

# Tecniche di programmazione in chimica computazionale

Optimization & parallelization

Emanuele Coccia

Dipartimento di Scienze Chimiche e Farmaceutiche

- Make the program **faster** and **more efficient**

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

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

- 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: binary format **not readable** by the user

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

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



- 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**

- Improve performance with do loops

# Unrolling loops

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

# Unrolling loops

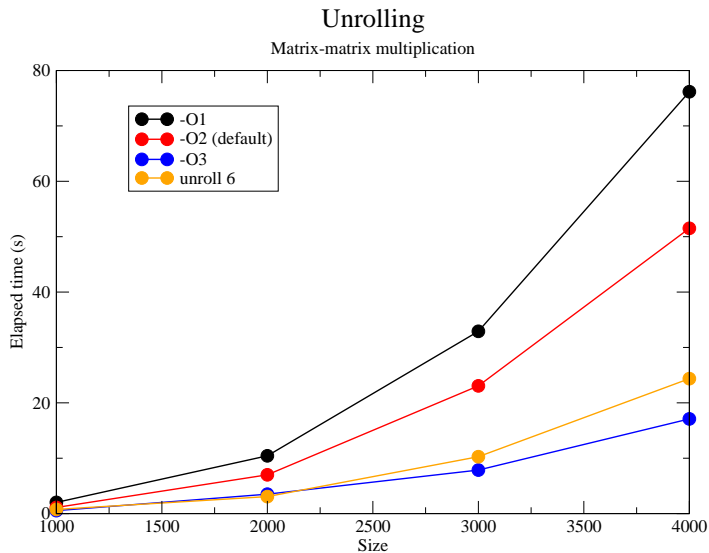
- Improve performance with do loops
- **Explicitly unroll** iterations within a loop
- User source-code optimization

# 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



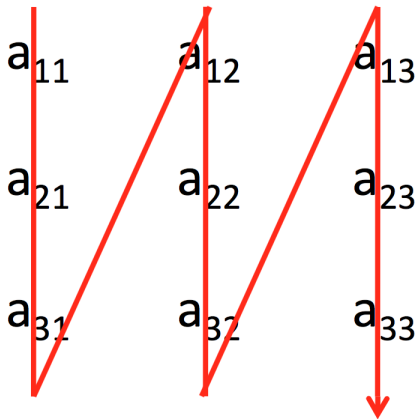
# Swap indexes in matrices

- Methods to linearly store multidimensional arrays in RAM
- Fortran memory management: [column-major order](#)



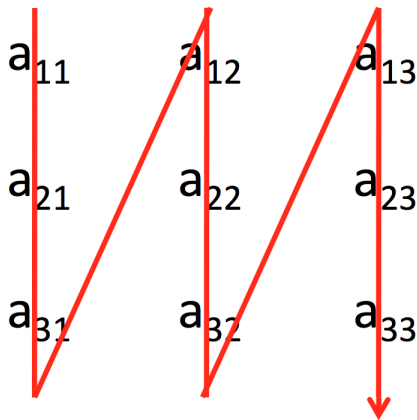
# Swap indexes in matrices

- 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`)

- Detailed analysis of the code performance

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

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

- 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`

- 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**

- RS/6000: Practical MPI Programming



- 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

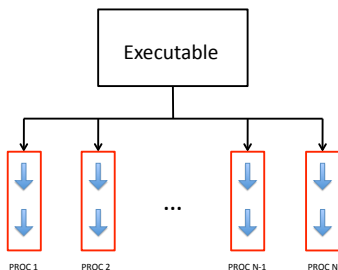
- **MPI**: Message Passing Interface protocol
- **Goal**: reduce the time spent for computation

# 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)

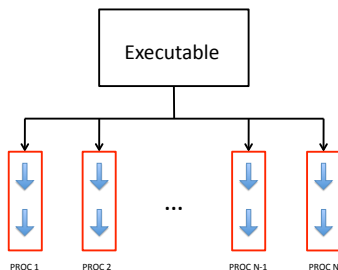
# MPI parallelization

- **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

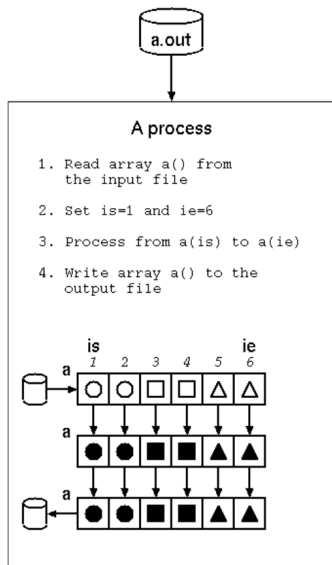


# MPI parallelization

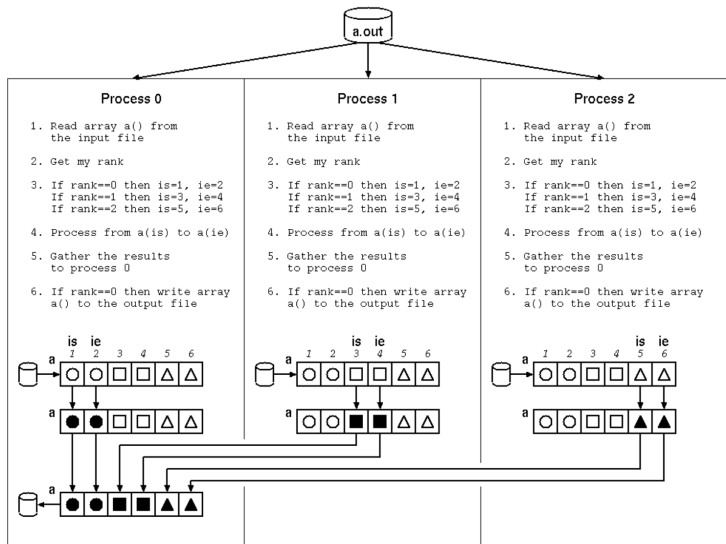
- **MPI**: Message Passing Interface protocol
- **Goal**: reduce the time spent for computation
- Ideally, the parallel program is  $p$  times faster than the serial one ( $p$  being the number of processes)
- **SIMD**: single instruction (executable), multiple data



- Each process has a **unique** identifier (**rank**)



# MPI parallelization





- Mandatory module `mpi`

# MPI parallelization

- Mandatory module `mpi`
- Compile with mpi compiler, run using `mpirun -N ./code.x`

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

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

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

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

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

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

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

*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.)



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

*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)

*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

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

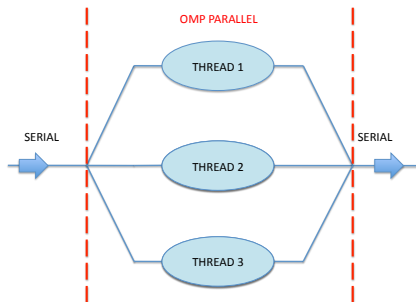
- **buffer**: starting address of the buffer (variable name)
- **count**: number of elements of the buffer
- **datatype**: data type of buffer elements (MPI\_INTEGER, MPI\_DOUBLE\_PRECISION etc.)
- **root**: rank of the root process
- **comm**: communicator (MPI\_COMM\_WORLD)
- **ierror**: Fortran return code
- Examples **bcast.f90** and **simd.f90**

# OpenMP parallelization

- Fork-join model of parallel execution

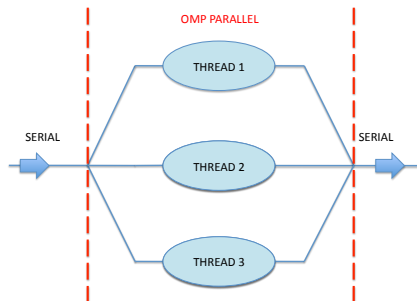
# OpenMP parallelization

- **Fork-join** model of parallel execution
- **Thread**: execution entity with a serial flow of control and an associate stack



# OpenMP parallelization

- **Fork-join** model of parallel execution
- **Thread**: execution entity with a serial flow of control and an associate stack



- **Shared-** (OpenMP) vs **distributed-**memory (MPI) model

# OpenMP parallelization

```
!$OMP PARALLEL
```

```
!$OMP DO
```

```
do i=1...
```

```
...
```

```
enddo
```

```
!$OMP END PARALLEL
```



# OpenMP parallelization

```
!$OMP PARALLEL
```

```
!$OMP DO
```

```
do i=1...
```

```
...
```

```
enddo
```

```
!$OMP END PARALLEL
```

- Shared (accessed by all the threads) and private (each thread has its own copy) variables

# OpenMP parallelization

```
!$OMP PARALLEL
```

```
!$OMP DO
```

```
do i=1...
```

```
...
```

```
enddo
```

```
!$OMP END PARALLEL
```

- Shared (accessed by all the threads) and private (each thread has its own copy) variables
- *REDUCTION(+:name\_variable)* to sum into *name\_variable*

# OpenMP parallelization

```
!$OMP PARALLEL
```

```
!$OMP DO
```

```
do i=1...
```

```
...
```

```
enddo
```

```
!$OMP END PARALLEL
```

- Shared (accessed by all the threads) and private (each thread has its own copy) variables
- *REDUCTION*(+:*name\_variable*) to sum into *name\_variable*
- Use *-fopenmp* option (ifort) and module `omp_lib`

# OpenMP parallelization

```
!$OMP PARALLEL
```

```
!$OMP DO
```

```
do i=1...
```

```
...
```

```
enddo
```

```
!$OMP END PARALLEL
```

- Shared (accessed by all the threads) and private (each thread has its own copy) variables
- *REDUCTION*(+:*name\_variable*) to sum into *name\_variable*
- Use *-fopenmp* option (ifort) and module `omp_lib`
- Example `omp_matmul.f90`

# OpenMP parallelization

