Tecniche di programmazione in chimica computazionale

Optimization & parallelization

Emanuele Coccia

Dipartimento di Scienze Chimiche e Farmaceutiche

Optimization

- Make the program faster and more efficient

Optimization

- Make the program faster and more efficient
- User changes of the source code(s)

Optimization

- Make the program faster and more efficient
- User changes of the source code(s)
- Compiler options to optimize the program

Optimization

- Make the program faster and more efficient
- User changes of the source code(s)
- Compiler options to optimize the program
- Check the performances (profiling, see next slides)

Unformatted files

- Unformatted: binary format not readable by the user

Unformatted files

- Unformatted: binary format not readable by the user
- Advantage: faster and smaller

Unformatted files

- Unformatted: binary format not readable by the user
- Advantage: faster and smaller
- Disadvantage: need to be read on the same architecture

Unformatted files

- Unformatted: binary format not readable by the user
- Advantage: faster and smaller
- Disadvantage: need to be read on the same architecture
- Examples form1.f90 and form2.f90

Unrolling loops

- Improve performance with do loops

Unrolling loops

- Improve performance with do loops
- Explicitly unroll iterations within a loop

Unrolling loops

- Improve performance with do loops
- Explicitly unroll iterations within a loop
- User source-code optimization

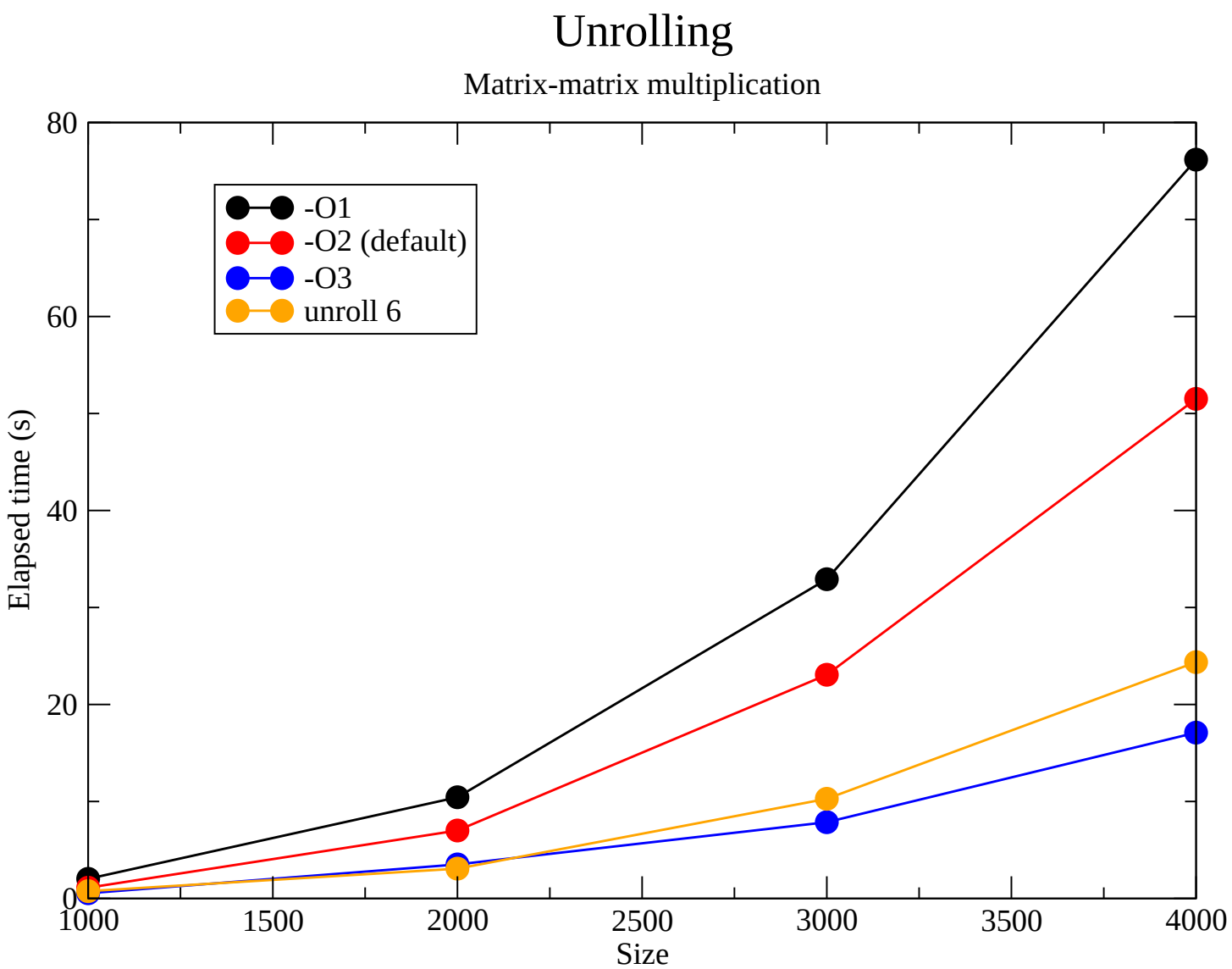
Unrolling loops

- Improve performance with do loops
- Explicitly unroll iterations within a loop
- User source-code optimization
- Example unrolling.f90 (matrix-matrix multiplication)

Unrolling loops

- Improve performance with do loops
- Explicitly unroll iterations within a loop
- User source-code optimization
- Example unrolling.f90 (matrix-matrix multiplication)
- Use optimization options of the compiler: -On (n=0,1, 2, 3) (type *man ifort* and search "Specifies the code optimization for applications.")
- Automatic optimization improving code performance

Unrolling loops

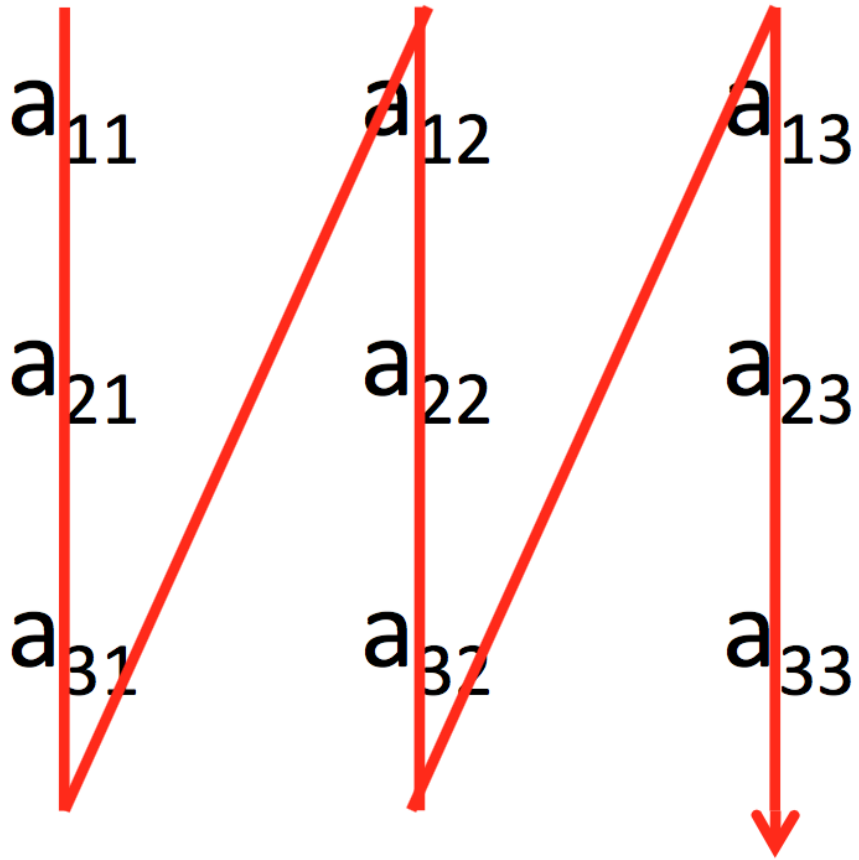


Swap indexes in matrices

- Methods to linearly store multidimensional arrays in RAM
- Fortran memory management: column-major order

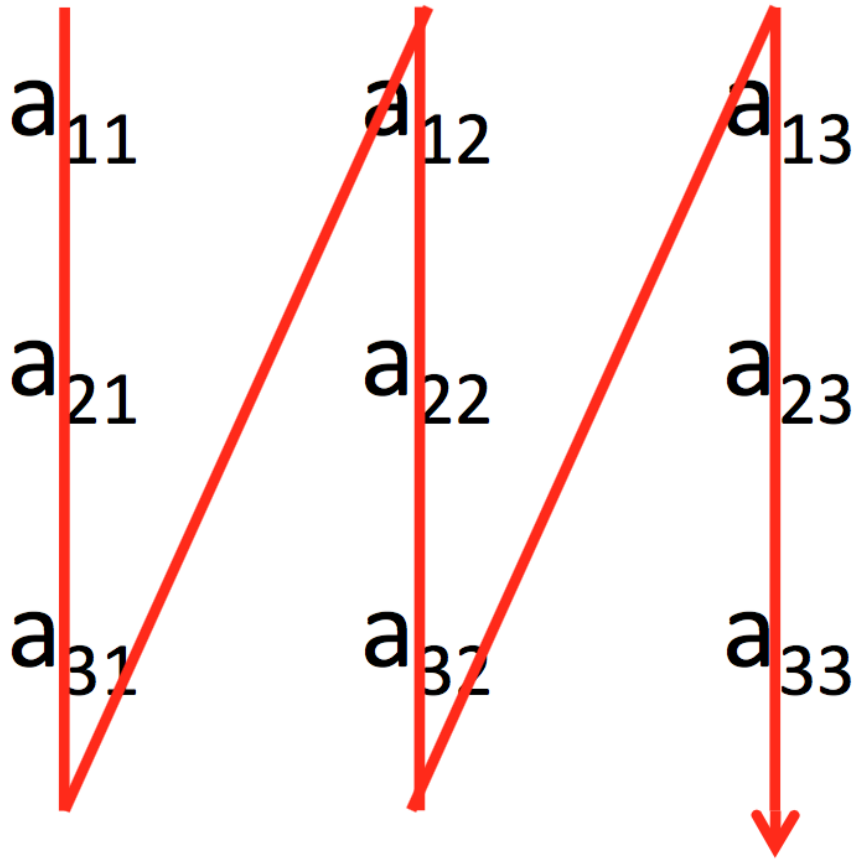
Swap indexes in matrices

- Methods to linearly store multidimensional arrays in RAM
- Fortran memory management: column-major order
- User source-code optimization



Swap indexes in matrices

- Methods to linearly store multidimensional arrays in RAM
- Fortran memory management: column-major order
- User source-code optimization



- Example matrix_swap.f90 (compile using -O0)

Profiling

- Detailed analysis of the code performance

Profiling

- Detailed analysis of the code performance
- Done using a compiler option:
  1. ifort -pg -o code.x code.f90

Profiling

- Detailed analysis of the code performance
- Done using a compiler option:
  1. ifort -pg -o code.x code.f90
  2. ./code.x

Profiling

- Detailed analysis of the code performance
- Done using a compiler option:
  1. ifort -pg -o code.x code.f90
  2. ./code.x
  3. gprof code.x > profile

Profiling

- Detailed analysis of the code performance
- Done using a compiler option:
  1. ifort -pg -o code.x code.f90
  2. ./code.x
  3. gprof code.x > profile
- Example profiling.f90

MPI parallelization

- MPI: Message Passing Interface protocol

MPI parallelization

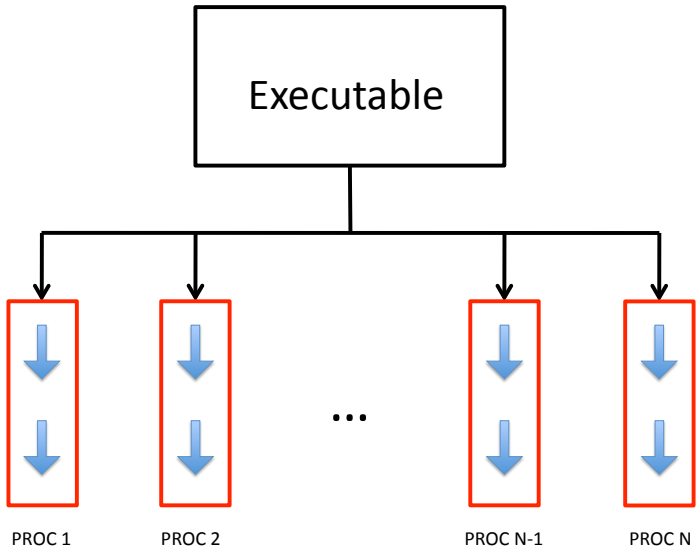
- MPI: Message Passing Interface protocol
- Goal: reduce the time spent for computation

MPI parallelization

- MPI: Message Passing Interface protocol
- Goal: reduce the time spent for computation
- Ideally, the parallel program is p times faster than the serial one (p being the number of processes)

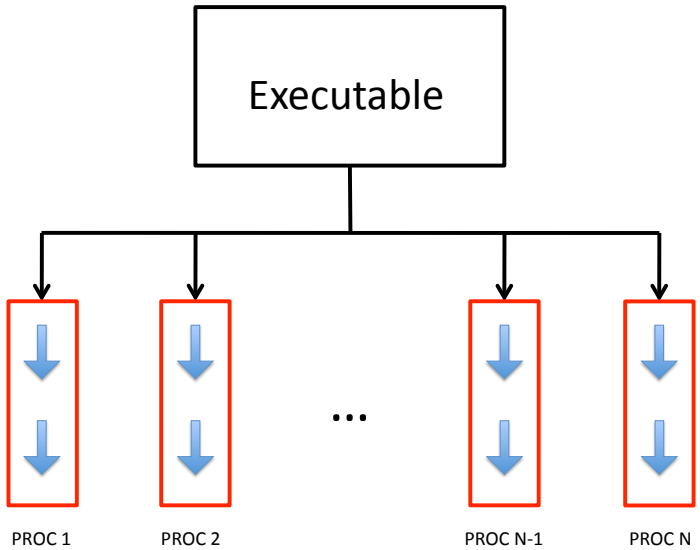
MPI parallelization

- MPI: Message Passing Interface protocol
- Goal: reduce the time spent for computation
- Ideally, the parallel program is p times faster than the serial one (p being the number of processes)
- SIMD: single instruction (executable), multiple data



MPI parallelization

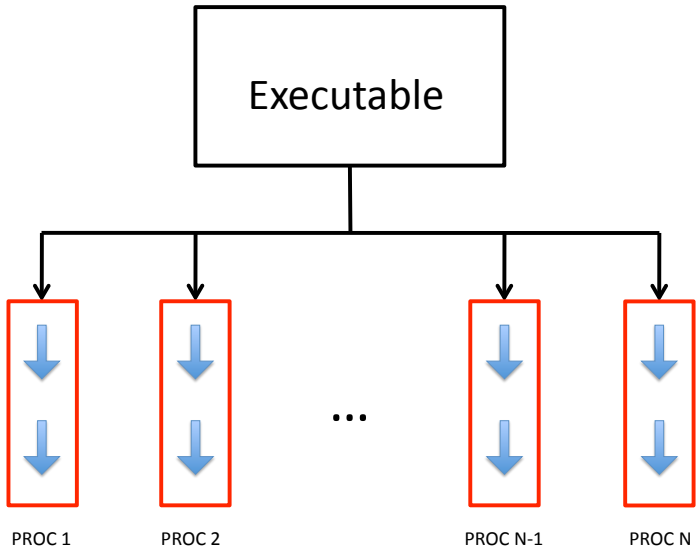
- MPI: Message Passing Interface protocol
- Goal: reduce the time spent for computation
- Ideally, the parallel program is p times faster than the serial one (p being the number of processes)
- SIMD: single instruction (executable), multiple data



- Each process has a unique identifier (rank)

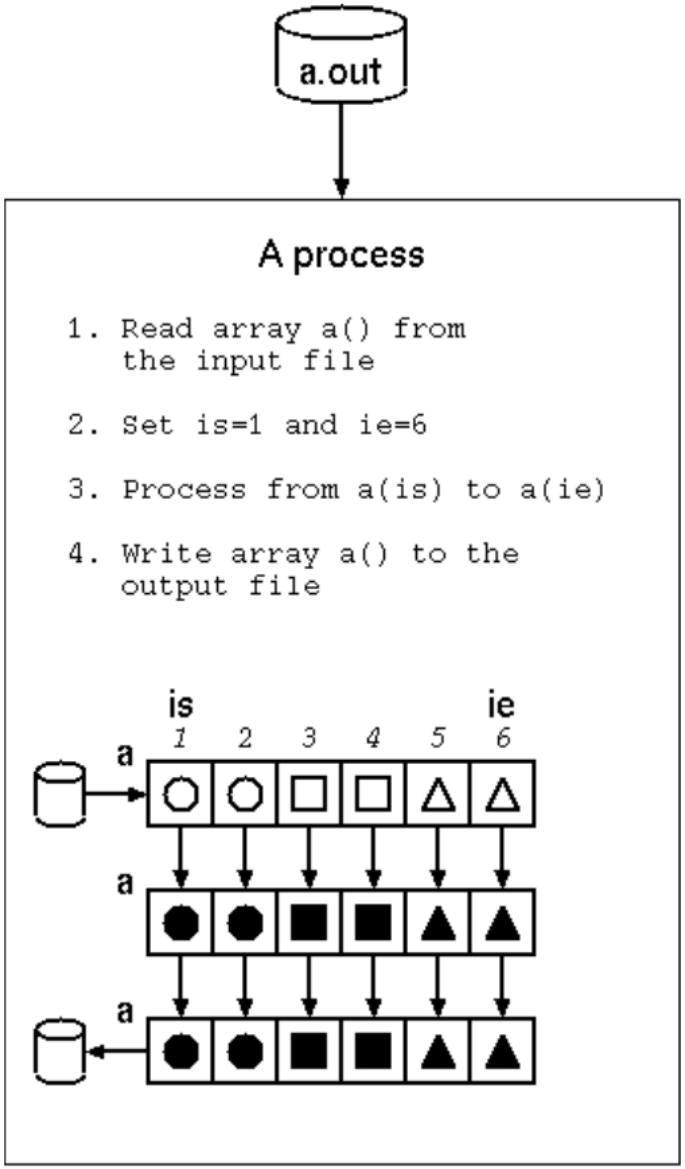
MPI parallelization

- MPI: Message Passing Interface protocol
- Goal: reduce the time spent for computation
- Ideally, the parallel program is p times faster than the serial one (p being the number of processes)
- SIMD: single instruction (executable), multiple data

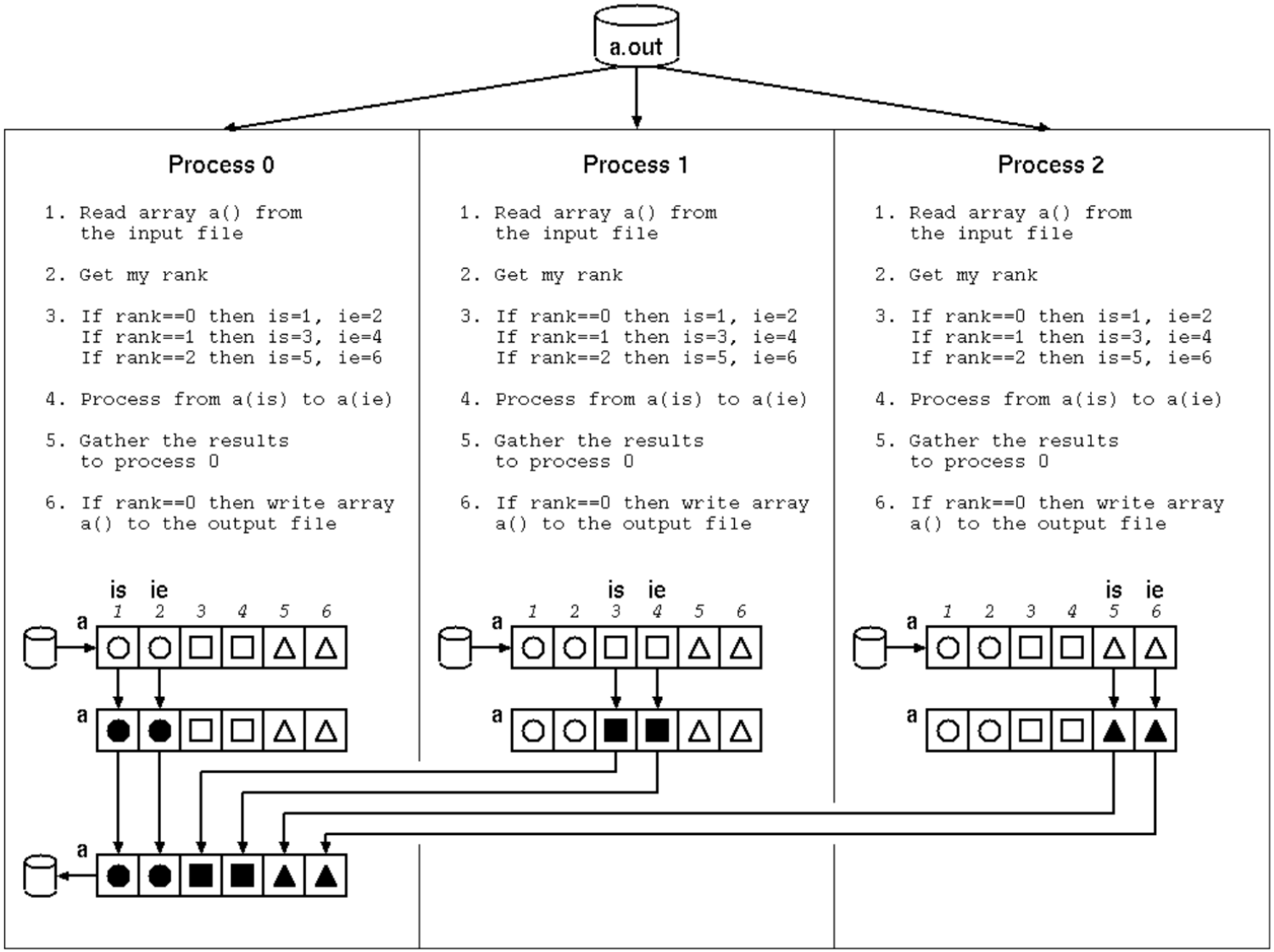


- Each process has a unique identifier (rank)
- Where: RS/6000: Practical MPI Programming

MPI parallelization



MPI parallelization



MPI parallelization

- Mandatory module mpi

MPI parallelization

- Mandatory module mpi
- Compile with mpi compiler, run using *mpirun -n ./code.x*

MPI parallelization

- Mandatory module mpi
- Compile with mpi compiler, run using *mpirun -n ./code.x*
- MPI suborutines: collective and point-to-point communication, environment management, communicators etc.

MPI parallelization

- Mandatory module mpi
- Compile with mpi compiler, run using *mpirun -n ./code.x*
- MPI suborutines: collective and point-to-point communication, environment management, communicators etc.
- Example: hello_world.f90

MPI parallelization

call mpi_bcast(buffer,count,datatype,root,comm,ierror)

MPI parallelization

call mpi_bcast(buffer,count,datatype,root,comm,ierror)

- buffer: starting address of the buffer (variable name)

MPI parallelization

call mpi_bcast(buffer,count,datatype,root,comm,ierror)

- buffer: starting address of the buffer (variable name)
- count: number of elements of the buffer

MPI parallelization

call mpi_bcast(buffer,count,datatype,root,comm,ierror)

- buffer: starting address of the buffer (variable name)
- count: number of elements of the buffer
- datatype: data type of buffer elements (MPI_INTEGER, MPI_DOUBLE_PRECISION etc.)

MPI parallelization

call mpi_bcast(buffer,count,datatype,root,comm,ierror)

- buffer: starting address of the buffer (variable name)
- count: number of elements of the buffer
- datatype: data type of buffer elements (MPI_INTEGER, MPI_DOUBLE_PRECISION etc.)
- root: rank of the root process

MPI parallelization

call mpi_bcast(buffer,count,datatype,root,comm,ierror)

- buffer: starting address of the buffer (variable name)
- count: number of elements of the buffer
- datatype: data type of buffer elements (MPI_INTEGER, MPI_DOUBLE_PRECISION etc.)
- root: rank of the root process
- comm: communicator (MPI_COMM_WORLD)

MPI parallelization

call mpi_bcast(buffer,count,datatype,root,comm,ierror)

- buffer: starting address of the buffer (variable name)
- count: number of elements of the buffer
- datatype: data type of buffer elements (MPI_INTEGER, MPI_DOUBLE_PRECISION etc.)
- root: rank of the root process
- comm: communicator (MPI_COMM_WORLD)
- ierror: Fortran return code

MPI parallelization

call mpi_bcast(buffer,count,datatype,root,comm,ierror)

- buffer: starting address of the buffer (variable name)
- count: number of elements of the buffer
- datatype: data type of buffer elements (MPI_INTEGER, MPI_DOUBLE_PRECISION etc.)
- root: rank of the root process
- comm: communicator (MPI_COMM_WORLD)
- ierror: Fortran return code
- Example bcast.f90