Experience in the Supercomputer Ben Arabí when solving competitive location problems

A. G. Arrondo
J. Fernández
J. L. Redondo
P. M. Ortigosa

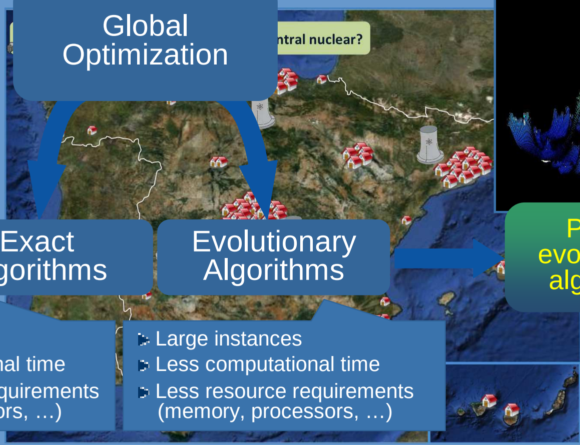
University of Murcia, March 2012

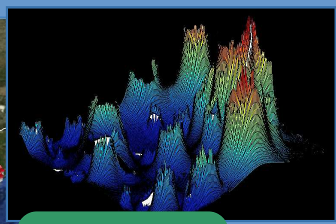
Motivation

Operational Research

- Transportation and logistics
- Routings problems
- Game theory
- Location problems, etc.

Global Optimization





Exact Algorithms

- Small instances
- Larger computational time
- Larger resource requirements (memory, processors, …)

Evolutionary Algorithms

- Large instances
- Less computational time
- Less resource requirements (memory, processors, …)

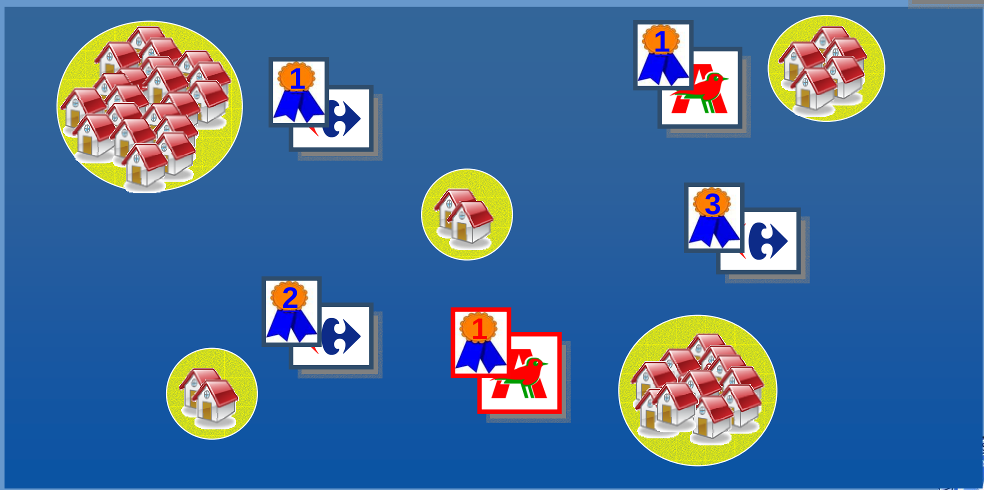
Parallel evolutionary algorithms

Outline

1. **The single facility location problem with variable demand**
2. **The leader-follower problem with variable demand**

The single location problem with variable demand

Scenario



The single location problem with variable demand

Mathematical formulation

Problem to solve

$$\begin{cases} \max \Pi(x,\alpha) = F(M(x,\alpha)) - G(x,\alpha) & \text{.............Profit obtained by the chain} \\ s.t. \quad d_i(x) \geq d_i^{\min} \forall i & \text{........................ Distance between demand point } p_i \text{ and x} \\ \qquad \alpha \in [\alpha_{\min}, \alpha_{\max}] & \text{... Quality of the new facility} \\ \qquad x \in S \subset \Re^2 & \text{..Location of the new facility} \end{cases}$$

$$F(M(x,\alpha)) = c \cdot M(x,\alpha)$$

$$G(x,\alpha) = G_1(x) + G_2(\alpha)$$

$M(x, \alpha)$ .... Market share attracted by the chain

c ………………Income per unit of goods sold

$G(x, \alpha)$ ....……………......... Operating costs

The single location problem with variable demand 6

Mathematical formulation

$$M(x,\alpha) = \sum_{i=1}^{n} \boxed{w_i(U_i)} \frac{u_{i0} + \sum_{j=1}^{k} u_{ij}}{u_{i0} + \sum_{j=1}^{m} u_{ij}}$$

$M(x, \alpha)$ .... Market share attracted by the chain

n .... Number of demand points

m .... Number of existing facilities

k .... Number of existing facilities which belong to the chain

$$U_i = u_{i0} + \sum_{j=1}^{m} u_{ij}$$

$$\boxed{w_i(U_i) = w_i^{\min} + incr_i \cdot e_i(U_i)}$$

$$incr_i = w_i^{\max} - w_i^{\min}$$

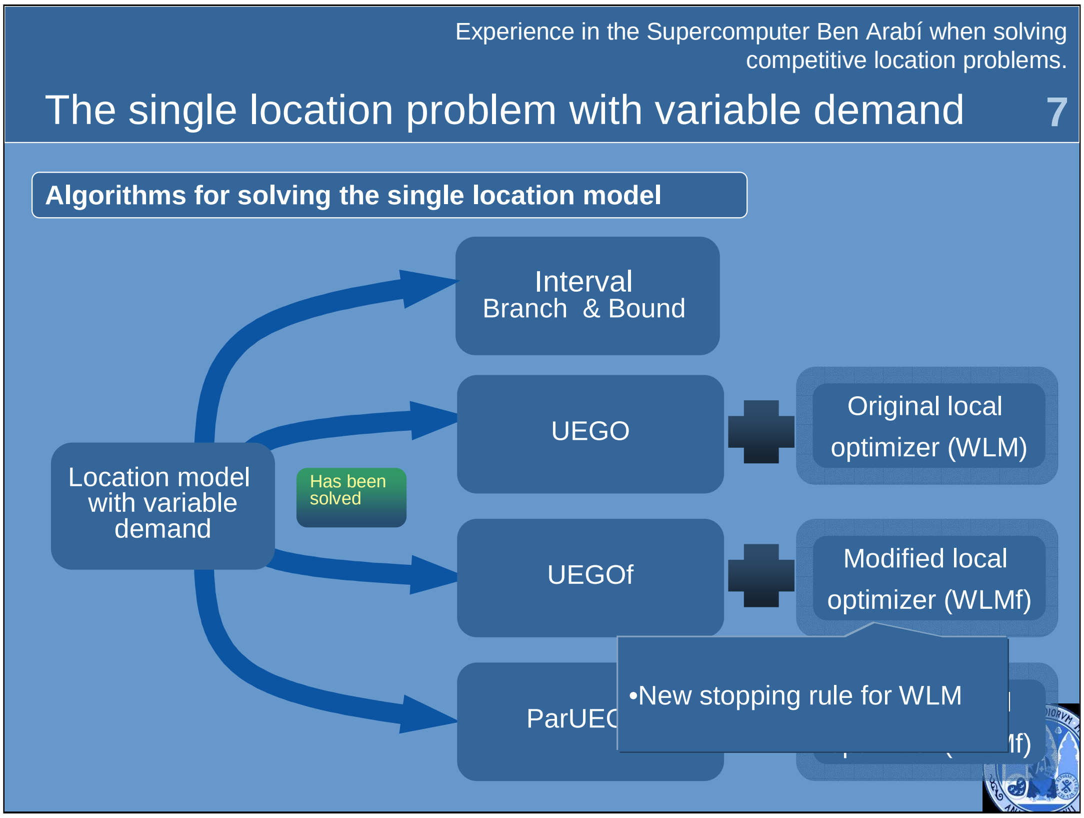
$$e_i(U_i) \in [0,1]$$

$\boxed{w_i(U_i)}$ ....Demand or buying power at pi

U_i ....Total utility derived by a customer at pi

$w_i^{\max}$ (resp. $w_i^{\min}$) .... Maximum (resp. minimum) possible demand at pi

$e_i(U_i)$ .... Share of the maximum possible increment that a customer decides to expend at pi

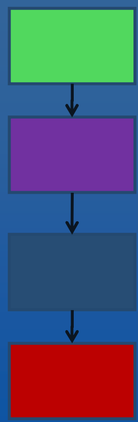

Experience in the Supercomputer Ben Arabí when solving competitive location problems.
The single location problem with variable demand
7
Algorithms for solving the single location model
Interval
Branch & Bound
UEGO
Original local
optimizer (WLM)
Location model
with variable
demand
Has been
solved
UEGOf
Modified local
optimizer (WLMf)
•New stopping rule for WLM

UEGO: Universal evolutionary global optimizer

Basic concepts

Specie:

- UEGO is an algorithm based on subpopulations (species).
- During the optimization process, a list of species is kept by UEGO.
- Each species can evolve to the local or global optima without participation of the remaining ones.



UEGO: Universal evolutionary global optimizer

Parameters…

User given parameters

Evals (N): The maximum number of function evaluations for the whole optimization process.

levels (L): The maximum number of levels.

max_spec_ num (M): The maximum length of the species list.

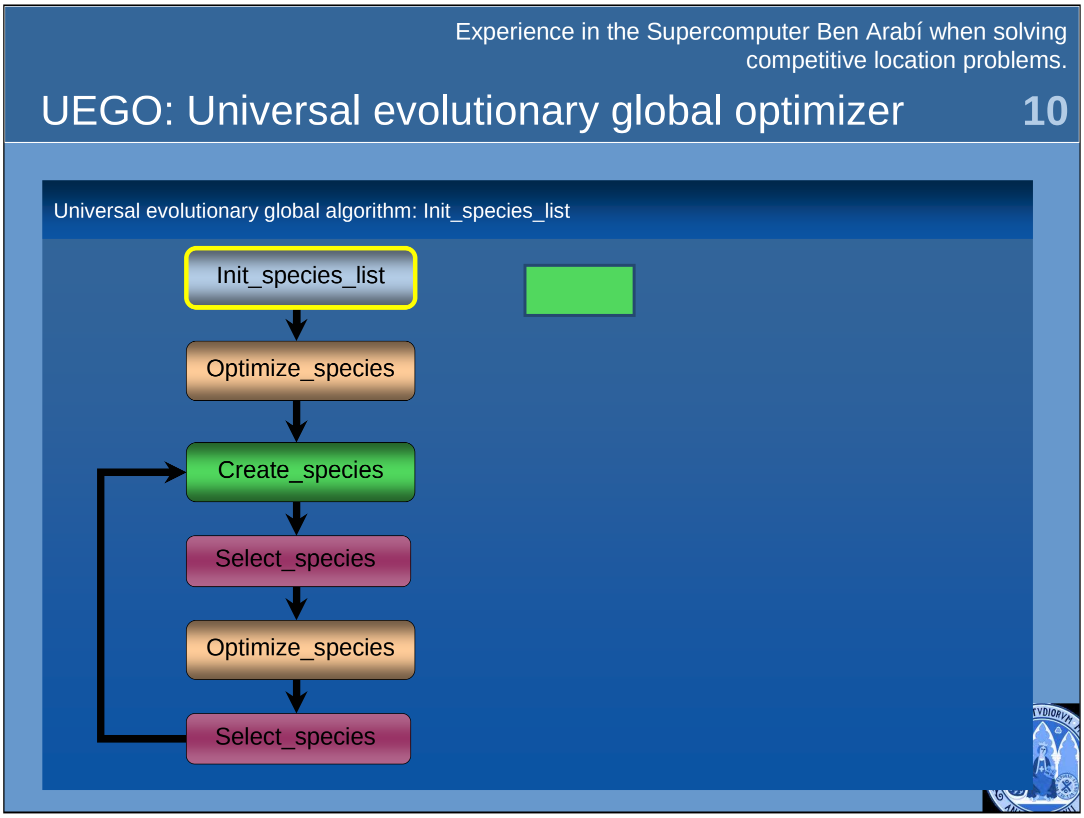
min_r (R_L): The radius that is associated with the minimum level.

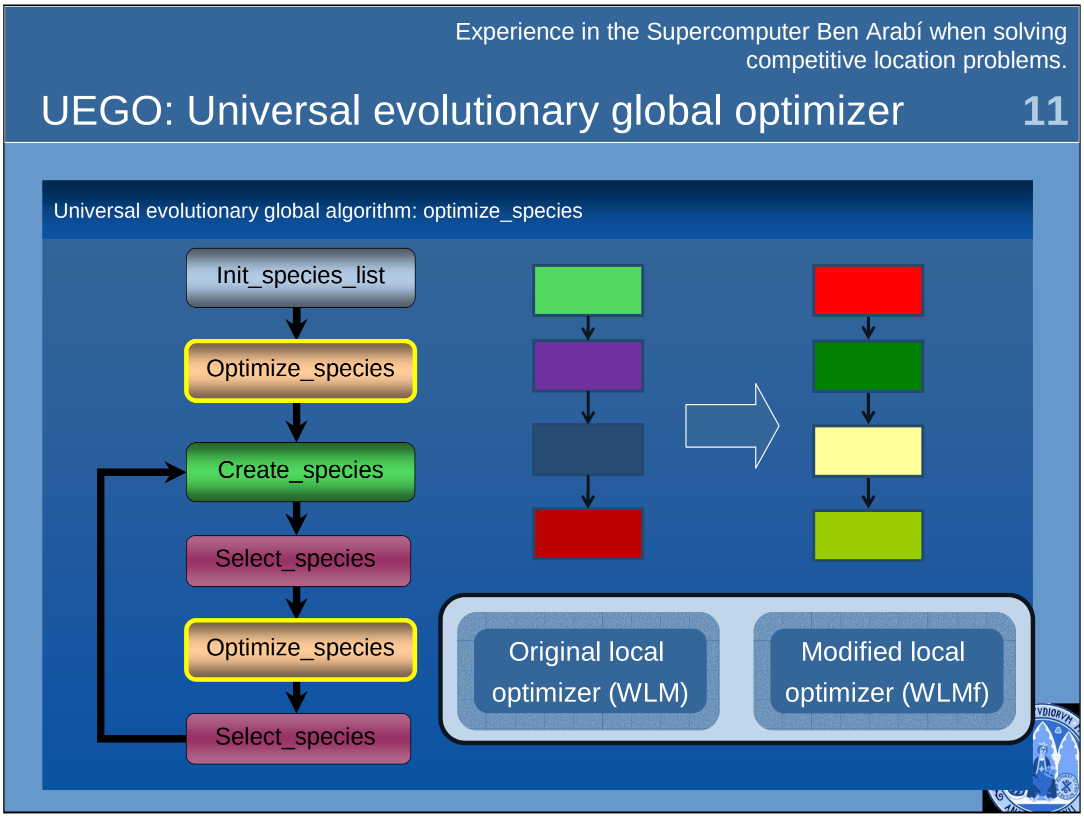
Parameter at each level

R_i : Radius associated with level *i*.

new_i : Maximum number of function evaluations allowed when creating new species.

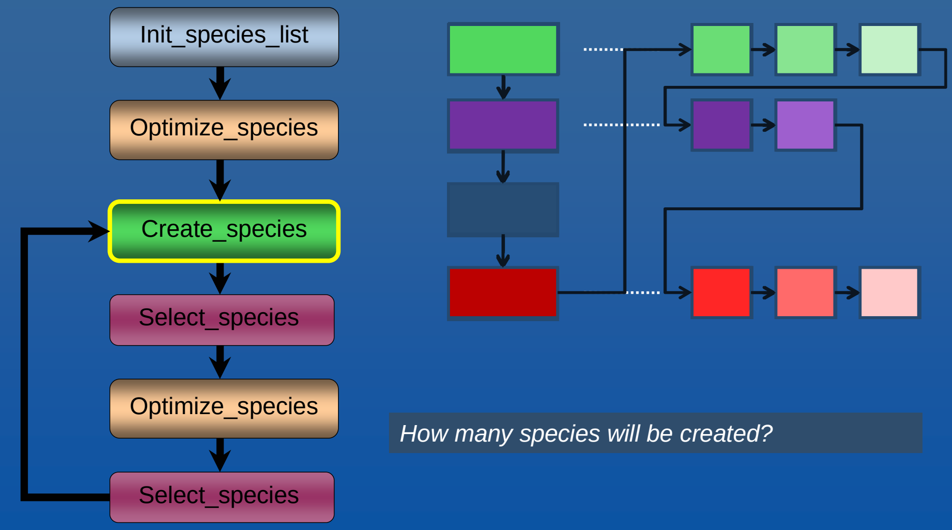
n_i : Maximum number of function evaluations allowed when optimizing species.


Experience in the Supercomputer Ben Arabí when solving competitive location problems.
UEGO: Universal evolutionary global optimizer
10
Universal evolutionary global algorithm: Init_species_list
Init_species_list
Optimize_species
Create_species
Select_species
Optimize_species
Select_species


Experience in the Supercomputer Ben Arabí when solving competitive location problems.
UEGO: Universal evolutionary global optimizer
11
Universal evolutionary global algorithm: optimize_species
Init_species_list
Optimize_species
Create_species
Select_species
Optimize_species
Select_species
Original local optimizer (WLM)
Modified local optimizer (WLMf)

UEGO: Universal evolutionary global optimizer

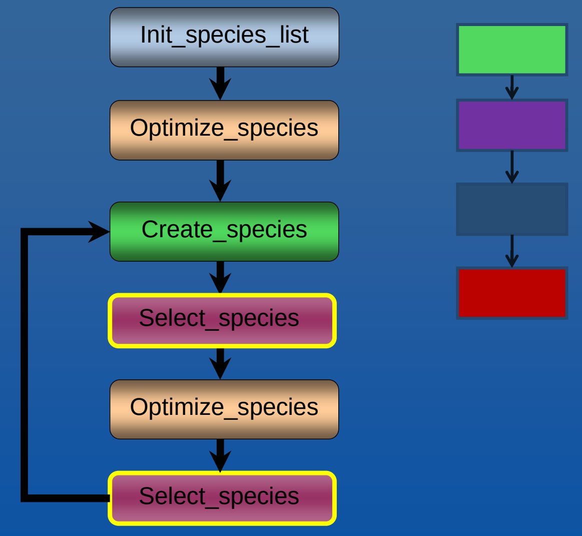
Universal evolutionary global algorithm: create_species

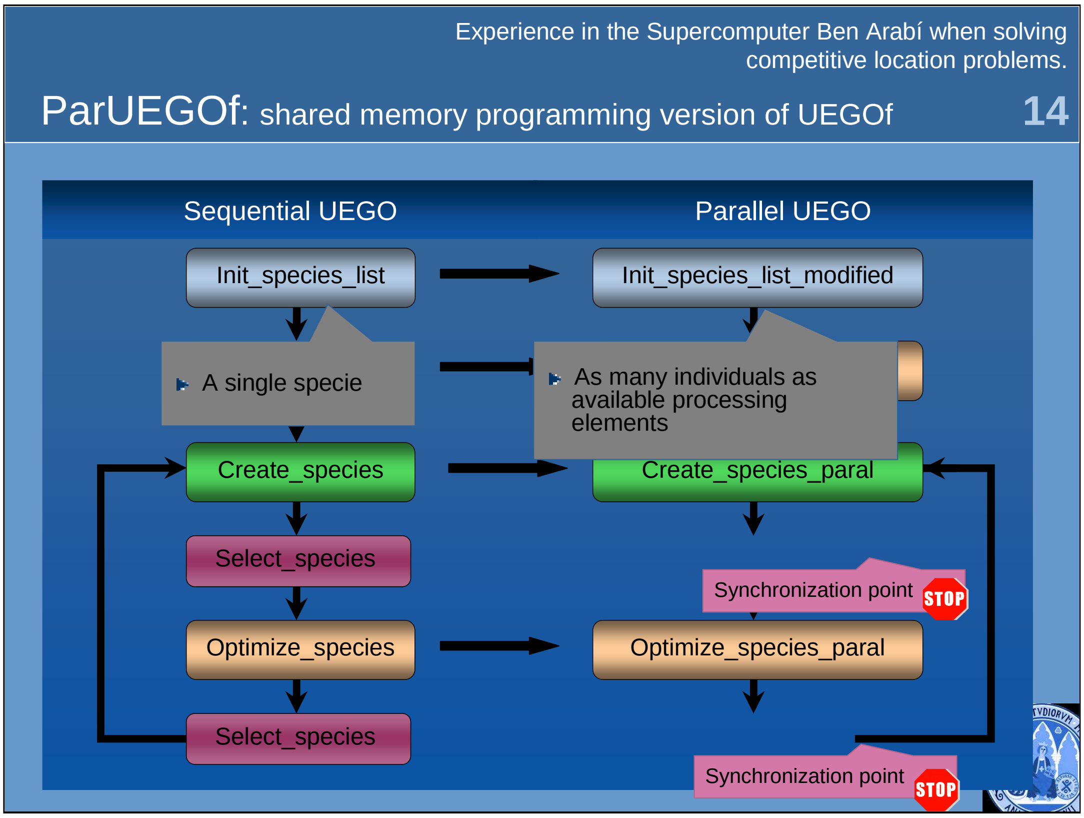


How many species will be created?

UEGO: Universal evolutionary global optimizer

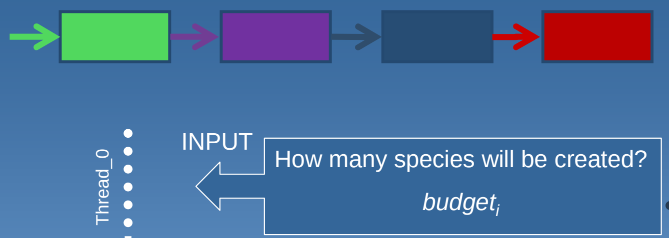
Universal evolutionary global algorithm: select_species

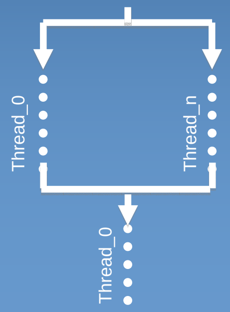



Experience in the Supercomputer Ben Arabí when solving competitive location problems.
ParUEGOf: shared memory programming version of UEGOf
14
Sequential UEGO
Parallel UEGO
Init_species_list
Init_species_list_modified
A single specie
As many individuals as available processing elements
Create_species
Create_species_paral
Select_species
Synchronization point
STOP
Optimize_species
Optimize_species_paral
Select_species
Synchronization point
STOP

ParUEGOf: shared memory programming version of UEGOf

Creation_species_paral





- Each thread:
  1) Creation of a new set of candidate solutions (sub-list).
  2) Partial selection on its sub-list.
  3) Updating the main species list with its sub-list.

Shared variables

- length(species_list$_i$)
- new$_i$
- *end

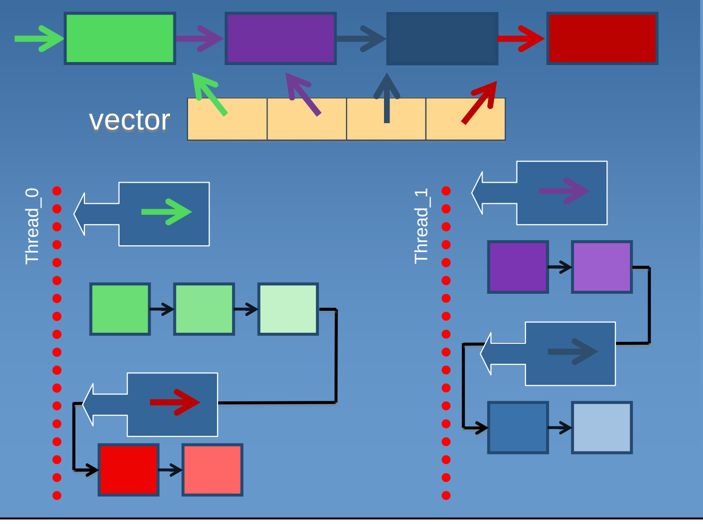
Private variables

- budget$_i$
- sub-list

ParUEGOf: shared memory programming version of UEGOf

Creation_species_paral: 1) Creation of new candidates

How to do it?



Shared variables

- length(species_list$_i$)
- new$_i$
- *end
- vector$_i$

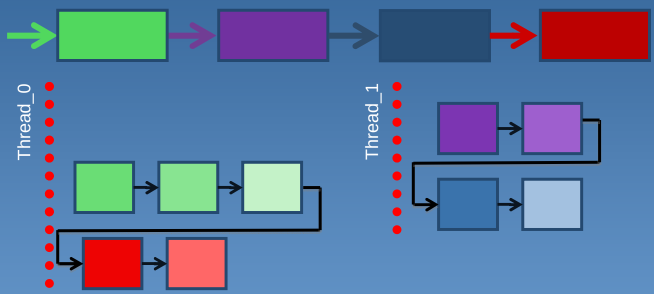
Private variables

- budget$_i$
- sub-list

ParUEGOf: shared memory programming version of UEGOf

Creation_species_paral: 2) Partial selection on threads

How to do it?

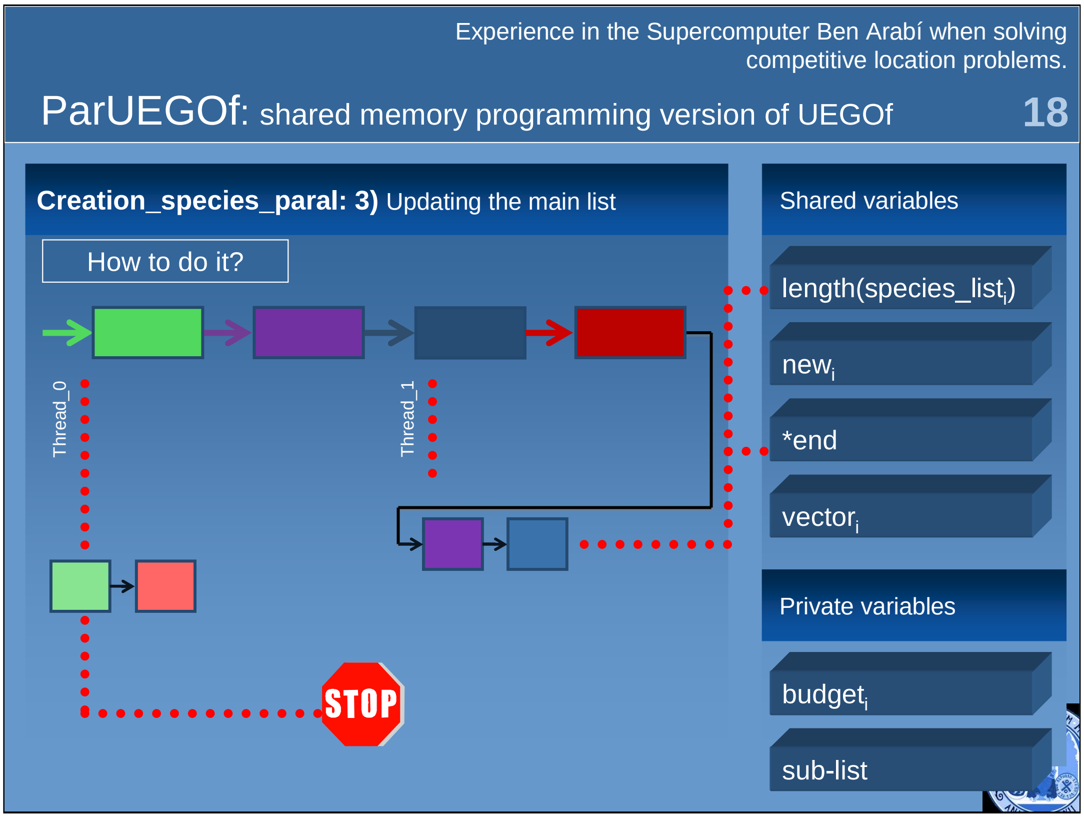


Shared variables

- $length(species\_list_i)$
- new_i
- *end
- $vector_i$

Private variables

- $budget_i$
- sub-list


Experience in the Supercomputer Ben Arabí when solving competitive location problems.
ParUEGOf: shared memory programming version of UEGOf
18
Creation_species_paral: 3) Updating the main list
How to do it?
Thread_0
Thread_1
STOP
Shared variables
length(species_list$_i$)
new$_i$
*end
vector$_i$
Private variables
budget$_i$
sub-list

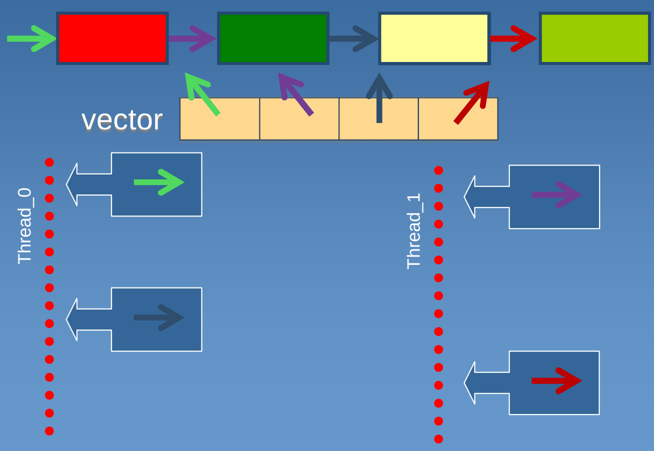
Experience in the Supercomputer Ben Arabí when solving competitive location problems.

ParUEGOf: shared memory programming version of UEGOf

19

Optimize_species_paral: modified local optimizer (WLMf)

How to do it?



Shared variables

- length(species_list$_i$)
- new$_i$
- vector$_i$

Private variables

Computational studies

Environment

- The input parameters were set to $N = 1 \cdot 10^8$, $M = 350$, $L = 30$ and $R_L = 0.05$ for all the instances and algorithms.
- Algorithms have been implemented in C++.
- Shared-memory applications programming interface used was OpenMP (version 3.0, supported by gcc 4.6.0).
- All the computational studies have been obtained in the Supercomputer BenArabi of Murcia, Spain. The shared-memory machine is a HP Integrity Superdome with 128 cores and 1.5 TB of memory.

Computational studies

Test problems

n	50			100			200		
m	2	5	10	2	5	10	2	10	15
k	0,1	0,1,2	0,2,4	0,1	0,1,2	0,2,4	0,1	0,2,4	0,5,10
S	$([0,10],[0,10])$								

Table 2. Settings of the small test problems.

iB&B	24 x 10 x 1 = 240 executions
UEGO	24 x 10 x 5 = 1200 executions
UEGOf	24 x 10 x 5 = 1200 executions
Total	2640 executions

Execution per instance

Instance per setting

Settings

- n Number of demand points
- m Number of existing facilities
- k Number of existing facilities which belong to the chain
- S Region of the plane where the new facility can be located

Computational studies

Test problems

n	500			1000			5000		
m	2	15	25	5	25	50	10	50	100
k	0,1	0,5,10	0,7,15	0,1,2	0,7,15	0,15,30	0,2,4	0,15,30	0,25,50
S	$([0,25],[0,25])$			$([0,50],[0,50])$					

n	10000			20000		
m	20	100	200	40	200	400
k	0,5,10	0,25,50	0,50,100	0,10,20	0,50,100	0,100,200
S	$([0,100],[0,100])$			$([0,200],[0,200])$		

Table 3. Settings of the larger test problems.

UEGO	26 x 5 x 5 x 1 = 650 executions
UEGOf	44 x 5 x 5 x 1 = 650 executions
ParUEGOf	44 x 5 x 5 x 8 = 8800 executions
Total	10100 executions

128)

Instance per setting

Settings

Computational studies

Results for small problems

- iB&B can solve the small problems with reliability.
- UEGO and UEGOf find the optimal solution with 100% success.
- UEGO is much faster than iB&B.
- UEGOf is faster than UEGO, expect for the easiest problems with setting (n=50, m=2,5) reductions varying from 1% up to 37% in some cases.

n	m	$T(iB\&B)$	$T(UEGO)$	$T(UEGOf)$	$\%I_{UEGO-iB\&B}$	$\%I_{UEGOf-iB\&B}$	$\%I_{UEGOf-UEGO}$
50	2	42.377	5.974	8.894	85.90	79.01	-48.86
	5	56.757	7.651	8.165	86.52	85.61	-6.71
	10	57.860	8.547	8.443	85.23	85.41	1.21
100	2	232.786	17.063	14.593	92.67	93.73	14.48
	5	254.252	20.500	15.039	91.94	94.08	26.64
	10	293.232	22.917	14.267	92.18	95.13	37.75
200	2	1320.607	36.510	28.821	97.24	97.82	21.06
	10	1491.524	33.246	28.755	97.77	98.07	13.51
	15	1473.295	32.471	28.341	97.80	98.08	12.72

Table 4. Average results for small problems.

.

Computational studies

Result for larger problems

- iB&B runs out of memory.
- Both UEGO and UEGOf can solve them without difficulties.
- Both UEGO and UEGOf are rather robust (it obtains the same solution in all the runs).
- UEGOf is much faster than UEGO (around 50%).
- UEGO has not been able to solve the largest problems.

	Time	ObjF	MaxDist	Time	ObjF	MaxDist	Time	ObjF	MaxDist
Algorithm	$n = 500$			$n = 1000$			$n = 5000$		
UEGO	428	21.903	0.000	1015	6.684	0.000	-	-	-
UEGOf	286	21.903	0.000	352	6.683	0.000	1727	-31.663	0.000
ParUEGOf(2)	146	21.902	0.000	178	6.684	0.000	869	-31.663	0.000
ParUEGOf(4)	76	21.903	0.000	90	6.683	0.000	436	-31.663	0.000
ParUEGOf(8)	40	21.903	0.000	48	6.683	0.000	226	-31.663	0.000
ParUEGOf(16)	23	21.902	0.000	27	6.684	0.000	123	-31.663	0.000
ParUEGOf(32)	14	21.902	0.000	17	6.684	0.000	72	-31.664	0.000
ParUEGOf(64)	10	21.902	0.000	12	6.684	0.000	48	-31.663	0.000
ParUEGOf(128)	9	21.902	0.000	11	6.684	0.000	37	-31.663	0.000

Table 5. Average results for larger problems.

Computational studies

Result for larger problems

Algorithm	Time	ObjF	MaxDist	Time	ObjF	MaxDist
	$n = 10000$			$n = 20000$		
UEGO	-	-	-	-	-	-
UEGOf	2914	-21.299	0.000	5811	-17.057	0.000
ParUEGOf(2)	1463	-21.299	0.000	2918	-17.057	0.000
ParUEGOf(4)	737	-21.299	0.000	1469	-17.057	0.000
ParUEGOf(8)	378	-21.298	0.000	754	-17.057	0.000
ParUEGOf(16)	199	-21.299	0.000	397	-17.057	0.000
ParUEGOf(32)	111	-21.298	0.000	238	-17.057	0.000
ParUEGOf(64)	70	-21.299	0.000	209	-17.057	0.000
ParUEGOf(128)	52	-21.299	0.000	204	-17.057	0.000

Table 6. Average results for larger problems.

Computational studies

Performance evaluation

$$Eff(P) = \frac{T(1)}{P \cdot T(P)}$$

Notation:

P: number of processors

P	n = 500	1000	5000	10000	20000
2	0.982	0.991	0.995	0.996	0.996
4	0.943	0.978	0.990	0.989	0.989
8	0.887	0.928	0.957	0.963	0.964
16	0.773	0.818	0.882	0.915	0.916
32	0.621	0.648	0.753	0.819	0.777
64	0.428	0.442	0.563	0.652	0.443
128	0.253	0.256	0.364	0.444	0.228

Figure 7. Average efficiency of ParUEGOf algorithm.

Conclusions

- The deterministic exact interval B&B method can only solve small problem. For larger problems heuristics are mandatory.
- The evolutionary algorithms UEGO and UEGOf can find the optimal solution of small problems with 100% success, much faster than B&B.
- For larger problems, the heuristic algorithms are rather robust.
- UEGO can solve problems with up to 1000 demand points and UEGOf is able to solve problems with up to 20000 demand points.
- A parallel algorithm to reduce the runtime of UEGOf has been developed. This new method has a good behaviour in terms of effectiveness and efficiency.

Publications

[1] J. L. Redondo, J. Fernández, A. G. Arrondo, I. García, P. M. Ortigosa, *Deterministic or variable demand? Does it matter when locating a facility?.* Omega, Vol. 40, n. 1, pp, 99-20. ISSN: 0305-0483. 2012. Category (position/total): Operations research & Management science: 2/73

[2] A. G. Arrondo, J. Fernández, , J. L. Redondo and P. M. Ortigosa, *An approach for solving competitive location problems with variable demand using multicore systems.* Optimization Letters, submitted.

Use of Ben Arabí supercomputer


```
aranzazugila$
aranzazugila$ ssh -XY ben
aranzazugila$ cd carpeta_algoritmo
aranzazugila$ module load gcc/4.6.0
aranzazugila$ make
aranzazugila$ cp uego carpeta_experimentos
aranzazugila$ cd carpeta_experimentos
aranzazugila$ bsub < script_r20000-200-0-1
aranzazugila$
```

scrip_r20000-200-0-1

```
#!/bin/bash
#BSUB -a openmp
#BSUB -J openmp_n20000
#BSUB -q ben_128x24h
#BSUB -n 128
#BSUB -u agarrondo@um.es
#BSUB -B
#BSUB -N
#BSUB -o result-20000-200-0-1.o
#BSUB -e result-20000-200-0-1.e

source /etc/profile.d/modules.sh
module load gcc/4.6.0

./uego r-20000-200-0-1 -s -t128 -r5
```

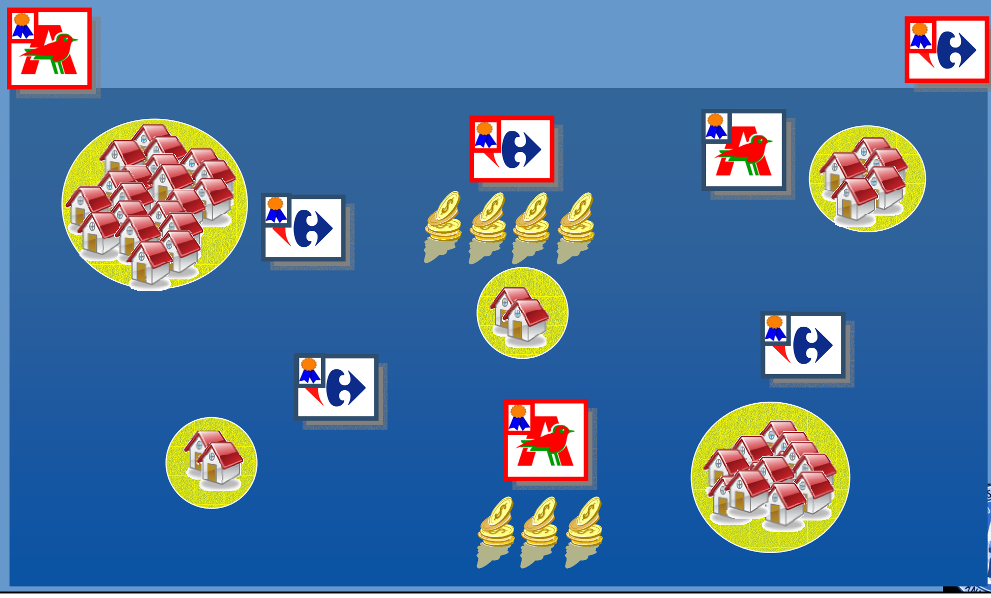
It is very important to know your code and the machine where your code is going to run.

Outline

1. The single facility location problem with variable demand
2. **The leader-follower problem with variable demand**

The leader-follower problem with variable demand

Scenario



The leader-follower problem with variable demand

Mathematical formulation

Market share attracted by the leader`s chain

$$M_1(nf_1, nf_2) = \sum_{i=1}^{n} \boxed{w_i(U_i)} \frac{\dfrac{\gamma_i \alpha_2}{g_i(d_{iz_2})} + \displaystyle\sum_{j=1}^{k} \frac{\alpha_{ij}}{g_i(d_{ij})}}{\dfrac{\gamma_i \alpha_1}{g_i(d_{iz_1})} + \dfrac{\gamma_i \alpha_2}{g_i(d_{iz_2})} + \displaystyle\sum_{j=1}^{m} \frac{\alpha_{ij}}{g_i(d_{ij})}}$$

Problem to solve

$$(LP) \begin{cases} \max \ \ \Pi_1(nf_1, nf_2^*(nf_1)) = F_1(M_1(nf_1, nf_2^*(nf_1))) - G_1(nf_1) \\ s.t. \quad z_1 \in S_1 \subset \Re^2 \\ \qquad d_{iz_1} \geq d_i^{\min} \quad i = 1, \ldots, n \\ \qquad \alpha_1 \in \left[q_1^{\min}, q_1^{\max}\right] \end{cases}$$

The leader-follower problem with variable demand

The Algorithms

The follower (medianoid) problem

- Interval Branch & Bound
- UEGO + Original local optimizer (WLM)
- UEGOf + Modified local optimizer (WLMf)
- ParUEGOf + Modified local optimizer (WLMf)

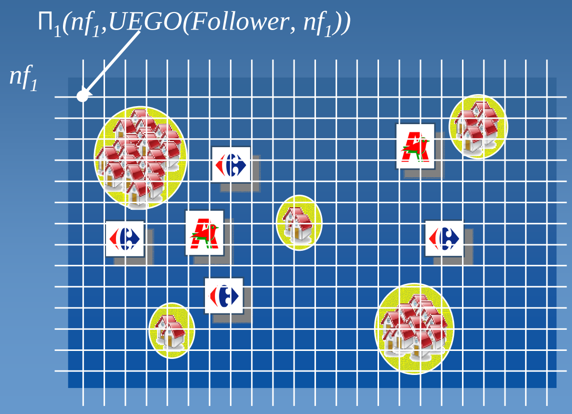
The leader (centroid) problem

- GS: a grid search procedure
- MSH: multistart procedure → MSH_BB, MSH_UE
- TLUEGO → TLUEGO_BB, TLUEGO_UE

The leader-follower problem with variable demand

Algorithms to solve the leader (centroid) problem

GS: A grid search procedure



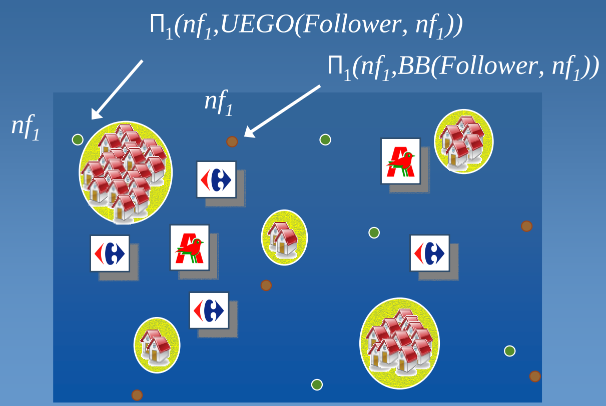
The leader-follower problem with variable demand

Algorithms to solve the leader (centroid) problem

MSH: A multistart procedure

MSH_UE

MSH_BB



The leader-follower problem with variable demand

Algorithms to solve the leader (centroid) problem

TLUEGO: Two-level evolutionary algorithm for the centroid problem

Computational studies

Environment

- The input parameters were set to $N = 1 \cdot 10^6$, $M = 350$, $L = 30$ and $R_L = 0.05$ for all the instances and algorithms.
- Algorithms have been implemented in C++.
- All the computational studies have been obtained in a processor Xeon IV with 2.4GHz and 1GByte RAM.

Computational studies

Test problems

n	15			25			50		
m	2	5	10	2	5	10	2	5	10
k	0,1	0,1,2	0,2,4	0,1	0,1,2	0,2,4	0,1	0,1,2	0,2,4
S	$([0, 10], [0, 10])$								

Table 2. Settings of the test problems.

GS	24 x 1 x 1 = 24 executions
MSH_UE	24 x 1 x 10 = 240 executions
MSH_BB	24 x 1 x 10 = 240 executions
TLUEGO_BB	24 x 1 x 10 = 240 executions
TLUEGO_UE	24 x 1 x 10 = 240 executions
Total	984 executions

Execution per instance

Instance per setting

Settings

n Number of demand points

m Number of existing facilities

k Number of existing facilities which belong to the chain

S Region of the plane where the new facility can be located

Computational studies

Results

- GS is rather time-consuming and is not able to find the global optimum.
- TLUEGO and MSH behave similarly independent of whether iB&B or UEGO is employed.
- TLUEGO is both the algorithm giving the best and most robust results, and this using less computational time.
- MSH provides the worst objective function value and different runs may provide very different objective values.

Algorithm	Av(T) Secs.	Max Dist	Objective Function Min	Objective Function Av	Objective Function Max	Dev
TLUEGO_BB	3674	0.099	28.149	28.189	28.230	0.033
TLUEGO_UE	3690	0.123	28.151	28.219	28.264	0.043
MSH_BB	4316	1.614	21.034	23.804	26.958	2.206
MSH_UE	4528	1.528	20.845	24.834	27.283	2.371
GS	1469370	-	-	27.052	-	-

Table 4. Average results considering all the problems (n=15,25, 50).

Conclusions

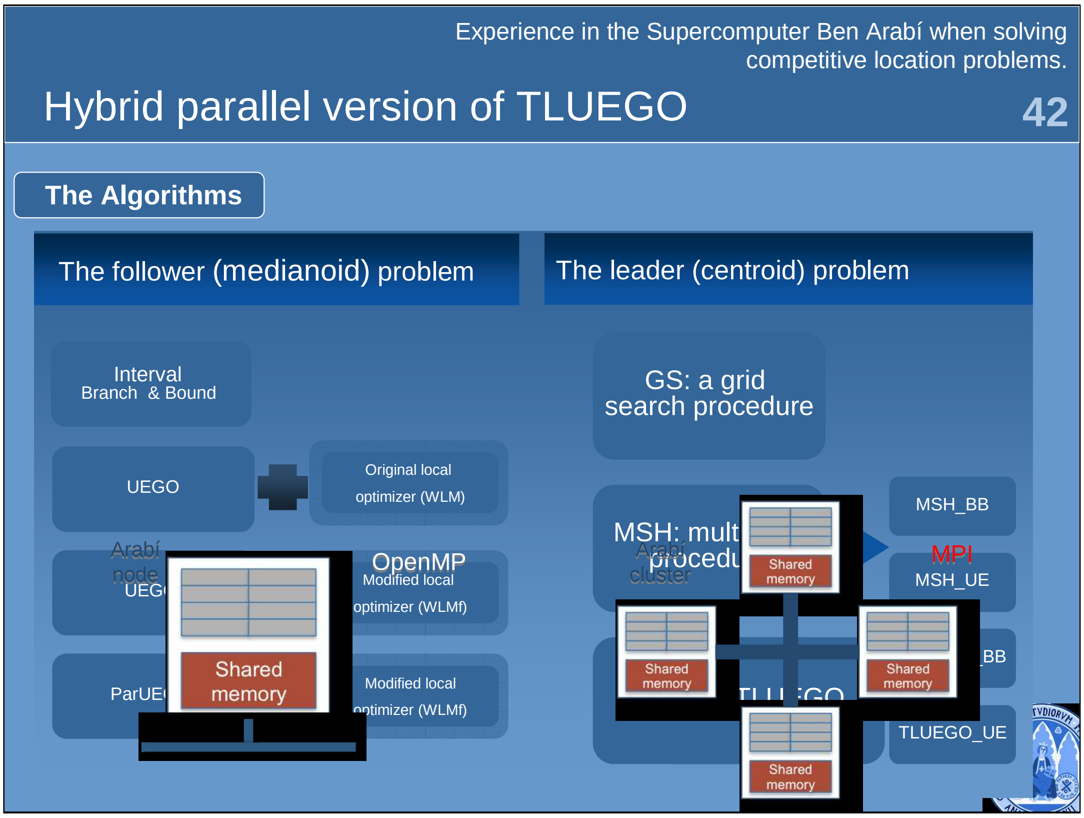
- The computational studies have shown that the evolutionary algorithm TLUEGO provides the best results and is more robust than the other strategies.

- However, the computational time employed by TLUEGO for solving a problem with 50 demand points is in average more than 2.5 hours.

- This clearly suggests that a parallelization of the algorithm is needed.

Publications

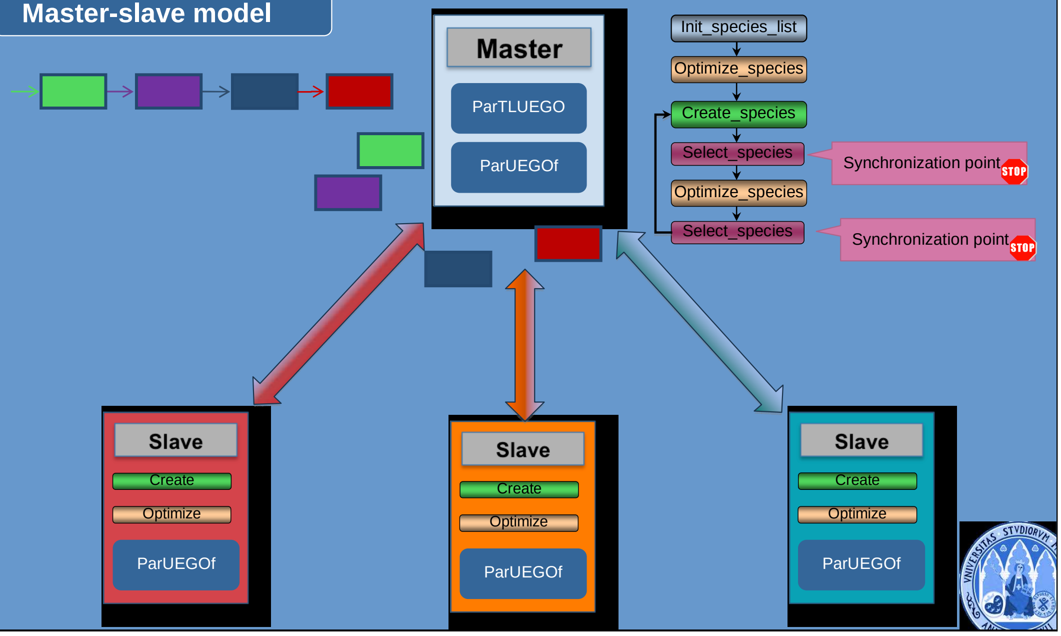
[3] J. L. Redondo, A. G. Arrondo, J. Fernández and P. M. Ortigosa, *A Two-level evolutionary algorithm for solving the facility location and design (1|1)-centroid problem on the plane with variable demand.* Journal of Global Optimization.



Experience in the Supercomputer Ben Arabí when solving competitive location problems.
Hybrid parallel version of TLUEGO
42
The Algorithms
The follower (medianoid) problem
The leader (centroid) problem
Interval Branch & Bound
UEGO
Original local optimizer (WLM)
Arabí node
OpenMP
Modified local optimizer (WLMf)
Shared memory
ParUE
Modified local optimizer (WLMf)
GS: a grid search procedure
MSH_BB
MSH_UE
MPI
Arabí cluster
Shared memory
Shared memory
Shared memory
Shared memory
BB
TLUEGO_UE

Hybrid parallel version of TLUEGO

Master-slave model



Use of Ben Arabí supercomputer


```
aranzazugila$ ssh -XY arabi
aranzazugila$ cd carpeta_algoritmo
aranzazugila$ module load impi
aranzazugila$ make
aranzazugila$ cp uego carpeta_experimentos
aranzazugila$ cd carpeta_experimentos
aranzazugila$ bsub < script_2nodes
aranzazugila$ bsub < script_4nodes
```

script_2nodes

```
#!/bin/bash
#BSUB -J mpi_n50-2-0-1
#BSUB -q arabi_192x72h
#BSUB -n 16
#BSUB -u agarrondo@um.es
#BSUB -B
#BSUB -N
#BSUB -o result-50-2-0-1.o
#BSUB -e result-50-2-0-1.e

source /etc/profile.d/modules.sh
module load impi

impiexec.lsf -ppn 1 ./uego r-50-2-0-1 -s -r5
```

script_4nodes

```
#!/bin/bash
#BSUB -J mpi_n50-2-0-1
#BSUB -q arabi_192x72h
#BSUB -n 32
#BSUB -u agarrondo@um.es
#BSUB -B
#BSUB -N
#BSUB -o result-50-2-0-1.o
#BSUB -e result-50-2-0-1.e

source /etc/profile.d/modules.sh
module load impi

impiexec.lsf -ppn 1 ./uego r-50-2-0-1 -s -r5
```

It is very important to know your code and the machine where your code is going to run.

Conclusions

- Compiler
- Architecture of machine
- Synchronization points, load balance
- Perform speedup with your serial code

Solving competitive location problems via evolutionary algorithms in Ben Arabí supercomputer.

A. G. Arrondo
J. Fernández
J. L. Redondo
P. M. Ortigosa

University of Murcia, March 2012