and so on...
Definire una file-view in modo tale che:
l'unità di accesso elementare (etype) sia di tipo MPI_INT
il pattern di accesso (fileytpe) sia formato da:
accesso alle prime 2 unità elementari
salti le 4 unità elementari successive
venga saltata la prima parte del file di 5 interi (displacement)
CINECA
16


SCAI
SuperComputing Applications and Innovation
Esercizio 2
File View
MPI_File fh;
MPI_Datatype etype, filetype, contig;
MPI_Offset displacement;
MPI_Aint lb, extent;
MPI_File_open(MPI_COMM_WORLD, "/pfs/datafile",
MPI_MODE_RDWR, MPI_INFO_NULL, &fh);
MPI_Type_contiguous(2, MPI_INT, &contig);
lb = 0; extent = 6 * sizeof(int);
MPI_Type_create_resized(contig, lb, extent, &filetype);
MPI_Type_commit(&filetype);
etype = MPI_INT;
displacement = 5*sizeof(int); /* always in bytes */
MPI_File_set_view(fh, displacement, etype, filetype,
"native", MPI_INFO_NULL);
MPI_File_write(fh, buf, 1000, MPI_INT, MPI_STATUS_IGNORE);
CINECA
17


SCAI
SuperComputing Applications and Innovation
Binding di MPI_File_set_view
In C
int MPI_File_set_view(MPI_File fh, MPI_Offset disp,
MPI_Datatype etype, MPI_Datatype filetype,
char *datarep, MPI_Info info)
In Fortran
MPI_FILE_SET_VIEW(FH, DISP, ETYPE, FILETYPE, DATAREP,
INFO, IERR)
CHARACTER*(*) DATAREP
INTEGER FH, ETYPE, FILETYPE, INFO, IERR
INTEGER(KIND=MPI_OFFSET_KIND) DISP
[INOUT] fh file aperto (handle)
[IN] disp displacement in bytes (integer)
[IN] etype datatype elementare (handle)
[IN] Filetype pattern di accesso in unità del datatype elementare (handle)
[IN] datarep rappresentazione dei dati (native, internal, external32)
[IN] info oggetto info (handle)
CINECA
18

Rappresentazione dei Dati

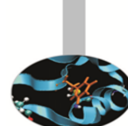


- data representation: definisce la modalità di rappresentazione e accesso ai dati (byte order, type sizes, ecc)
  - **native:** (default) rappresentazione usata in memoria senza conversione
    - nessuna perdita di precisione o prestazioni di IO per conversione
    - non portabile
  - **internal:** rappresentazione implementation-dependent della libreria MPI in uso
    - portabile su macchine che usano la stessa implementazione MPI
  - **external32:** standard definito da MPI (32-bit big-endian IEEE)
    - portabile su ogni architettura e su ogni implementazione di MPI
    - qualche overhead di conversione e perdita di precisione
- Per modalità internal e external32 la portabilità è garantita solo se si usano i corretti MPI datatype per leggere e scrivere (non con MPI_BYTE)

CINECA

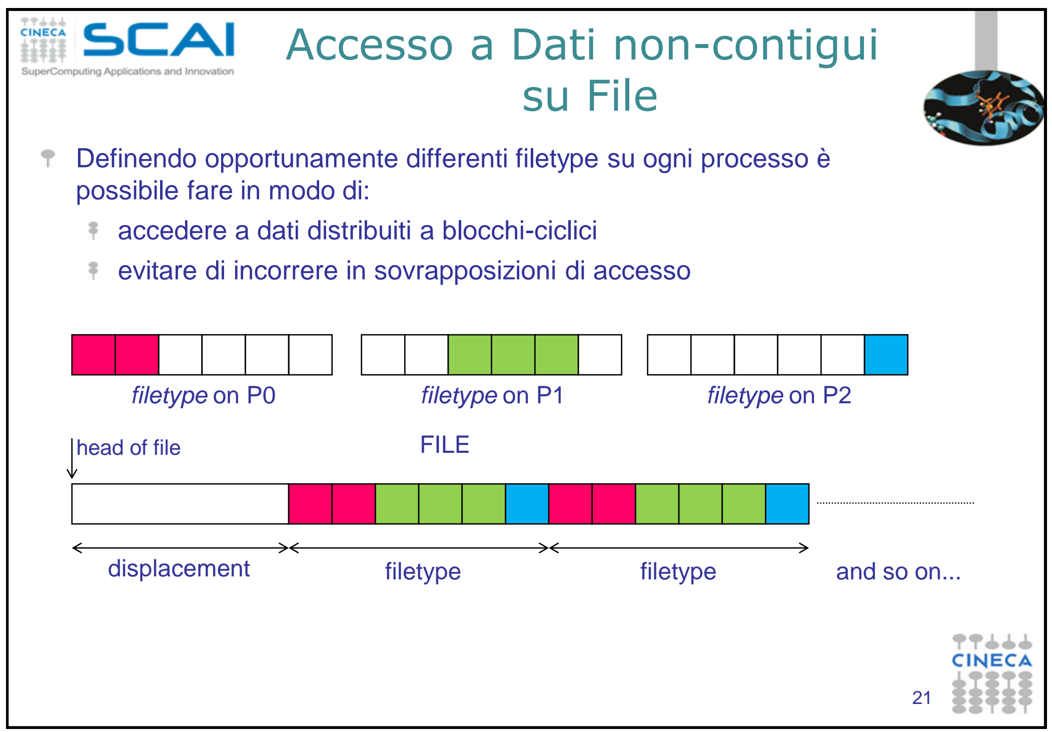
19

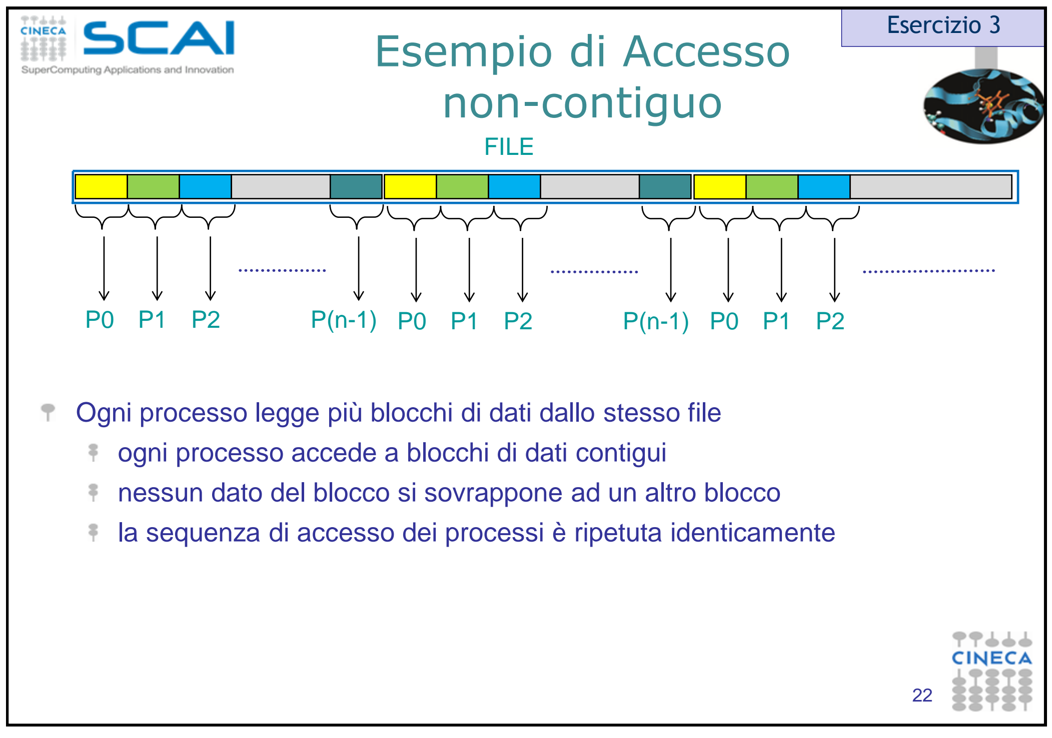
Passare Informazioni sul Filesystem



- MPI consente all'utente di fornire informazioni sulle caratteristiche del file-sytem in uso
  - sono opzionali
  - potrebbe migliorare le performance
  - dipendono dall'implementazione di MPI
  - default: passare MPI_INFO_NULL se non si vuole specificarle
- Le info sono oggetti creati da MPI_Info_create
  - elementi di tipo chiave/valore
  - utilizzare MPI_Info_set per aggiungere elementi
- ... riferirsi allo standard per maggiori informazioni
  - vedi ad esempio implementazione di ROMIO di MPICH
  - info specifiche per vari filesystem (PFS, PVFS, GPFS, Lustre, ...)

20


SCAI
SuperComputing Applications and Innovation
Accesso a Dati non-contigui su File
Definendo opportunamente differenti filetype su ogni processo è possibile fare in modo di:
accedere a dati distribuiti a blocchi-ciclici
evitare di incorrere in sovrapposizioni di accesso
filetype on P0
filetype on P1
filetype on P2
head of file
FILE
displacement
filetype
filetype
and so on...
CINECA
21

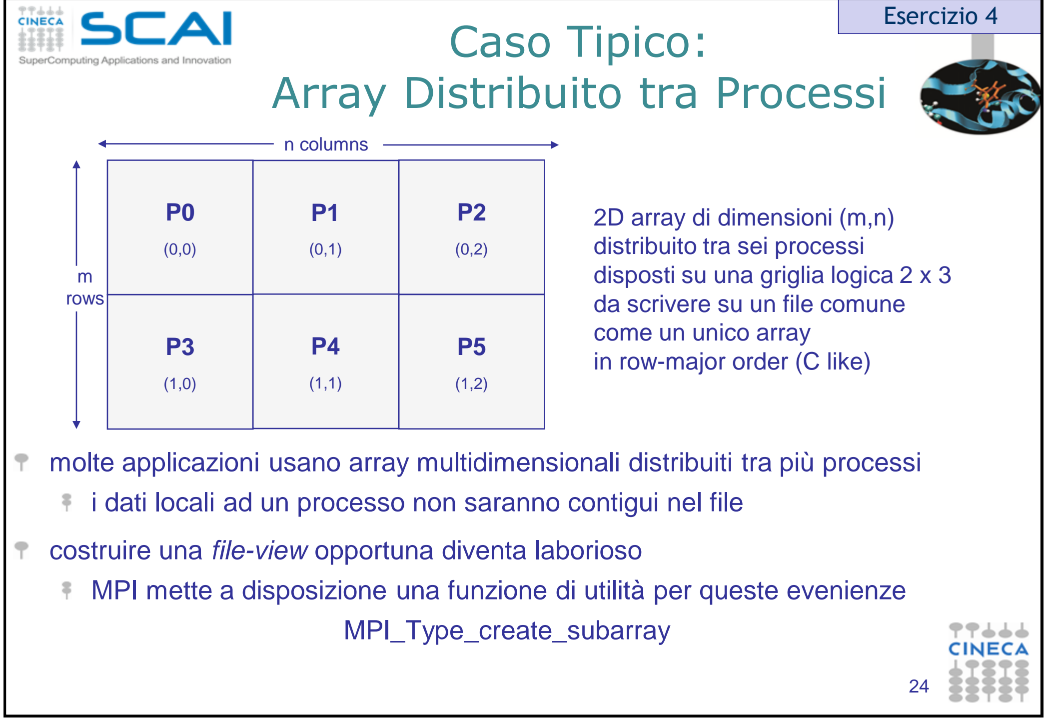

SCAI
SuperComputing Applications and Innovation
Esercizio 3
Esempio di Accesso non-contiguo
FILE
P0 P1 P2 P(n-1) P0 P1 P2 P(n-1) P0 P1 P2
Ogni processo legge più blocchi di dati dallo stesso file
ogni processo accede a blocchi di dati contigui
nessun dato del blocco si sovrappone ad un altro blocco
la sequenza di accesso dei processi è ripetuta identicamente
CINECA
22

SCAI SuperComputing Applications and Innovation

Esercizio 3

```
MPI_File fh;
MPI_Datatype filetype;
MPI_Offset displacement;

MPI_File_open(MPI_COMM_WORLD, "/pfs/datafile",
    MPI_MODE_RDONLY, MPI_INFO_NULL, &fh);
/* WARNING: got to check if open has failed */

MPI_Type_vector(nints/ints_per_block, ints_per_block,
    ints_per_block*nprocs, MPI_INT, &filetype);
MPI_Type_commit(&filetype);

displacement = ints_per_block*sizeof(int)*rank;
MPI_File_set_view(fh, displacement, etype, filetype,
    "native", MPI_INFO_NULL);

MPI_File_read(fh, buf, nints, MPI_INT, MPI_STATUS_IGNORE);
...
MPI_File_close(&fh);
MPI_Type_free(&filetype);
```

23



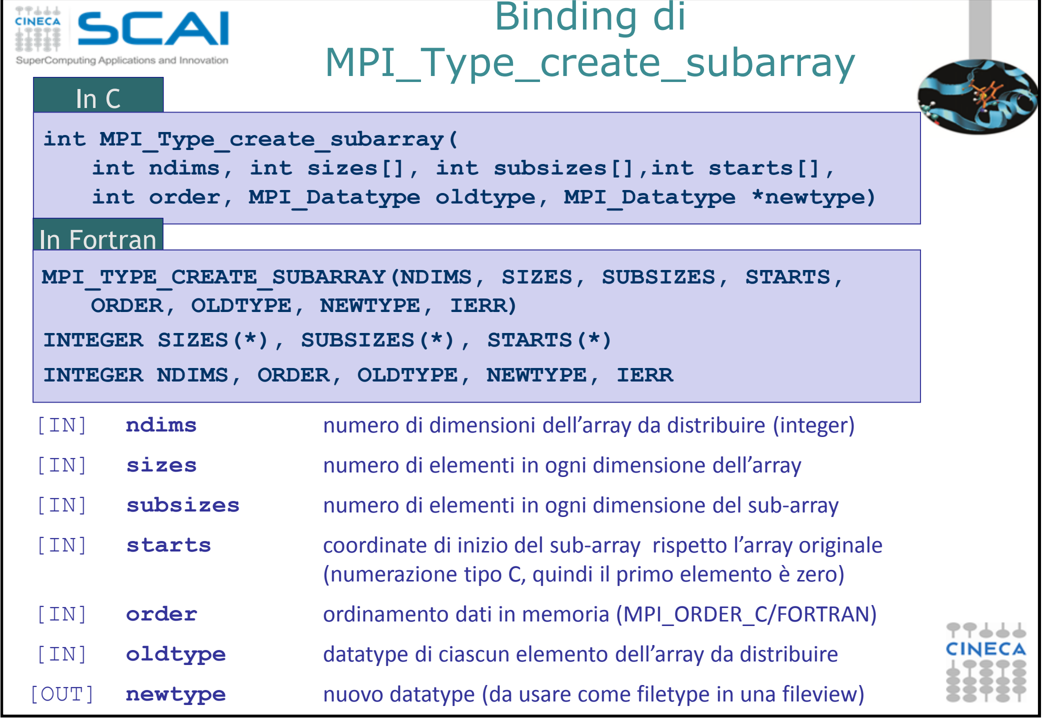
SCAI SuperComputing Applications and Innovation — CINECA

Esercizio 4

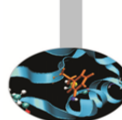
MPI_Type_create_subarray

```
gsizes[0] = m;  /* no. of rows in global array */
gsizes[1] = n;  /* no. of columns in global array*/

psizes[0] = 2; /* no. of procs. in vertical dimension */
psizes[1] = 3; /* no. of procs. in horizontal dimension */

lsizes[0] = m/psizes[0]; /* no. of rows in local array */
lsizes[1] = n/psizes[1]; /* no. of columns in local array */

dims[0] = 2; dims[1] = 3;
periods[0] = periods[1] = 1;
MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, 0, &comm);
MPI_Comm_rank(comm, &rank);
MPI_Cart_coords(comm, rank, 2, coords);

/* global indices of first element of local array */
start_indices[0] = coords[0] * lsizes[0];
start_indices[1] = coords[1] * lsizes[1];

MPI_Type_create_subarray(2, gsizes, lsizes, start_indices,
                         MPI_ORDER_C, MPI_FLOAT, &filetype);
MPI_Type_commit(&filetype);
```

25



Collective Access

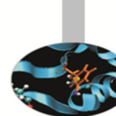


- il nuovo filetype può essere utilizzato per definire la nuova vista del file e accedere o scrivere i dati

```
MPI_File_open(MPI_COMM_WORLD, "/pfs/datafile",
              MPI_MODE_CREATE | MPI_MODE_WRONLY,
              MPI_INFO_NULL, &fh);

MPI_File_set_view(fh, 0, MPI_FLOAT, subarray_defined_file_view,
             "native", MPI_INFO_NULL);

local_array_size = num_local_rows * num_local_cols;
MPI_File_write_all(fh, local_array, local_array_size,
                MPI_FLOAT, &status);

MPI_File_close(&fh);
```

27

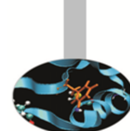
Collective Access



- MPI_File_read/write sono locali al processo chiamante
  - sebbene si acceda allo stesso file, non c'è coordinazione tra processi
  - l'implementazione non può fare alcuna assunzione sull'attività degli altri processi
  - deve soddisfare ogni richiesta individualmente
- MPI-IO mette a disposizione read/write collettive
- MPI_File_read_all / MPI_File_write_all
  - l'implementazione può adottare strategie di ottimizzazione
    - coordinare gli accessi, unificare la lettura in blocchi più grandi
    - implementazione two-phase I/O o collective buffering
  - guadagno di performance molto significativi
  - la funzione deve essere chiamata da tutti i processi del comunicatore con cui si è aperto il file

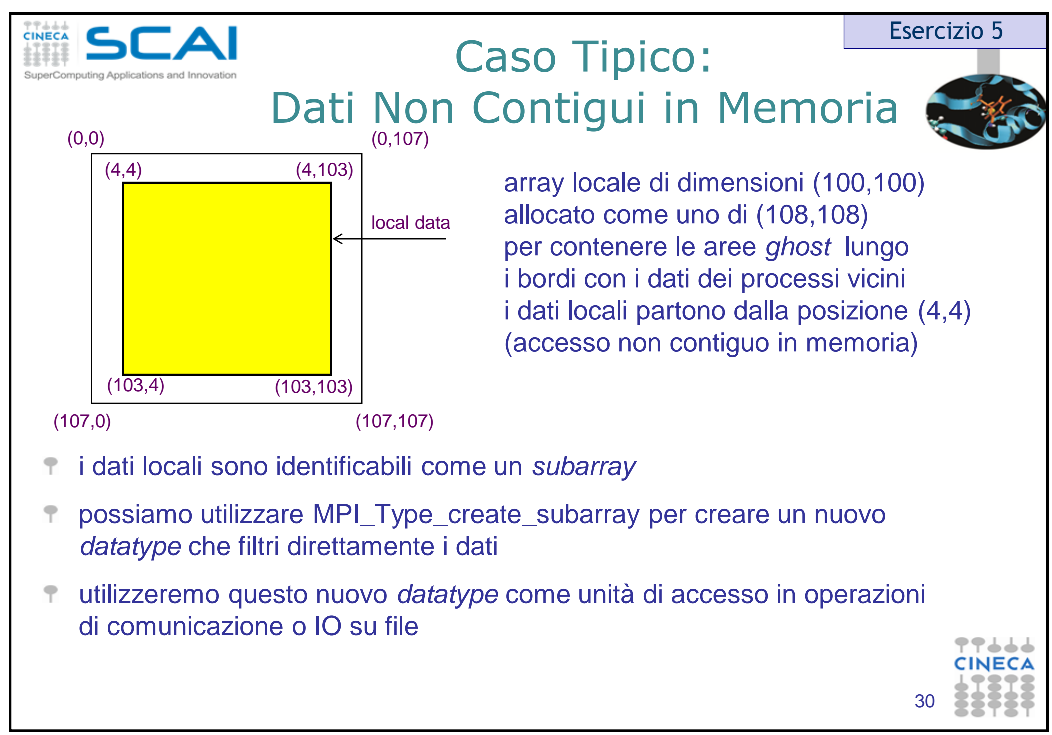
28

Non-contiguous Memory Access

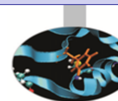


- Molte implementazioni algoritmiche utilizzano array locali sovra dimensionati per poter ospitare una copia dei dati assegnati ai processi vicini (ghost cells)
- I moderni linguaggi di programmazione a oggetti consentono la descrizione e l'operabilità su strutture dati più complesse
  - aggregazione di informazioni e dati su ogni elemento
- Queste situazioni portano una frammentazione dei dati della stessa natura in memoria
  - accessi non contigui in caso di read/write da memoria
  - utilizzare buffer per riordinamento è costoso e poco efficiente
- Utilizzare derived datatype per accedere con pattern

CINECA 29



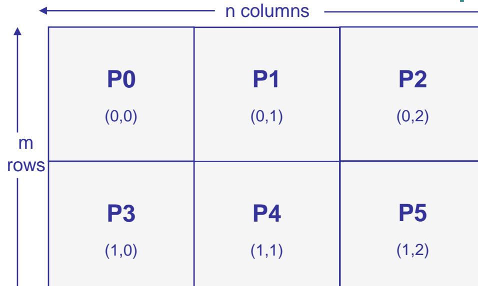
Non-contiguous Memory Access (ghost cells)



```
/* create a derived datatype describing the layout of
   local array in memory buffer that includes ghosts
   This is just another sub-array datatype!
*/
memsizes[0] = lsizes[0] + 8; /* rows in allocated array */
memsizes[1] = lsizes[1] + 8; /* columns in allocated array */

/* indices of first local elements in the allocated array */
start_indices[0] = start_indices[1] = 4;

MPI_Type_create_subarray(2, memsizes, lsizes, start_indices,
         MPI_ORDER_C, MPI_FLOAT, &memtype);
MPI_Type_commit(&memtype);

/* create filetype and set fileview as in subarray example */
...

/* write local data as one big new datatype */
MPI_File_write_all(fh, local_array, 1, memtype, &status);
```

31

Un Semplice Benchmark



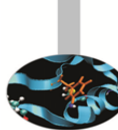
- **Traditional I/O:** gather dei dati su master per righe/colonne successive di processi
- **MPI-IO:** uso di MPI_Type_create_subarray per definire la file-view su ogni processo seguito da una scrittura collettiva
- griglia locale 10000x10000 double-precision su ogni processo

processi	1	2	8	16	32
filesize (Mb)	763	1526	6103	12207	24414
Traditional-IO (s)	8	22	86	1738	3570
MPI-IO (s)	1	2	18	33	48

Benchmark di I/O dipendono da molti fattori quali il tipo di filesystem utilizzato, l'infrastuttura di storage in uso, l'implementazione MPI-IO, la rete di comunicazione, etc.

32

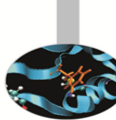
Consigli d'uso



- Pianificare sempre l'IO in fase di progettazione dell'applicazione
  - massimizzare il parallelismo
  - unire, se possibile, restart file con dati simulazione
  - output formattato solo per file che leggerete con gli occhi
- mitigare la latenza di accesso al file-system
  - trasferire blocchi di dati più grandi possibili per chiamata
  - utilizzare accessi collettivi quanto più possibile
  - derived datatype per accessi non continui in memoria
- Leggere una volta nella vita lo Standard MPI-2.x o MPI-3.x
  - oppure appoggiarsi a librerie come HDF5 o NetCDF

33 CINECA

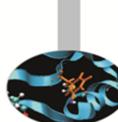
Cosa Abbiamo Lasciato Fuori



- non-blocking IO
  - consente di sovrapporre IO ad altre operazioni
- shared file pointer
  - utile per accodare dati sullo stesso file da più processi
  - quando non importa l'ordine di accesso
- passing hints to the implementation
  - alcune implementazioni beneficiano di queste informazioni
- consistency
  - quando il processo A vede i dati scritti dal processo B ?
- file interoperability and portability
  - la portabilità su varie architetture dei dati generati

34

Approfondimenti



- MPI – The Complete Reference vol.2, The MPI Extensions
  (W.Gropp, E.Lusk et al. - 1998 MIT Press)
- Using MPI-2: Advanced Features of the Message-Passing Interface
  (W.Gropp, E.Lusk, R.Thakur - 1999 MIT Press)
- Standard MPI-2.x (oppure l'ultima versione MPI-3.x)
  (http://www.mpi-forum.org/docs)
- Users Guide for ROMIO (Thakur, Ross, Lusk, Gropp, Latham)
- ... un po' di pubblicità:
  corsi@cineca.it (http://www.hpc.cineca.it)
- ... la pratica è la via della sapienza

35