Scalability of Population-based Metaheuristics

- Motivation
- Parallelization models
- Distributed models
- Cooperative coevolution

Motivation

The evolutionary algorithms need a large amount of resources:

- Memory space (since they usually need large populations)
- Execution time (since the evolutionary process is usually long)

Costly operations:

- The evaluation of the population elements
- The application of operators

Solutions:

- Improving the convergence rate of the algorithm (by developing new operators)
- Increasing the efficiency of the implemention (parallel/ distributed implementation)

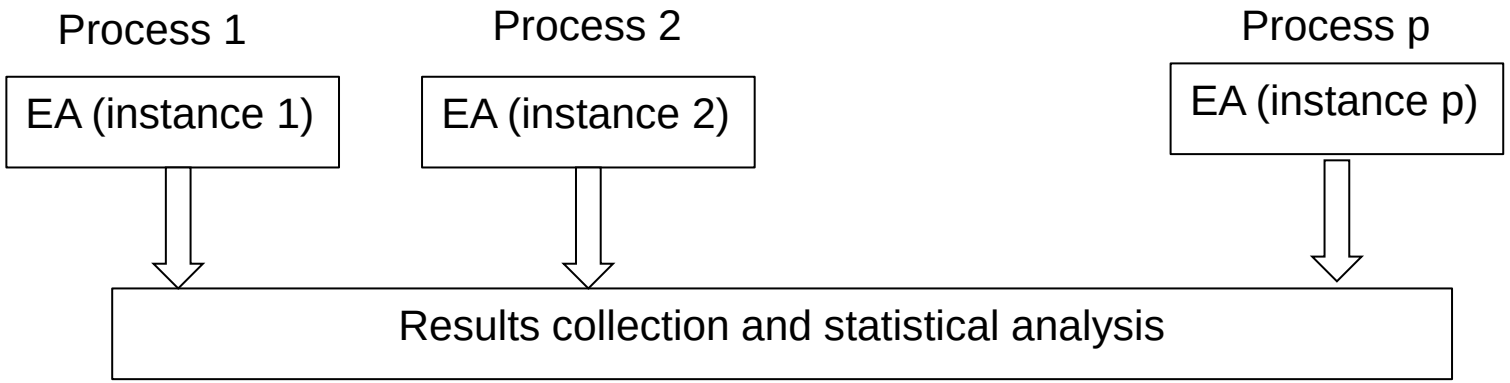
Parallel and distributed models

The parallelization can be implemented at different levels:

- Algorithm -> naive parallelization model
- Elements evaluation -> master-slave model
- Population -> island model
- Element -> cellular model

Naïve model

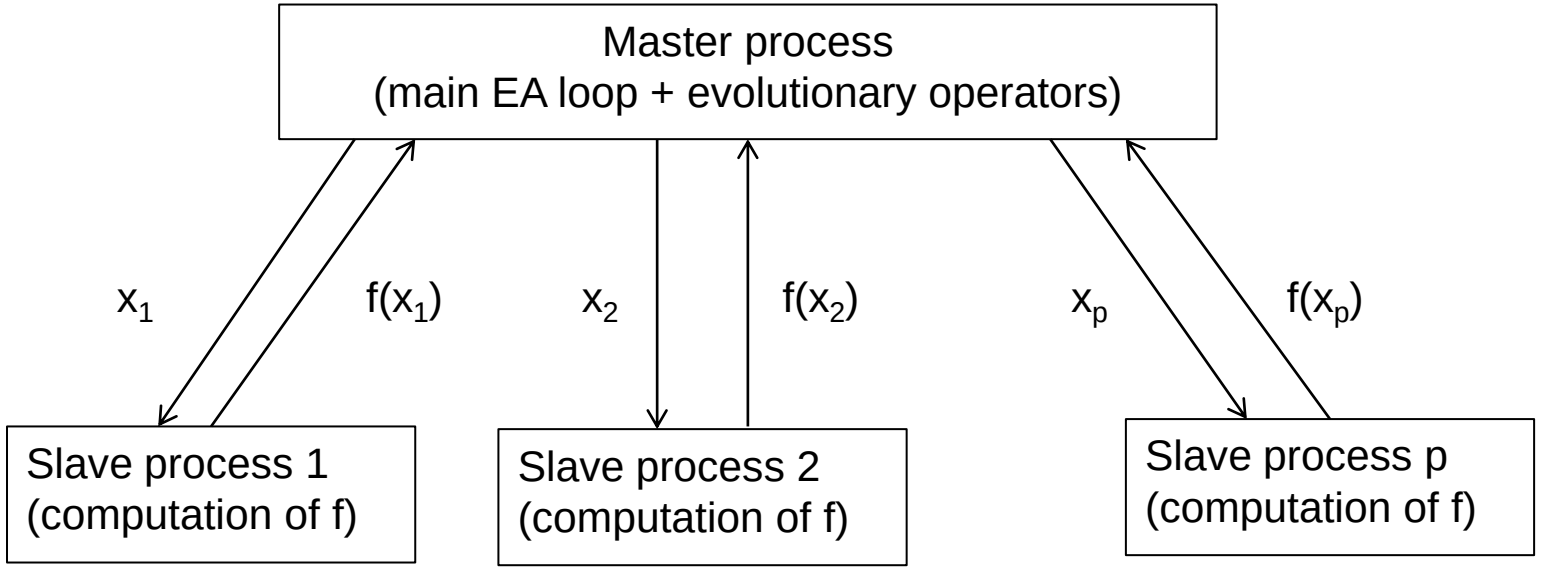
The algorithm is simultaneously executed on several processors which do not communicate



Is useful for statistical analysis or for parameter tuning

Master-slave model

The master process executes the EA and distributes the evaluation of the population elements to the slave processes



Master-slave model

Particularities:

- If the population size is larger than the number of available processors then the master process has to distribute the elements to processors.
- The evaluation time depends not only on the characteristics of the processor but also on the particularities of the element which should be evaluated (e.g. in genetic programming)
- In such a case there is necessary to synchronize the computations. In order to avoid frequent synchronization steps the generational (synchronous) strategy can be replaced with a an asynchronous (steady-state) strategy

Master-slave model

Sinchronous

Population initialization
Population evaluation
REPEAT
- Parents selection
- Generate a population of offspring
- Evaluate the offspring population
- Select the survivors

UNTIL <stopping condition>

Asynchronous

Population initialization
Population evaluation
REPEAT
- Parents selection
- Generate a new element
- Evaluate the new element
- Assimilate the new element in the population

UNTIL <stopping condition>

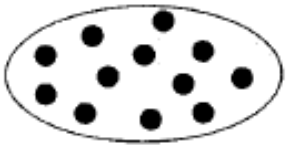
Master-slave model

- Is easy to implement
- Leads to a more efficient implementation only if the evaluation step is significantly more costly than the other operations involved in the EA.
- The behavior of the evolutionary algorithm (with respect to the converegence properties) is not changed
- It can be implemented both on systems with shared memory and on systems with distributed memory (including computer networks)

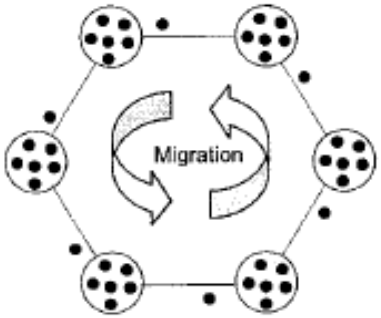
Structuring the population

- The population can be unstructured (panmictic) or structured
- Structuring the population has an influence on the evolutionary process, one of its effects being the stimulation of the population diversity.
- There are different models:
  - Coarse-grain model (island model)
  - Fine-grain model (cellular model)

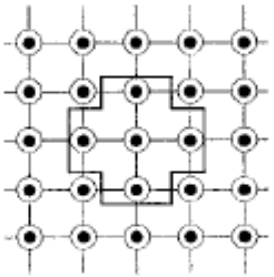
Alba, Tomassini; Parallelism and EAs, 2002



Model panmictic



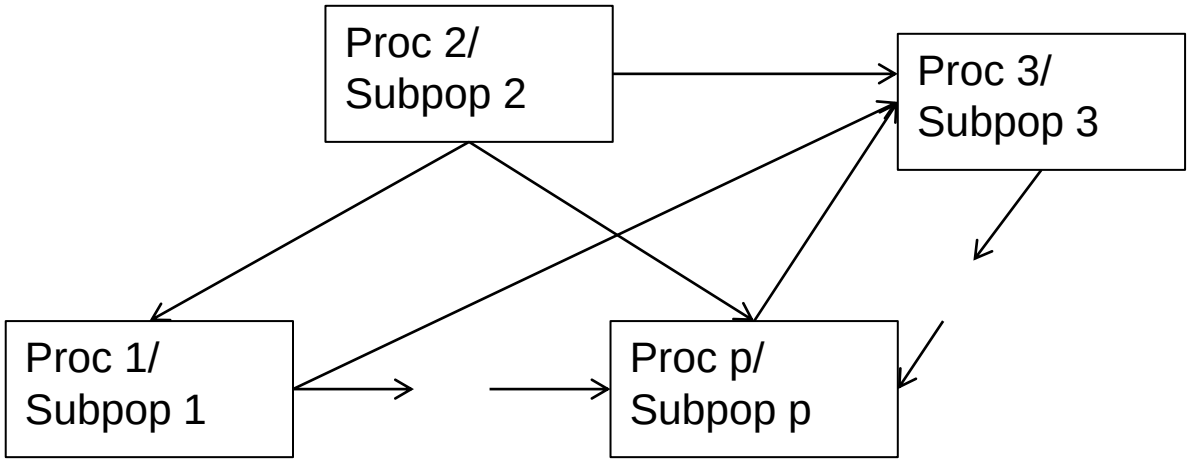
Island model



Cellular model

Island model

- Consists of dividing the population in subpopulations (“islands” or “demes”) on which there are executed identical or different EAs and which communicate between them by a so-called migration process.



- A processor can deal with one or several subpopulations
- In each subpopulation the evolutionary operators are applied for a given number of iterations then a migration process is initiated.

Island model

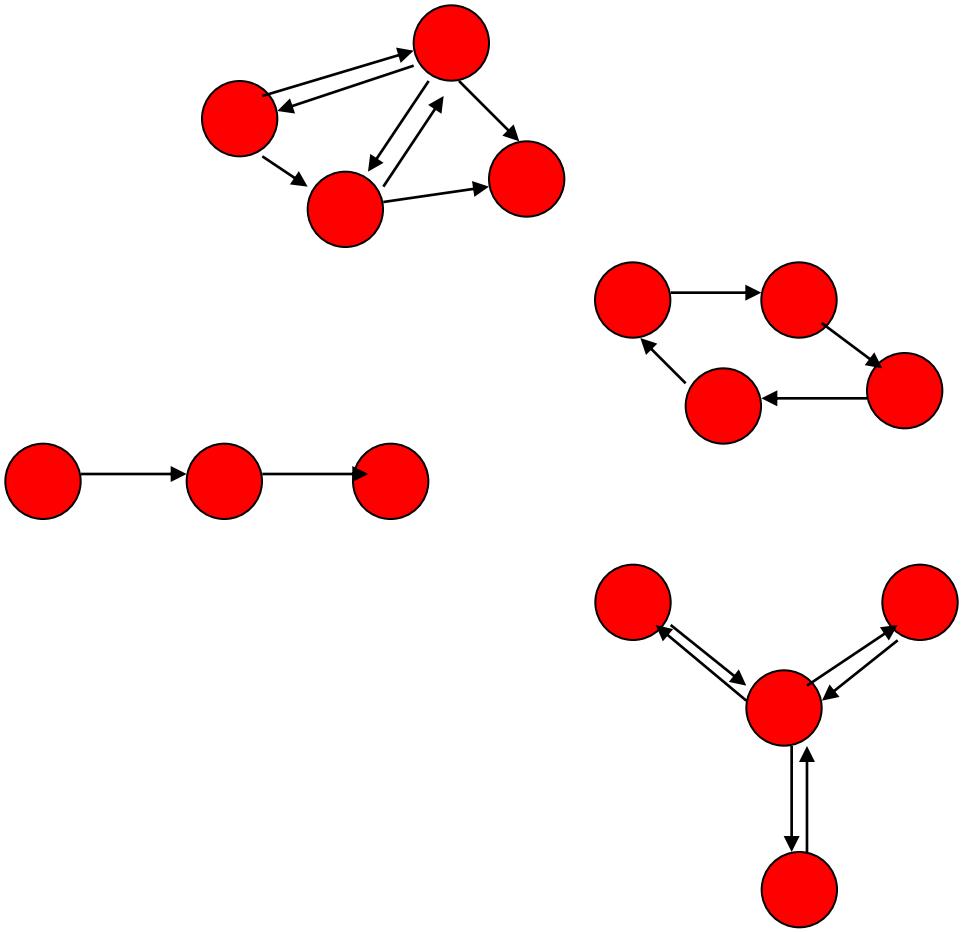
The communication process between subpopulations is characterized by:

- Communication topology
- Communication strategy
- Parameters controlling the communication

These elements have an important influence on the behaviour of the algorithm and on its efficiency.

Island model

- Communication topology
  - Random
  - Ring
  - Linear
  - Star



Island model

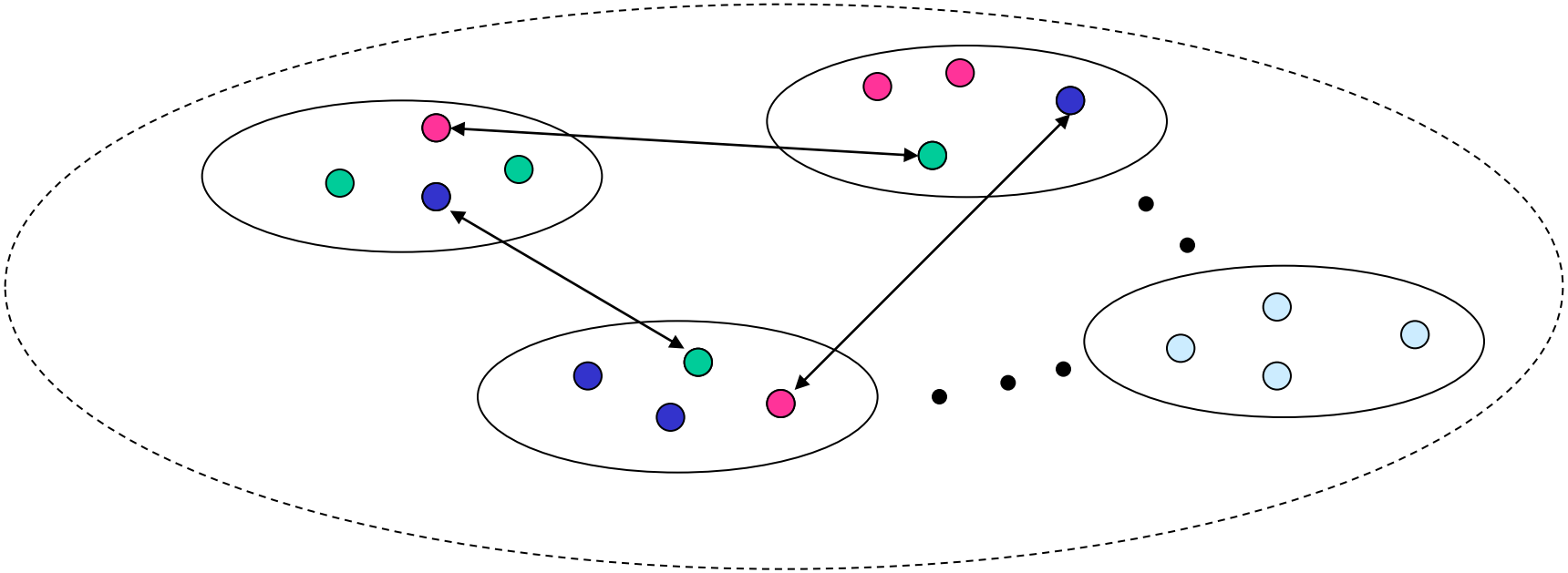
- Communication strategy
  - Migration: an element form the source subpopulation is exchanged with an element from the destination subpopulation
  - Pollination: a copy of an element from the source subpopulation is transferred in the target subpopulation

- Selection of the element in the source subpopulation
  - Random
  - Elitist (one of the best elements)

- Selection of the element in the destination subpopulation
  - Random
  - Elitist (one of the best elements in the case of migration; one of the worst elements in the case of pollynation)

Island model

- Example:
  - Elements exchange
  - The global distribution of the elements remains unchanged; only the distribution of elements in the subpopulations is changed

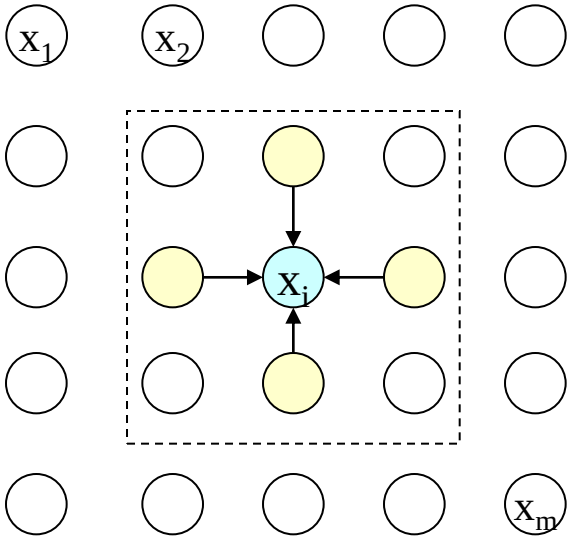


Island model

- Specific parameters:
  - Migration frequency:
    - Based on the number of generation
    - Based on the subpopulations properties
  - Migration probability:
    - A high value means a lot of communication between subpopulations

Cellular model

- The elements are placed in the nodes of a grid (characterized by a given topology)
- Only the neighbours are involved in the selection and crossover process
- In a parallel implementation each element is assigned to a processor (appropriate for implementations on supercomputers)



http://neo.lcc.uma.es/cEA-web/index.htm

Cellular model

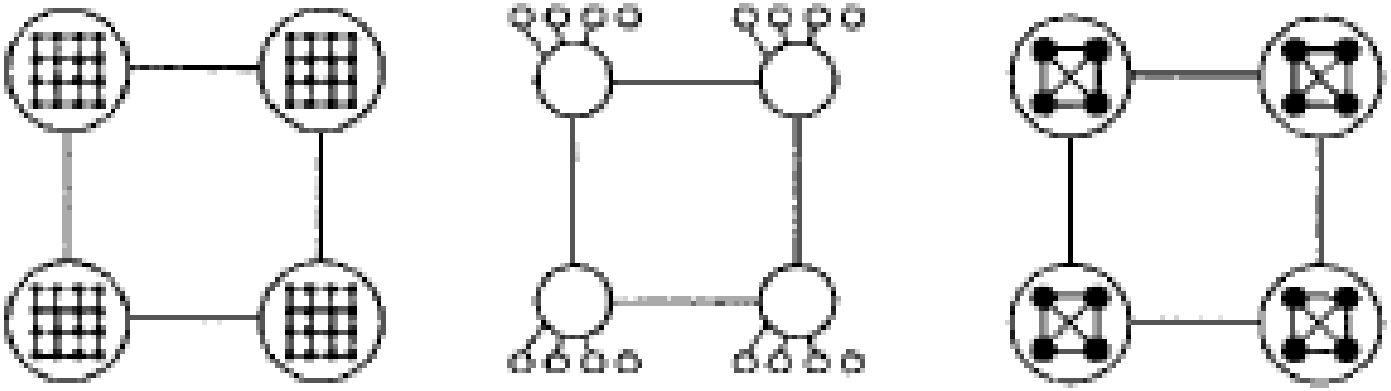
- Can be used also in the case of sequential implementations since it induces a different dynamics.
- Somehow similar to cellular automata
- There are two variants:
  - Synchronous: all offspring are computed in parallel and the replacement is done simultaneously
  - Asynchronous: the new elements replace their parents as soon as they are generated (asynchronously)

Cellular model

- Asynchronous variants:
  - Random selection of elements involved in the reproduction process
  - The cells in the grid are scanned systematically (e.g. row by row)
  - The elements are processed in the order given by a random permutation
- The asynchronous variant is usually quicker than the synchronous one

Hybrid variants

- The master/slave, island and cellular models cand be combined in one of the following variants:
  - Island+cellular
  - Island+MasterSlave
  - Island+island

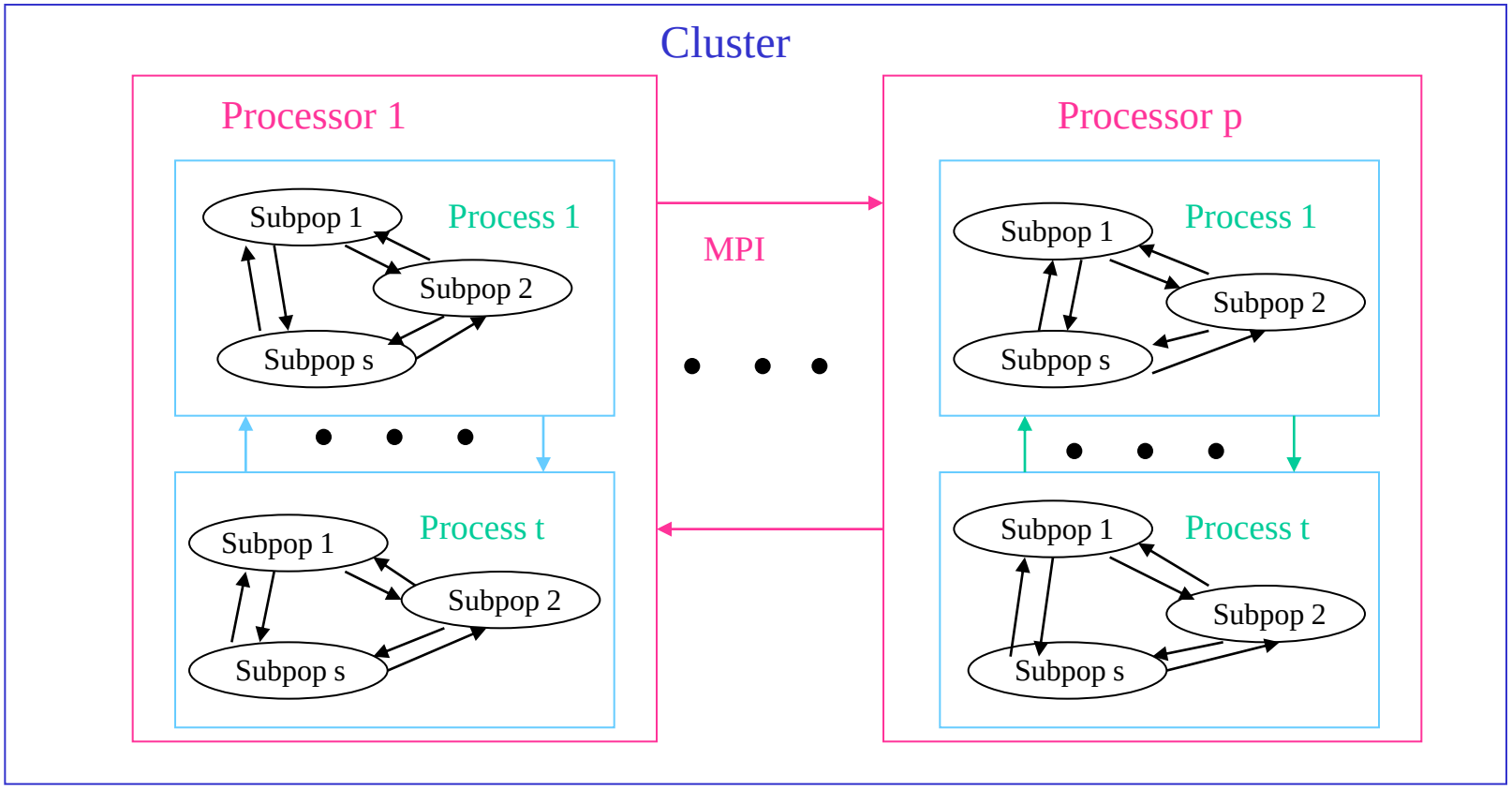


Implementation

- The appropriate computing environment depends on the model granularity and on the communication
  - Master-slave model: appropriate for cluster architectures
  - Island model: both for cluster and distributed architectures
  - Cellular model: multi-processors
- Software: tools PVM, MPI, OpenMP etc.

Implementation

- Example (for an island model implemented in a cluster environment)



Implementation

- Current trend: implementation on GPUs and hybrid CPU+GPU
- There are reported results corresponding to all models:
  - Master-slave (e.g. the EA is executed on CPU while elements evaluation is executed on GPU)
  - Fine-grained (cellular) – the whole EA is executed on GPU (rmk: there are also implementations which use CPU for generating random values); there are reported results of Cellular EAs using up to 10000 elements in the population
  - Coarse-grained (island model) – the population initialization and distribution in subpopulations is done on CPU, while the EA on each subpopulation is executed on GPU (the migration is realized by shuflling the subpopulations on GPU VRAM)
  - Hierarchical models
    - [Biblio: Arenas et al; GPU Parallel Computation in Bioinspired
    - Algorithms. A review.]

Cooperative Coevolution

- Appropriate in the case of high-dimensional problems
- Main idea: split the problem into smaller sub-problems
  - A potential solution consists of several components
  - Evolve independently the population corresponding to each component (coevolution)
  - Each component is evaluated in the context of other components

Cooperative Coevolution

Implementation issues:

- Choosing the components
  - How many components?
  - How to assign a variable to a component?
- Coevolution of components
  - How to construct the evaluation context for each component
  - How long should be the evolution of a component in the same context

Cooperative Coevolution

Simplest case:

- Assign each variable to one component = a problem of size n is decomposed in n one-dimensional problems

- This approach is appropriate in the case of separable problems (for instance if $f(x_1,x_2,...,x_n)=f_1(x_1)+f_2(x_2)+...+f_n(x_n)$)

but it does not work well in the case of nonseparable functions

(for instance $f(x_1,x_2)=100(x_1-x_2)^2+(1-x_1)^2$)

– in such a case the context used to evaluate a variable is important which means that the interacting variables should be evolved together

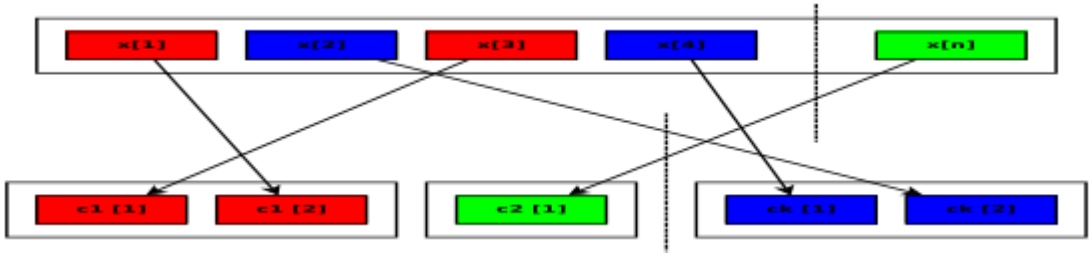
Cooperative Coevolution

Ideal case:

- Each component contains highly interacting variables while variables assigned to different components are only loosely coupled

Compromise variant:

- Assign variables to components in a random manner (at each iteration)
- It works when the interactions are limited to pairs of elements



Cooperative Coevolution

Differential grouping:

- Estimate for each variable the degree of interaction with other variables
- Two variables i and j are considered interacting if for an arbitrary vector x and two perturbations d_1 and d_2:

$$f(...,x_i+d_i,...,x_j+d_j,...)-f(...,x_i+d_i,...,x_j,...) \mathrel{!}= f(...,x_i,...,x_j+d_j,...)-f(...,x_i,...,x_j,...)$$

- The variables are grouped based on this interaction information

[Omidvar et al., Cooperative Coevolution with Differential Grouping for Large Scale Optimization, 2013]

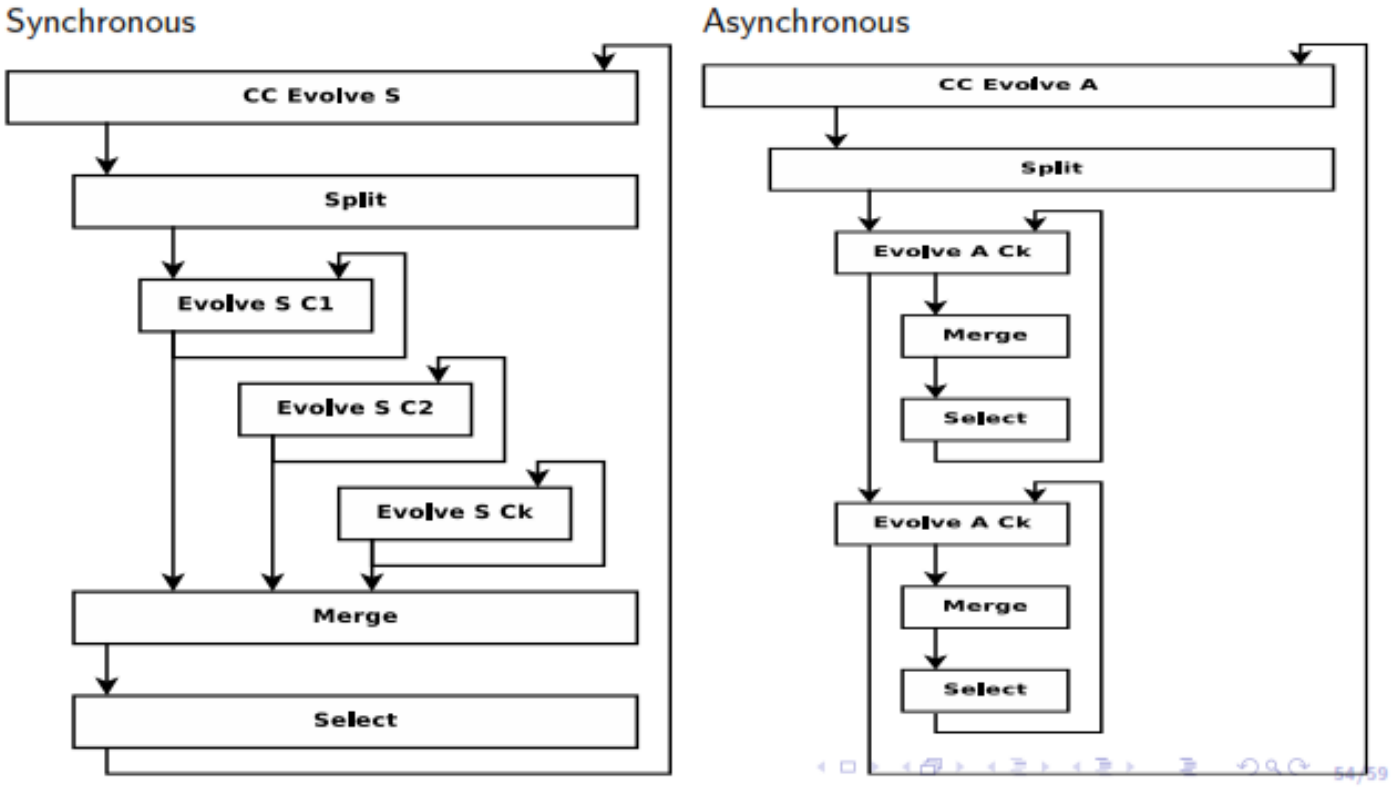
Cooperative Coevolution

Choosing the evaluation context:

- To evaluate a component ck one have to construct a virtual full solution, (c1,...,ck,....,cK) by defining an evaluation context consisting of collaborators ci provided by the subpopulation corresponding to each other component
- A collaborator can be:
  - Best element in the corresponding subpopulation
  - The corresponding component from the best element over the entire population
  - The corresponding component of a random element of the population

Cooperative Coevolution

Implementation variants:



Cooperative Coevolution

Impact on the number of function evaluations needed to attain a given accuracy:

Plot of:
nfe(n)/nfe(100)
versus n (the problem size)

Algorithm:
Differential Evolution (lecture 8)