MPS based on Genetic Algorithms implemented in a (Distributed)Shared-Memory Supercomputer

Oscar Peredo[1], Julián M. Ortiz[2]

1: Computer Applications in Science & Engineering Department, Barcelona Supercomputing Center

2: Advanced Lab for Geostatistical Supercomputing, Advanced Mining Technology Center

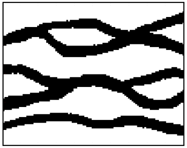
9th International Geostatistics Congress, June 11-15, 2012 - Oslo, Norway

Outline

Outline

Objectives of this talk

- We want to generate realizations that reproduce **multiple-point statistics** (MPS) inferred from a 2D training image.


Training image

Realization A

Realization B

Realization C

Realization D

- Several methods to do this task: snesim, unilateral path, neural networks, **simulated annealing**...
- **How good are genetic algorithms to reproduce MPS?**

Outline

GAs: overview

```
1:  Initial population: $N$ random individuals
2:  $k \leftarrow 0$
3:  Evaluate a **fitness function** $f$ in each individual
4:  **while** termination criteria is not achieved **do**
5:      $\{$generation $k\}$
6:      **Selection**: select best individuals using $f$ values
7:      **Crossover**: breed new individuals crossing bits
8:      **Mutation**: breed new individuals mutating some bits
9:      Replace old individuals by new ones
10:     $k \leftarrow k + 1$
11:     Evaluate a fitness function $f$ in each individual
12: **end while**
```

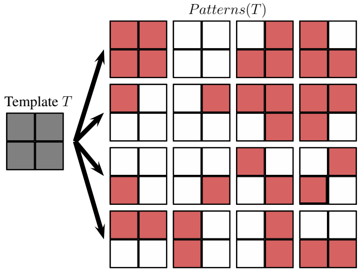
GAs: overview

```
1:  Initial population: $N$ random individuals
2:  $k \leftarrow 0$
3:  Evaluate a **fitness function** $f$ in each individual
4:  **while** termination criteria is not achieved **do**
5:      $\{$generation $k\}$
6:      **Selection**: select best individuals using $f$ values
7:      **Crossover**: breed new individuals crossing bits
8:      **Mutation**: breed new individuals mutating some bits
9:      Replace old individuals by new ones
10:     $k \leftarrow k + 1$
11:     Evaluate a fitness function $f$ in each individual
12: **end while**
```

Individuals 1D (nm) $\Longleftrightarrow$ Realization images 2D ($n \times m$)

GAs: fitness function

- It depends on a user-defined template T:



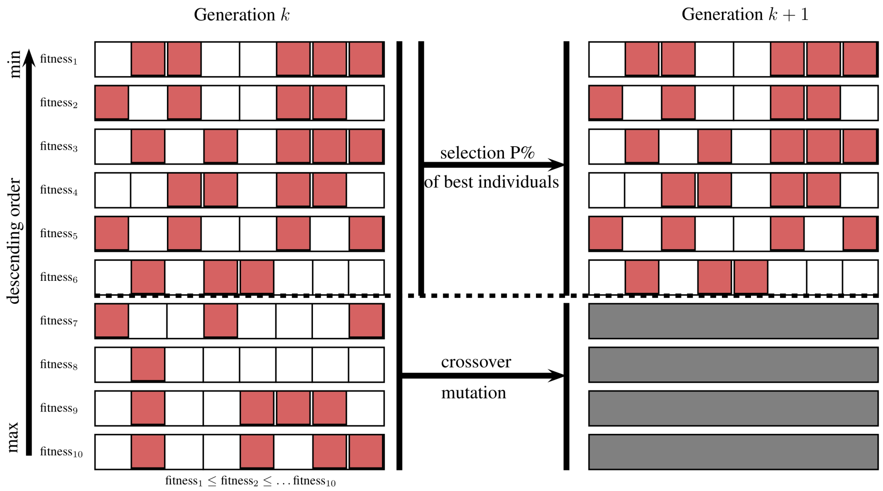
- Using $Patterns(T)$, we calculate:

$$fitness(indiv_k) = \sum_{p \in Patterns(T)} \left(\#pattern_p^{TI} - \#pattern_p^{indiv_k} \right)^2$$

- This function measures the mismatch between the target statistics (from the training image) and the current statistics for each individual.

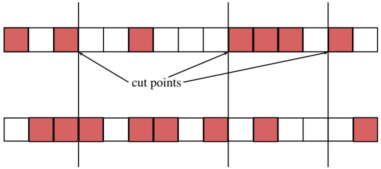
GAs: selection

P% of the population ”survives”. The rest is generated using crossover/mutation.

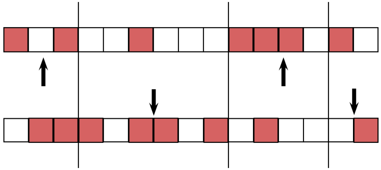


GAs: crossover

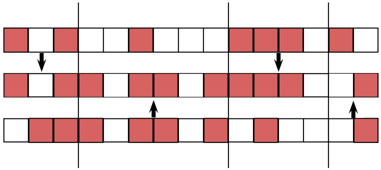
- Choose cut points



- Choose sections

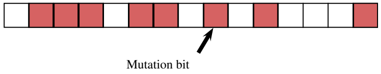


- Breed new individual with the chosen sections

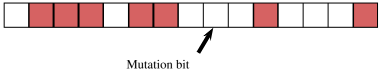


GAs: mutation

- Choose mutation bit

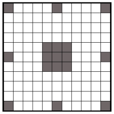


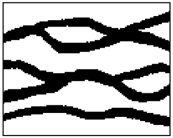
- Breed new individual changing its value with probability $m := m(t)$

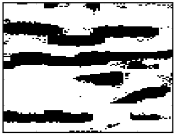


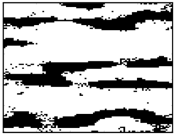
GAs: examples, (non)-conditional realizations

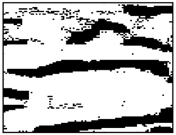
100$\times$100 images, 10000 generations, 4000 individuals,
20% selection, dynamic mutation probability, 100 cut-points
Template 17 nodes disconnected

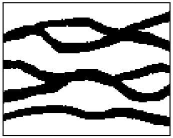



Training image

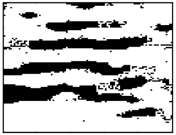

No conditionants

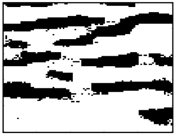

No conditionants


No conditionants


Training image

5% conditionants

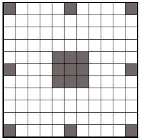

5% conditionants

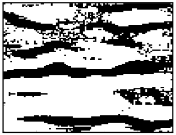

5% conditionants

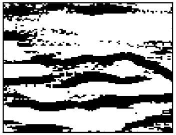
GAs: examples, dynamic template utilization

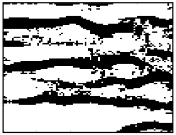
100×100 images, 10000 generations, 4000 individuals, 20% selection, dynamic mutation probability, 100 cut-points, no conditionants

Template: first half with template T_1, second half with template T_2

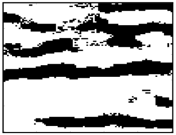

T_1: template 17 nodes

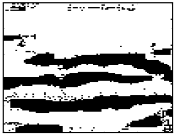

Realization gen. 5000

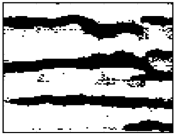

Realization gen. 5000


Realization gen. 5000

T_2: template 9 nodes


Realization gen. 10000


Realization gen. 10000


Realization gen. 10000

GAs: remarks

- Advantages:
    - Reasonable good realizations (the quality depends on the number of template's nodes).
    - straightforward and easy implementation (Fortran90)
    - the initial population can be enhanced with external information from different models to improve or refine the results.
- Disadvantages:
    - adjustment of several parameters (mutation probability, crossover cut points, population size,. . .)
    - time and memory expensive

GAs: remarks

- Advantages:
  - Reasonable good realizations (the quality depends on the number of template's nodes).
  - straightforward and easy implementation (Fortran90)
  - the initial population can be enhanced with external information from different models to improve or refine the results.
- Disadvantages:
  - adjustment of several parameters (mutation probability, crossover cut points, population size,. . .)
  - time and memory expensive $\Rightarrow$ ***can we use parallel computing?***

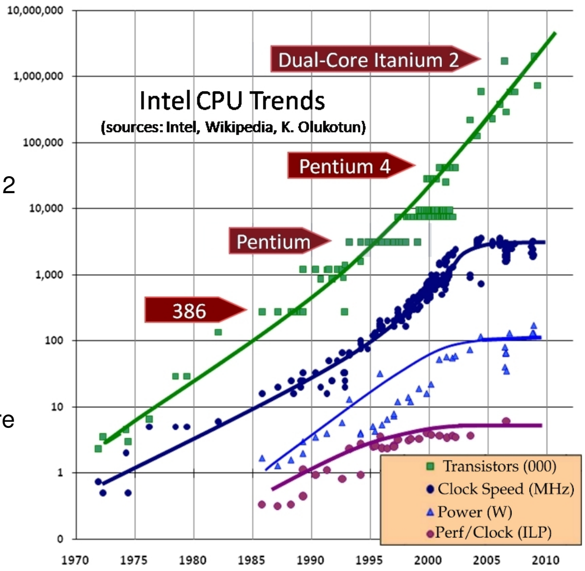
Outline

Outline

CPU trends

- Moore's law:
  $2\times$#transistors every 2 years

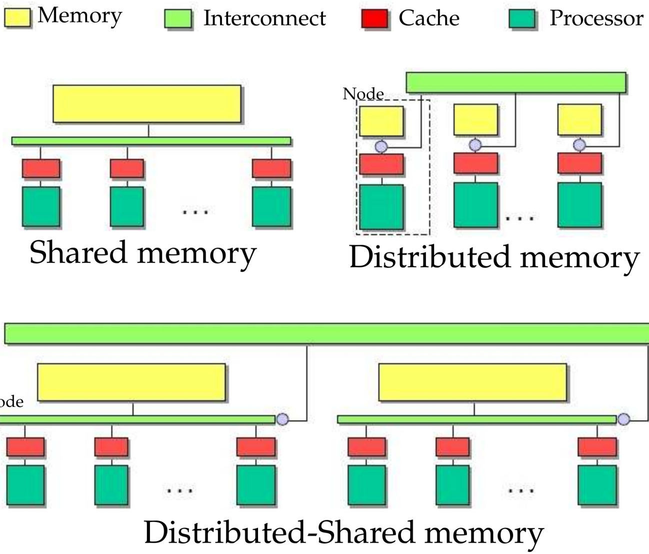
- Future trends:
  more processors in a single chip + hardware accelerators



CPU trends

We have to adapt our CPU/memory intensive algorithms (like GAs) to parallel computer architectures in order to exploit all their capabilities.

Examples of parallel computer architectures



Programming models

- Shared memory: **OpenMP**
  (addition of `#pragmas`, minimal modifications in the code)
- Distributed memory: **MPI**
  (synchronization of low-level network messages between nodes, considerable modifications in the code)
- Distributed-shared memory: **MPI+OpenMP**

Programming models

- Shared memory: **OpenMP**
  (addition of `#pragmas`, minimal modifications in the code)

- Distributed memory: **MPI**
  (synchronization of low-level network messages between nodes, considerable modifications in the code)

- Distributed-shared memory: **MPI+OpenMP** (best performance)

Exploiting parallelism in GAs

Task: we need to calculate the fitness function of N individuals.

Elements: P nodes, C processors per node.

Proposed solution: MPI+OpenMP based solution

- Distribute the population in N/P individuals per node (MPI).
- Each node calculates the fitness function of N/P individuals using C processors (OpenMP parallel loops).
- Each node shares its best individuals with the other nodes (MPI).

Outline

Speedup results

- **Hardware**: 126 nodes (12 CPUs, 2.53 GHz), 24 GB RAM/node.
- **OpenMP**: fitness function for an individual of size 1000×1000, 17-node template

CPUs	Time(secs)	Speedup
1	0.402347	1x
1+code opts	0.243200	1.65x
2+code opts	0.1280000	3.14x
4+code opts	0.076800004	5.23x
6+code opts	0.044799998	8.98x
8+code opts	0.038400002	10.47x
10+code opts	0.032000002	12.57x
12+code opts	0.025599999	15.71x

*code opts: techniques adapted from stencil optimization in Finite Differences methods, like memory accesses minimization, SIMDization, blocking, ...

Speedup results

- **Hardware**: 126 nodes (12 CPUs, 2.53 GHz), 24 GB RAM/node.
- **MPI+OpenMP**: 30 generations, 1000 individuals of size 1000×1000, 17-node template

Nodes/CPUs	Time(secs)	Speedup
1/1	9364	1x
1/12	1525	6.14x
2/12	781	11.98x
4/12	394	23.76x
8/12	203	46.12x

Outline

Conclusions

- Easy implementation and parallelization: unfortunatelly it takes too much time to converge...
- Dynamic mutation rate (annealing-style) and dynamic template accelerates the convergence of MPS-GAs.
- Refinement of realizations: adding realizations generated with other methods to the initial population.
- *Future work*: inclusion of hardware accelerators (GPUs) with new pragma-based programming models

$$\text{MPI+OpenMP+}\{\text{OpenACC,OmpSs,...}\}$$

Thanks for your attention!

Contact: `oscar.peredo [at] bsc.es`
Contact: `jortiz [at] ing.uchile.cl`
`http://www.bsc.es`
`http://www.alges.cl`
`http://www.amtc.cl`