Application of parallel metaheuristics to an execution time-power consumption bi-objective problem

José M. Cruz-Zapata
Domingo Giménez
Daniel Ruiz-García

Departamento de Informática y Sistemas, Universidad de Murcia

META'14, October 27-31, 2014

Time and energy optimization

- Traditionally, parallel algorithms to reduce the execution time.
- Today, Green computing $\Rightarrow$ development of power-aware algorithms.
- Bi-objective problem: reduction of execution time and power consumption.
- Based in simplified models of the execution time and the power consumption,
- in a heterogeneous system.

Contents

Contents

A master-slave scheme

IN PARALLEL in each process P_i $(i = 0, ..., p-1)$ DO
if $i = 0$ **then**
  for $j = 1$ TO $j = p-1$ **do**
    Send task to P_j
  end for
  for $j = 1$ TO $j = p-1$ **do**
    Receive solution from P_j
  end for
else
  Receive task from P_0
  Solve task
  Send solution to P_0
end if
END PARALLEL

Computational system

- p processors
- connected through an interconnection network.
- Heterogeneous in:
  - communication
  - computation
  - energy consumption

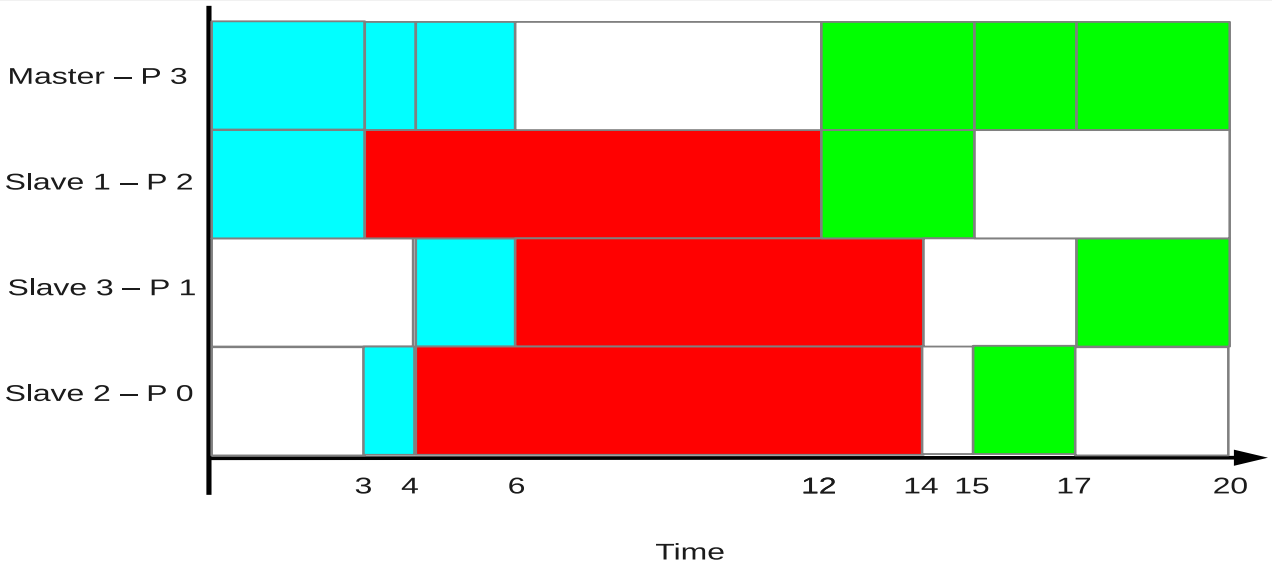
$\Rightarrow$ the costs vary depending on the parts of the algorithm and the processor where they are carried out or the source and target processors involved in a communication
$\Rightarrow$ on where the master and slave processes are assigned

processes-to-processors asignation problem

Parameters of the system

- Execution times:
  - $TimeComunT$, $p \times p$: costs of communications between two processors when sending-receiving a task.
  - $TimeComunS$, $p \times p$: costs of communications between two processors when sending-receiving a solution.
  - $TimeCompuT$, of size p: cost of solving a task by a process in a processor.
- Energy consumption:
  - $EnerComunT$, of size p: on a processor working on the communication of a task.
  - $EnerComunS$, of size p: on a processor working on the communication of the solution.
  - $EnerCompuT$, of size p: on a processor when working on the solution of a task.
  - $EnerInac$, of size p: when the processor is idle.

Example



TimeComunT

Source/Target	0	1	2	3
0	1	1	2	2
1	2	1	3	2
2	2	3	2	2
3	1	2	3	2

TimeComunS

Target/Source	0	1	2	3
0	1	2	1	2
1	1	2	2	2
2	2	2	3	2
3	2	3	3	3

Processor:	0	1	2	3
TimeComputT	10	8	9	12
EnerComunT	10	12	8	14
EnerComunS	12	14	10	14
EnerComputT	100	80	90	110
EnerInac	4	7	5	2

assignation $\pi = (2, 3, 1, 0) \Rightarrow \mathit{Time}(\pi) = 20,\ \mathit{Energy}(\pi) = 2910$

Contents

Sequential

Each individual is a permutation of $(0, 1, \ldots, p-1)$, and has associated the models of the execution time and the energy consumption.

- `GeneratePopulation`. Individuals are generated randomly. Execution time and energy consumption are calculated. The Pareto front is generated.
- `EndCondition`. A maximum number of iterations and a maximum number of iterations without including an individual in the front.
- `SelectParents`. With a roulette method, with more probability for individuals with low values of time and energy.
- `Crossover`. Pairs of individuals are crossed using the middle position of the permutations, and repeated values are substituted by values not in the element.
- `Mutation`. Permutation of two possitions.
- `SelectBestElements`. The elements with the best time or energy survive.

Shared-Memory (OpenMP)

- t threads: the population of size $|S|$ is divided in subpopulations of $|S|/t$ individuals.
- Each thread works independently for g iterations.
- After g iterations the threads share information: each thread stores in the global Pareto front his solutions with the lowest execution time and the lowest energy consumption.
- Each thread includes in its subpopulation pairs from the shared structure which are better in time or energy than some pair in its subpopulation.
- When the iterations finish, the Pareto front in the different threads are combined to obtain the final Pareto front.

Message-Passing (MPI)

Island model:

- Works similarly to the shared-memory version,
- but the master process manages the global Pareto front,
- and communicates the end condition.

An MPI+OpenMP hybrid version can be obtained easily.

Contents

Sequential

A set of particles, with each particle in a position given by the permutation representing the particle, and the movement speed is determined by its position with respect to the best local and global positions:

$$Sp(P) = w\ S + C_1\ R_1\ |LO(P) - P| + C_2\ R_2\ |GO - P|$$

w is the coefficient of inertia;
C_1 is the cognitive component, determines the influence of the previous position;
C_2 is the social component, determines the influence of the global optimum;
R_1 and R_2 random values between 0 and 1.
Distances are obtained as the sum of the quotient of the times and of the energies.

The speed represents the number of changes of pairs in the permutation representing the particle. Particles with higher speed analyze a wider neighborhood.

Shared-Memory (OpenMP)

- A number of particles is assigned to each thread.
- Synchronization after each iteration,
- and access in mutual exclusion to the global Pareto front when a new pair candidate to be included is found.

Message-Passing (MPI)

- A group of particles is assigned to each process.
- After each iteration broadcast from each process to communicate the local candidates to be included in the Pareto front.
- Each process stores and updates a copy of the Pareto front.
- Communications are asynchronous because the number of pairs to send-receive is not known.

An MPI+OpenMP hybrid version can be obtained easily.

Contents

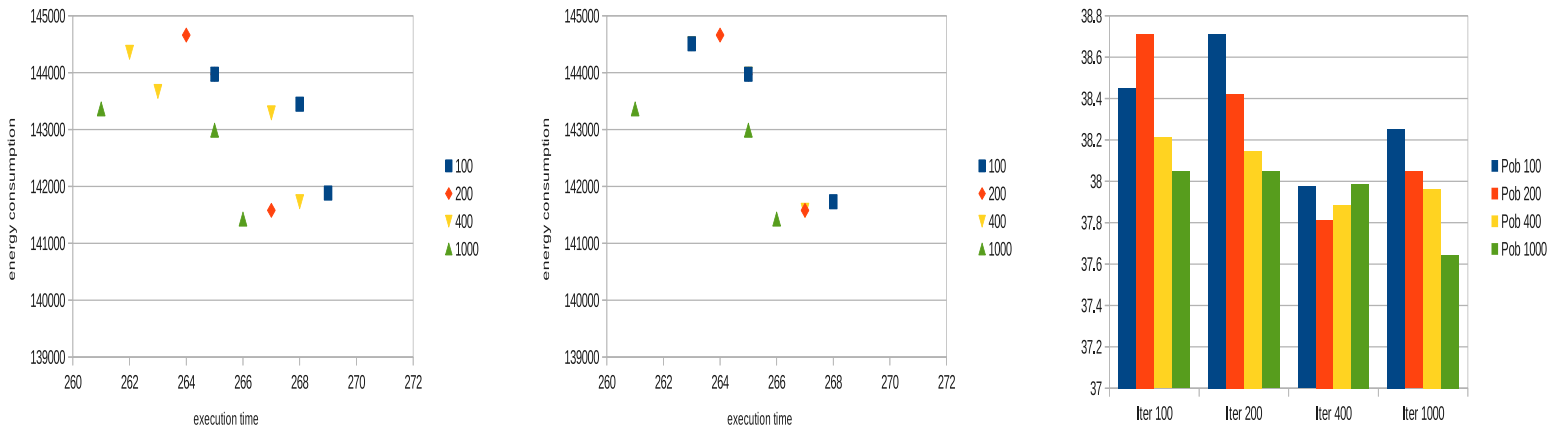
Computational system

A heterogeneous cluster with three nodes:

- *Saturno*: a NUMA system with 4 Intel hexa-core NEHALEM-EX EC E7530 (24 cores), 1.87 GHz, 32 GB of shared-memory.
- *Marte* and *Mercurio*: AMD Phenom II X6 1075T (hexa-core), 3 GHz, 15 GB (*Marte*) and 8 GB (*Mercurio*).

Pareto front

The two metaheuristics give similar results for the goodness of the solutions. Evolution of the Pareto front when the sequential GA is applied to a problem with 20 processes:

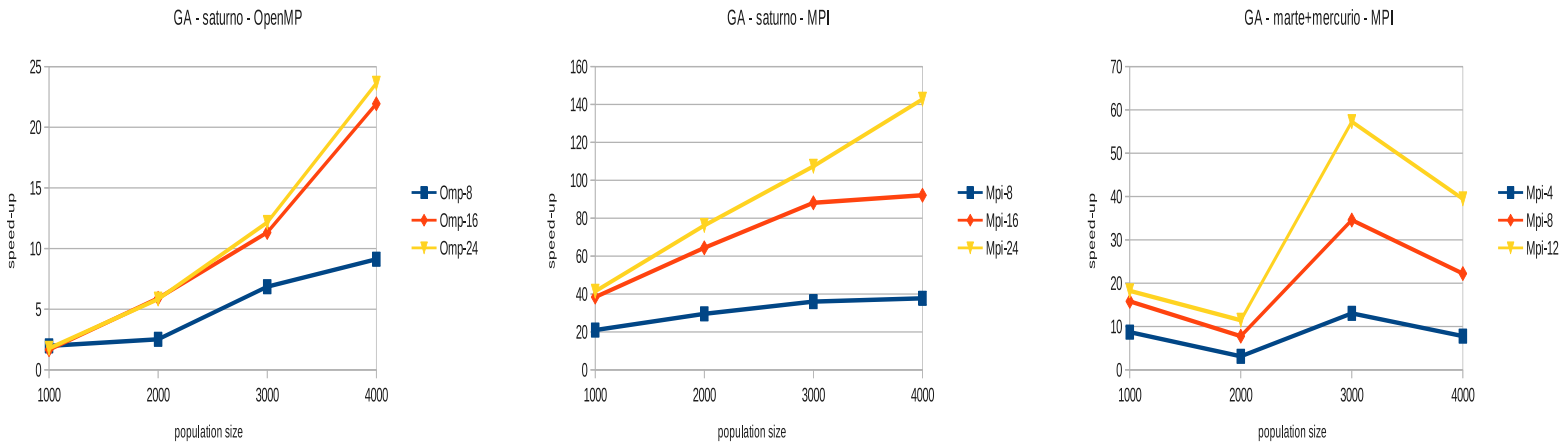


(a) (b) (c)

Figure: a) Population with 1000 individuals, the number of iterations varies. b) 1000 iterations, varying the population size. c) Mean of the products execution time-energy consumption of the pairs at the Pareto front, the population size and the number of iterations vary.

Parallelism - Genetic Algorithm

Speed-up of GA when varying the number of processes and threads (a problem with 40 tasks, 1000 iterations and varying the population size):

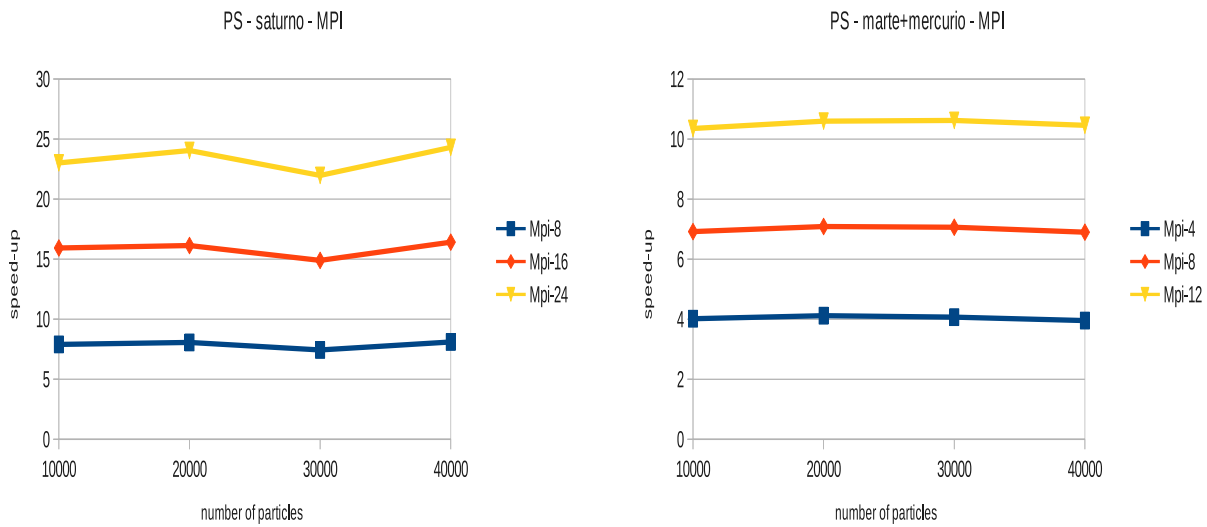


(a) (b) (c)

Figure: a) In *Saturno* with OpenMP. b) In *Saturno* with MPI. c) In *Marte+Mercurio* with MPI.

Parallelism - Particle Swarm Optimization

Speed-up of PSO when varying the number of processes (a problem with 40 tasks, 1000 iterations and varying the number of particles):

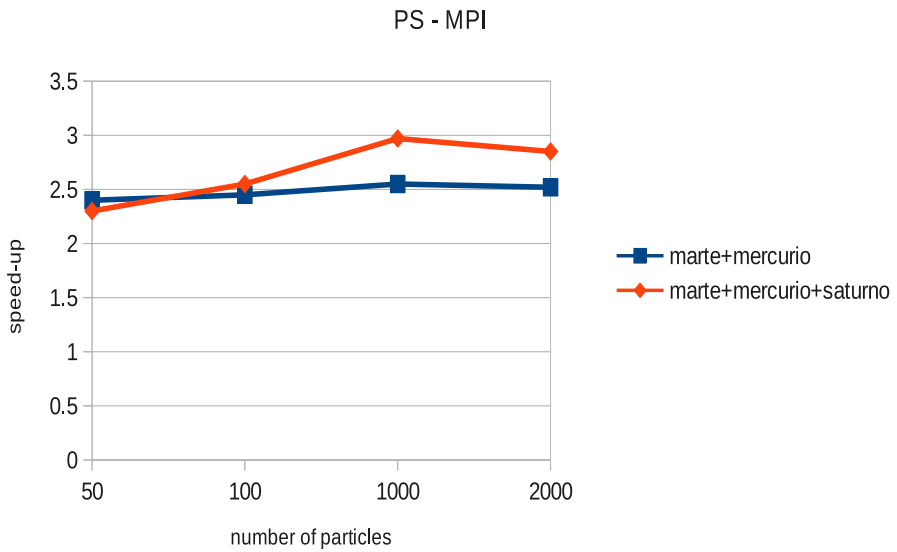


(a) (b)

Figure: a) In *Saturno* with MPI. b) In *Marte+Mercurio* with MPI.

Heterogeneous cluster

Speed-up of PSO when varying the number of particles:



- Larger systems can be used for larger sets of particles.
- If the nodes have similar speeds for the problem in question, satisfactory results are obtained with balanced distribution.
- Heterogeneous and hybrid versions could be used for higher speeds.

Contents

Conclusions

- An execution time-energy consumption bi-objective problem is tackled with metaheuristics.
  Results with Genetic Algorithms and Particle Swarm Optimization are shown.
  Similar results were obtained with other metaheuristics.
- A simple algorithm is considered, in a simulated system heterogeneous in computation and communications.
  The algorithm and the system correspond to real problems and systems.
- The speed-ups of the parallel versions are satisfactory for large populations and sets.

Future research

- Other parallel algorithmic schemes and particular applications can be considered, as can other metaheuristics.
- The use of heterogeneous systems can help to reduce the execution time.
  Heterogeneous metaheuristics should be developed,
  as should parallel implementations in GPU and hybrid systems.
- Hyperheuristics could be developed to select satisfactory metaheuristics for the problem.