Tecniche di programmazione in chimica computazionale

Optimization and 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
- Compiler options to optimize the program

Optimization

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

Optimization

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

Optimization options

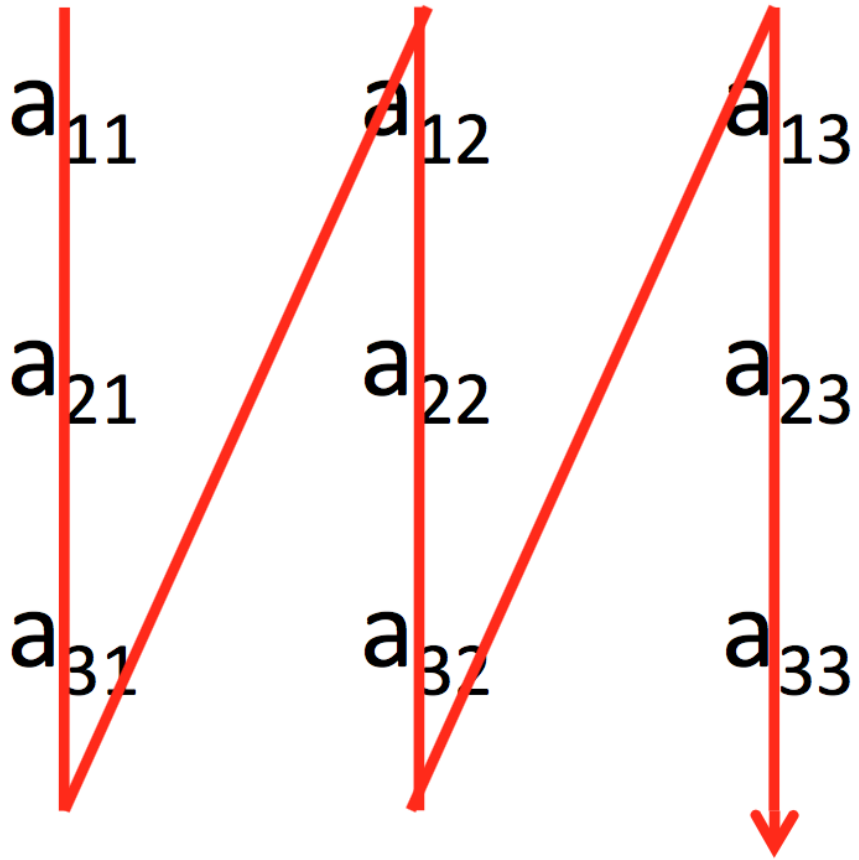
- Use optimization options of the compiler: -Ok (k=0,1, 2, 3) (type *man ifort* and search "Specifies the code optimization for applications.")
- Automatic optimization improving code performance
- Example with profiling.f90
- Compile with -O0, -O1, -O2 and -O3 (n=100 and m=1000 as input)
- Run the executable by typing time ./profiling.x

Swap indexes in matrices

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

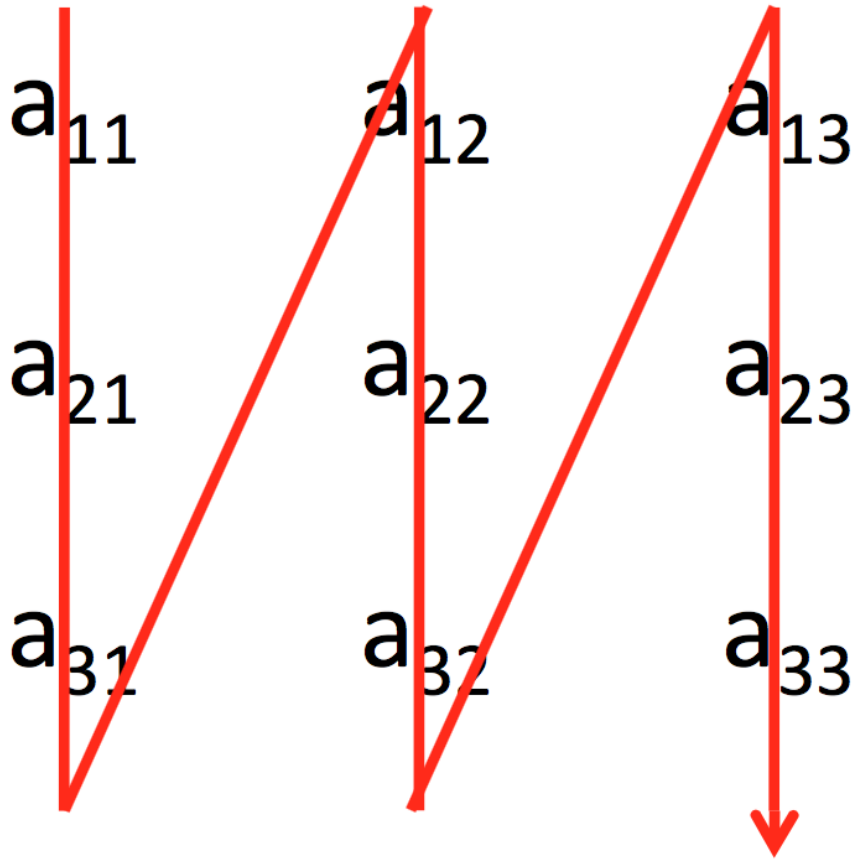
Swap indexes in matrices

- 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

Parallelization: where

- RS/6000: Practical MPI Programming

Parallelization: where

- RS/6000: Practical MPI Programming
- OpenMP Application Program Interface
- https://www.openmp.org/resources/tutorials-articles/

MPI parallelization

- MPI: Message Passing Interface protocol

MPI parallelization

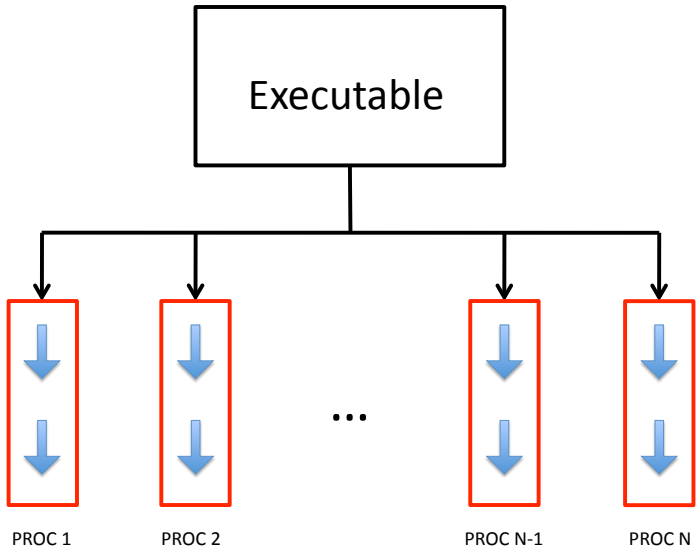
- MPI: Message Passing Interface protocol
- Goal: reduce the time spent for computation

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)

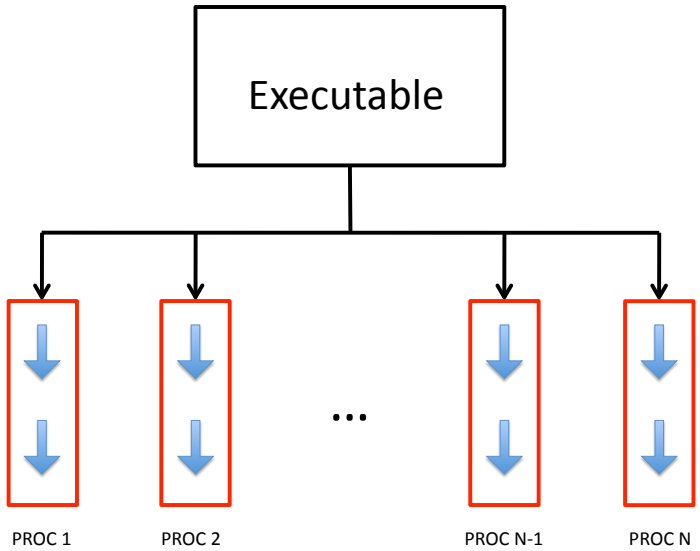
MPI parallelization

- 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



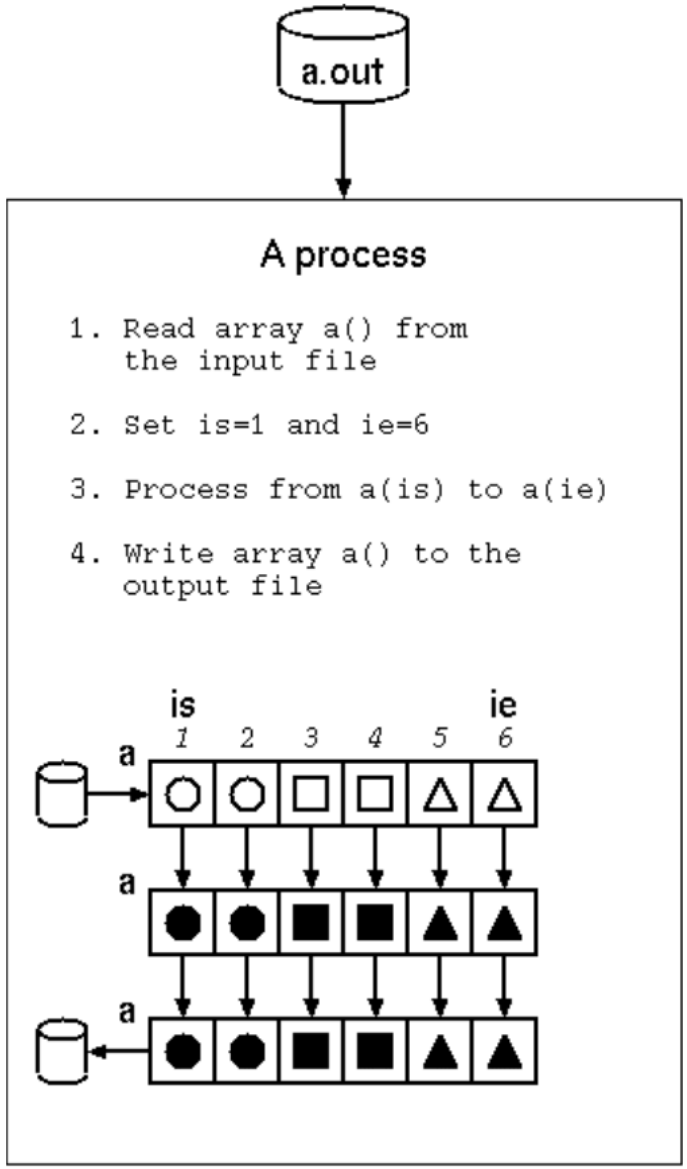
MPI parallelization

- 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

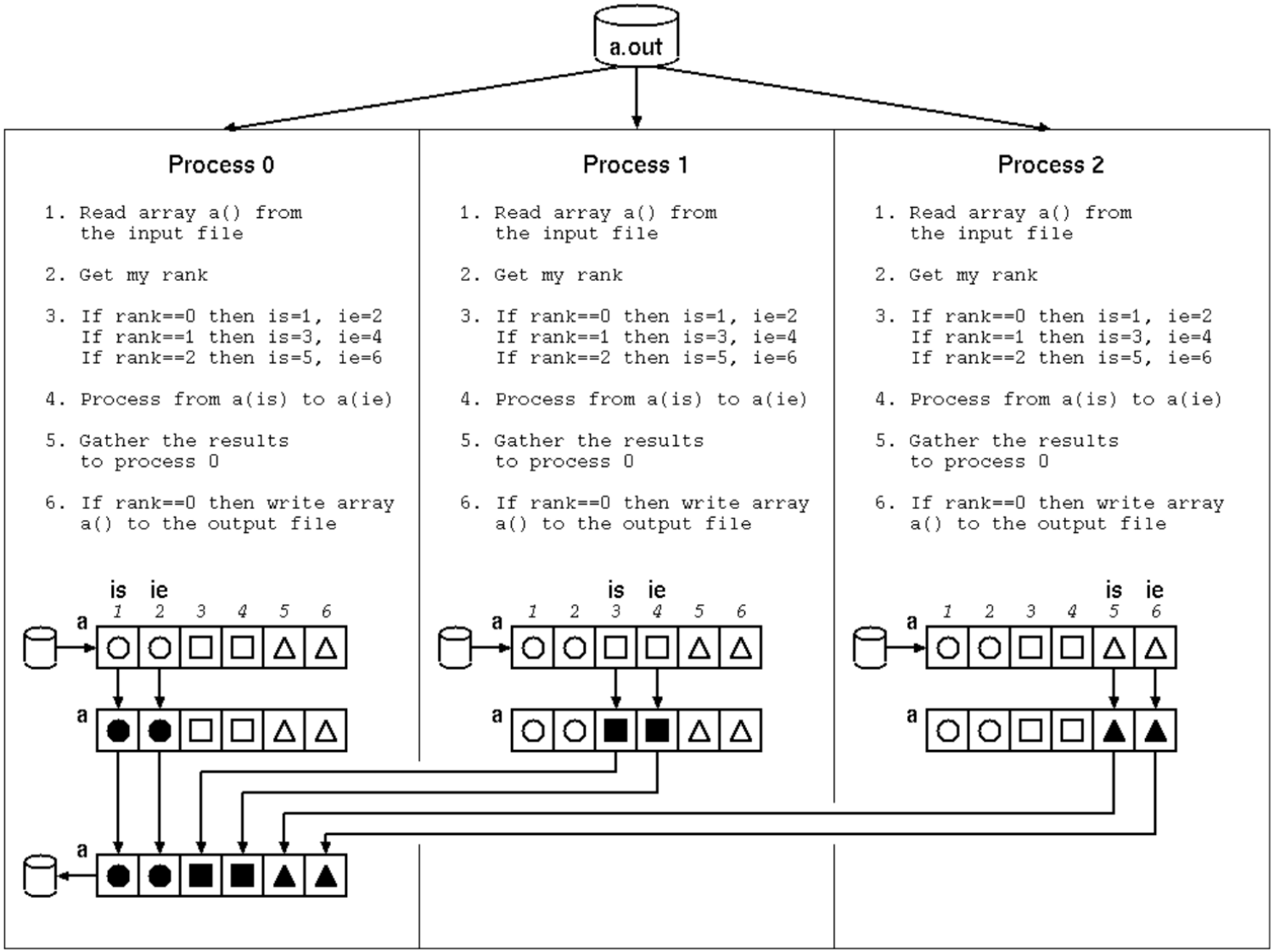


- Each process has a unique identifier (rank)

MPI parallelization



MPI 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: simd.f90