Polish J. of Environ. Stud. Vol. 17, No. 4C (2008), 240-245

General Methodology for Converting Sequential Evolutionary Algorithms into Parallel Algorithms with OpenMP as Applied to Combinatorial Optimization Problems

O. Díaz-Parra, M. A. Cruz-Chávez

CIICAp, Universidad Autónoma del Estado de Morelos
Cuernavaca, Morelos, México

Abstract

This paper presents a general methodology for the conversion of sequential evolutionary algorithms into parallel evolutionary algorithms using OpenMP, independent of the problem being approached. Because of the diversity of existing problems and the variation between each one, it is difficult to find a standard or a method to make problems parallel. Analyses that are specific to problems being approached exist, but there is not a general methodology. This article shows its methodology by applying it to a combinatorial optimization problem of Vehicle Routing with time windows, using an evolutionary heuristic with intelligent mutation and k-means clustering as a solution method. This paper shows the analysis of the sequential algorithm and the task assignment to each processor in the parallel algorithm. The justification of the elements used for the conversion of sequential and parallel versions is also shown.

Keywords: sequential algorithm, parallel algorithm, genetic algorithm, Vehicle Routing Problem with time windows.

Introduction

Heuristics can be effectively used to approach solutions of combinatorial problems, but because the computation times can be very long, parallel computation has arisen as an alternative in order to decrease the time spent searching for solutions [1].

The process of decomposition of a sequential algorithm to a parallel algorithm implies the data parallelism and the functional or control parallelism. The first, data parallelism, is when several processors are assigned data items and each processor conducts the same operation concurrently on its data. In other words, it executes SIMD (Simple Instruction Multiple Data), which utilizes the same sequence of instructions on different data items. The second, functional or control parallelism, involves applying different operations simultaneously on different data items, which are executed on MIMD machines (Multiple Instruction Multiple Data). In order to make a process parallel, it is necessary to take into account its granularity (number of sequential instructions that the process includes). When it is a process with many instructions, it is considered coarse-grained, when it is a process with a smaller number of sequential instructions, it is considered fine-grained.

There are different ways to design a parallel algorithm. In this paper, a sequential genetic algorithm is converted into a parallel genetic algorithm, exploiting the parts that are able to be parallelized [2]. The initial sequential evolutionary algorithm is a genetic algorithm that yields a solution to the Vehicle Routing Problem with Time Windows

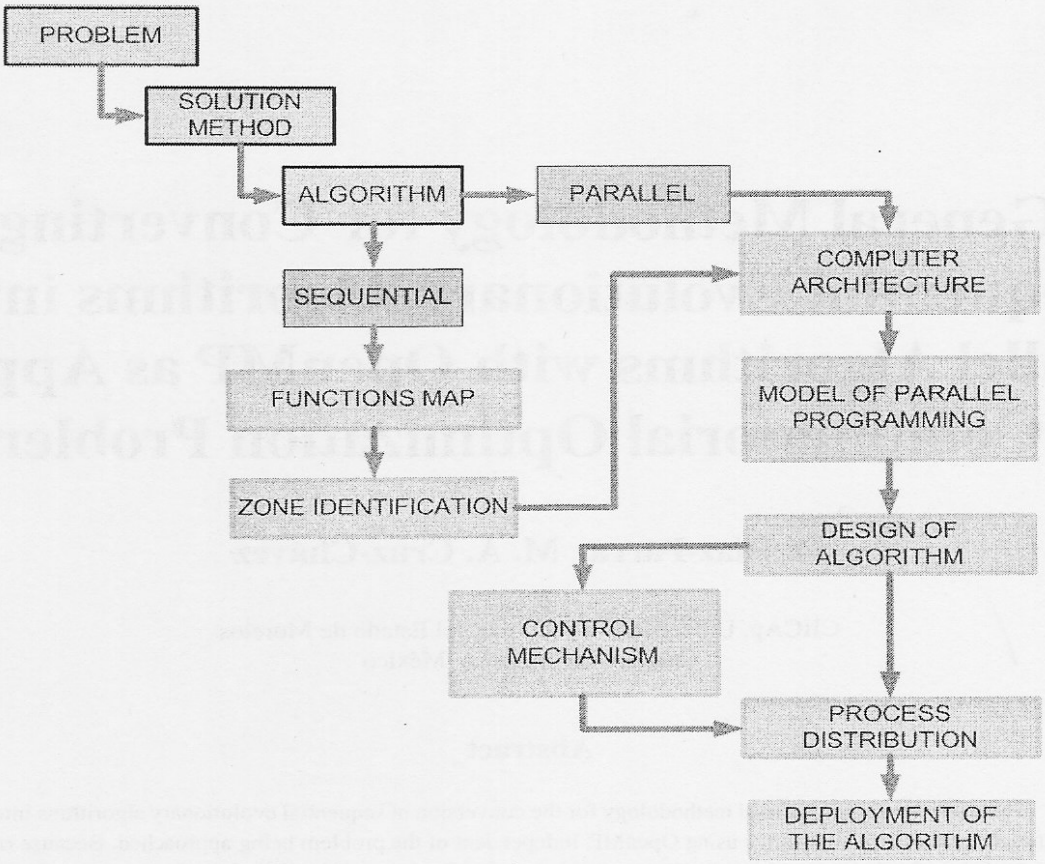


Fig. 1. Algorithm Conversion Methodology.

(VRPTW). It is necessary to take into account that the genetic algorithm is a heuristic algorithm and so it is difficult to parallelize [3]. Cantu-Paz [4] classify parallel genetic algorithms into three categories: a) global parallelization, which consists of carrying out iterations of the search method with the primary intention of reducing the run time of the method, not obtaining greater exploration of solutions, b) coarse-grained, and c) fine- grained. The last two classifications are defined according to the population size.

The steps suggested by Corona and Moreno [5] to parallelize a genetic algorithm are: to identify the components in which a significant computational gain can be obtained (for example: evaluation of the neighboring solutions in the landscape); to make a hierarchic scheme with cooperative techniques searching multithreads to improve the time, solution quality, computational efficiency, and the robustness of the search; to define the control functions of the processes to parallelize. In section two, this paper show the general methodology for the conversion of a sequential genetic algorithm to a parallel genetic algorithm. The methodology applied is the one proposed by the authors of this paper. In section three the general methodology for conversion is applied to the GA-VRPTW sequential genetic algorithm. Section four includes the conclusions drawn.

General Methodology for the Conversion

Work has been done on parallel algorithms for different types of problems in order to diminish the run time on the solution search, as mentioned in section one of this work. Nevertheless, because of the diversity of existing problems and the variation between each one, it is difficult to find a standard or a method to parallelize problems. Only specific analyses for the specific problem being approached exist, but there is not a general method. This paper proposes a general methodology for the conversion of a sequential algorithm into a parallel algorithm, independent of the problem being approached. For most problems there is: a model (implied definition of the problem, restrictions, and variables), a solution method (depending on the problem) and a type of algorithm that better solves it (deterministic, nondeterministic). For combinatorial optimization problems, for example, the objective is to find a solution to difficult problems to solve, by applying nondeterministic algorithms that approximate as closely as possible the optimal solution and require a considerable amount computational time. This activity implies the analysis of the run times of the proposed algorithm. Depending on the input instance, it is possible to determine whether it is computable or non-computable. Figure 1, shows the proposed

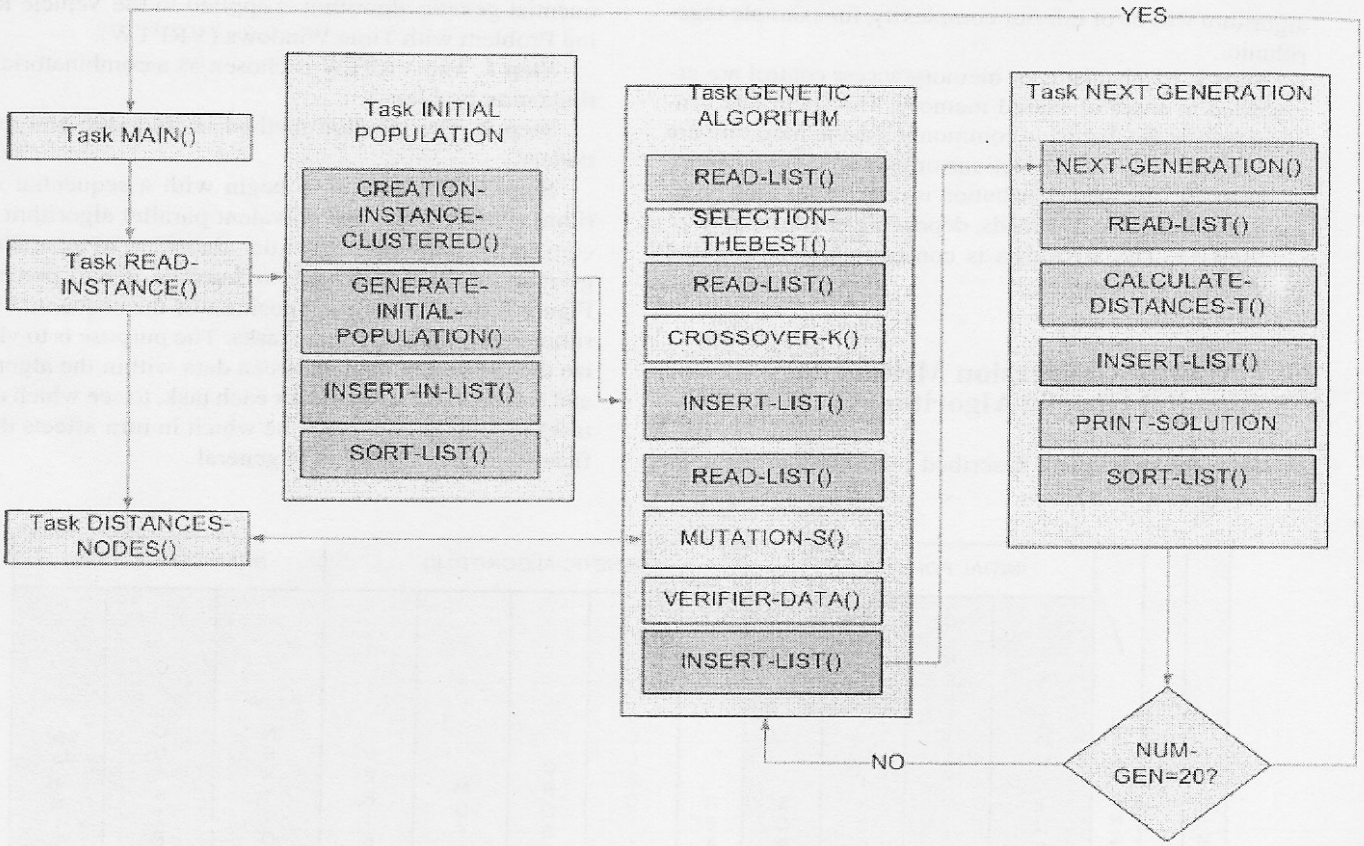


Fig. 2. Decomposition of the sequential genetic algorithm for the problem VRPTW into main tasks.

conversion methodology which consists of three main blocks: the identification of the problem, the solution technique and the algorithm.

Step 1. The problem block refers to the type of the problem to be solved.

Step 2. The solution block refers to the analysis of the construction of the solution based on the type of problem that is being analyzed.

Step 3. The algorithm block shows three options: first, starting with a sequential algorithm and constructing its equivalent parallel; second, starting with a parallel algorithm and finding a solution; third, improving an existing parallel algorithm.

Step 4. The sequential solution algorithm is analyzed and a functions map is constructed to show clearly the interaction of data between functions within the sequential algorithm and to identify dependent versus independent functions implied in the algorithm. By running the sequential algorithm, run times for each of the functions are assigned (dependent or independent).

Step 5. The zone identification is completed by detecting the functions with longer run time.

Step 6. Once the analysis of the sequential algorithm is finished, the computer architecture which will parallelize is examined. According to the Flynn [6] taxonomy, there are two main branches: a) the computers based on number of processors and number of programs that are executed and b) the computers based on the structure of the memory. The first branch is classified in four types: SISD (Simple Instruction Simple Data), SIMD (Simple Instruction Multiple Data), MISD (Multiple Instruction Simple Data), MIMD (Multiple Instruction Multiple Data). The second branch is classified in two types, shared memory and distributed memory. Shared memory has two types of access: uniform access to memory, which involves identical processors SMP (Symmetric Multi Processing) and non uniform access, which involves different access to memory for each processor. In distributed memory, each single processor has access to its own memory and the communication between processors is by passing messages MPI (Message Passing Interface).

Step 7. The model of parallel programming to be used is chosen. There are basically five models: shared memory, threads, MPI, data parallelization and hybrid parallelization. The choice is made depending on the equipment that will be used.

Step 8. The algorithm is designed, taking into account the main cycles that will conform it. It is important to mention that when starting with a sequential algorithm with polynomial complexity, the search with the parallel

algorithm will be of inferior complexity, for example logarithmic.

Step 9. Mechanisms of memory access control are established in cases of shared memory. The control or synchronization mechanisms commonly used in literature are the semaphores, critical sections or passage of messages

Step 10. The load distribution is realized for each process in processors or in threads, depending on the case.

Step 11. The algorithm is constructed using parallel programming languages.

Applied Conversion Methodology to Sequential Genetic Algorithm GA-VRPTW

Here the previously described methodology for a sequential genetic algorithm is applied to the Vehicle Routing Problem with Time Windows (VRPTW).

Step 1. The VRPTW is chosen as a combinatorial optimization problem.

Step 2. The solution method is Heuristic and evolutionary.

Step 3. The goal is to begin with a sequential algorithm and construct its equivalent parallel algorithm. The conversion is from a sequential algorithm to a parallel algorithm, so it is necessary to separate it into processes. Figure 2 shows the decomposition of the sequential algorithm in its respective main tasks. The purpose is to visualize the communication between data within the algorithm and establish the run time for each task, to see which of the tasks require greater run time which in turn affects the efficiency of the algorithm in general.

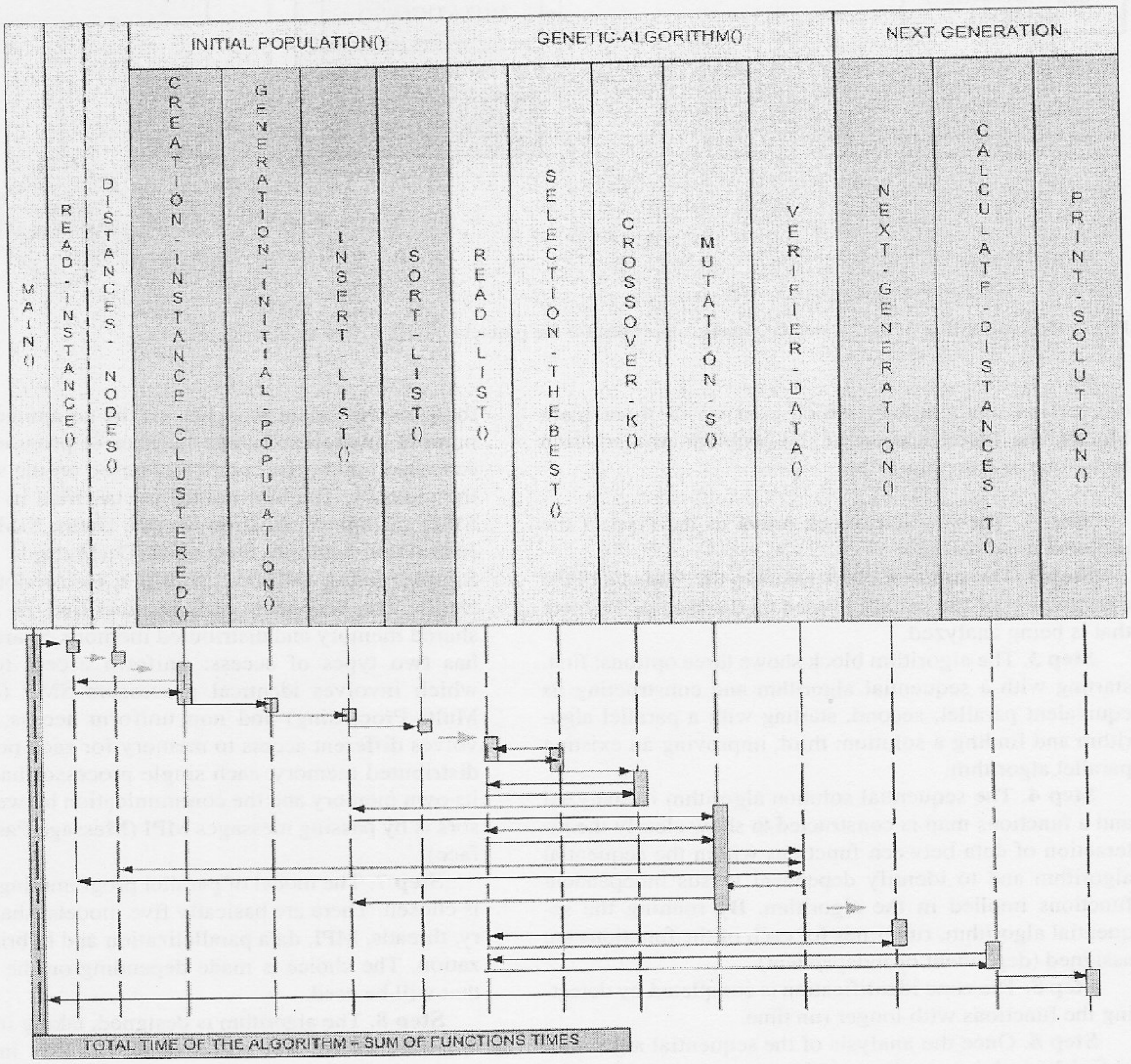


Fig. 3. Map of functions of the sequential genetic algorithm for VRPTW.

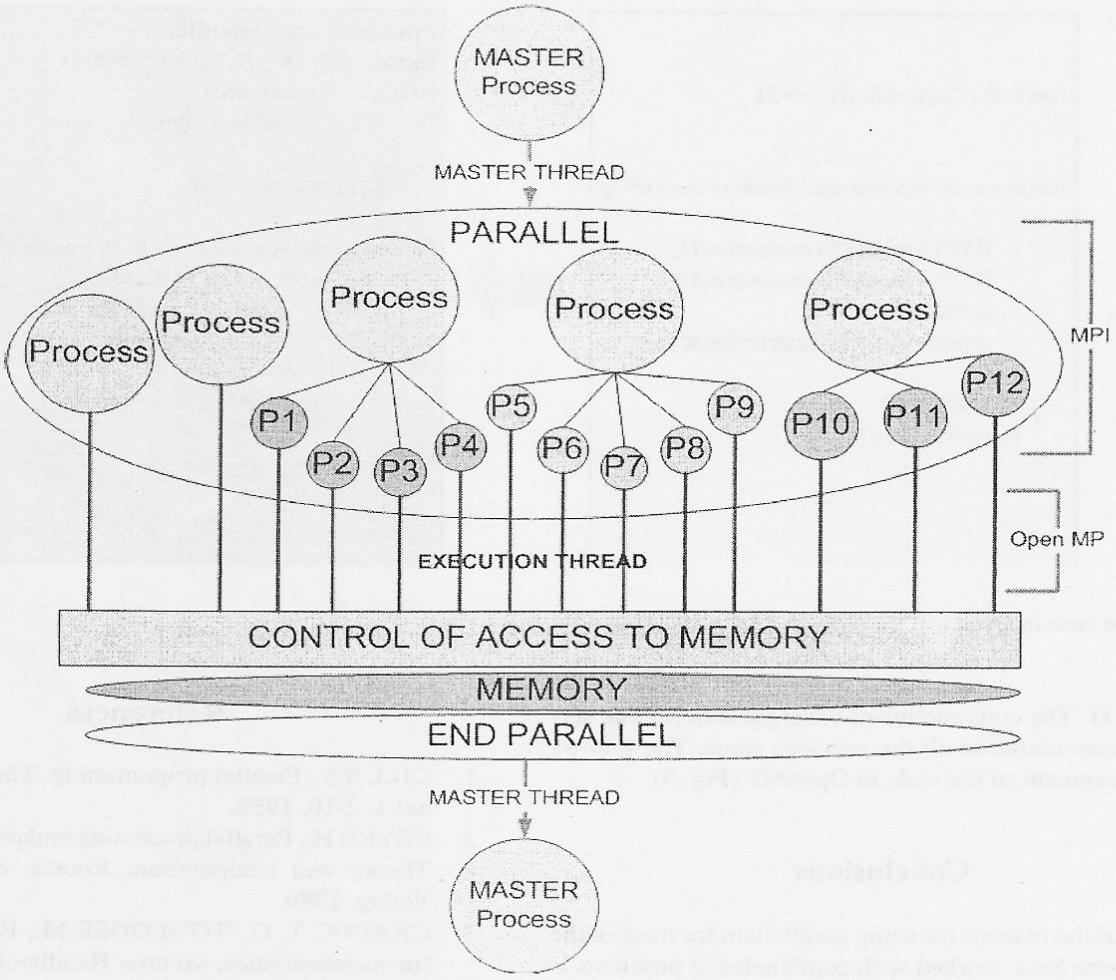


Fig. 4. Design and process load distribution of the sequential genetic algorithm for VRPTW .

Step 4. After detecting the tasks that consume greater run times, the map of functions is made, which is shown in Figure 3.

Step 5. The zone identification consists of analyzing the processes of the tasks that require more time to carry out. Those processes are candidates for parallelization or are the identified zones. For example, this can be seen in the mutation function that is presented in Figure 3.

Step 6. The architecture of the computer is an IBM Series 690 Parallel Supercomputer.

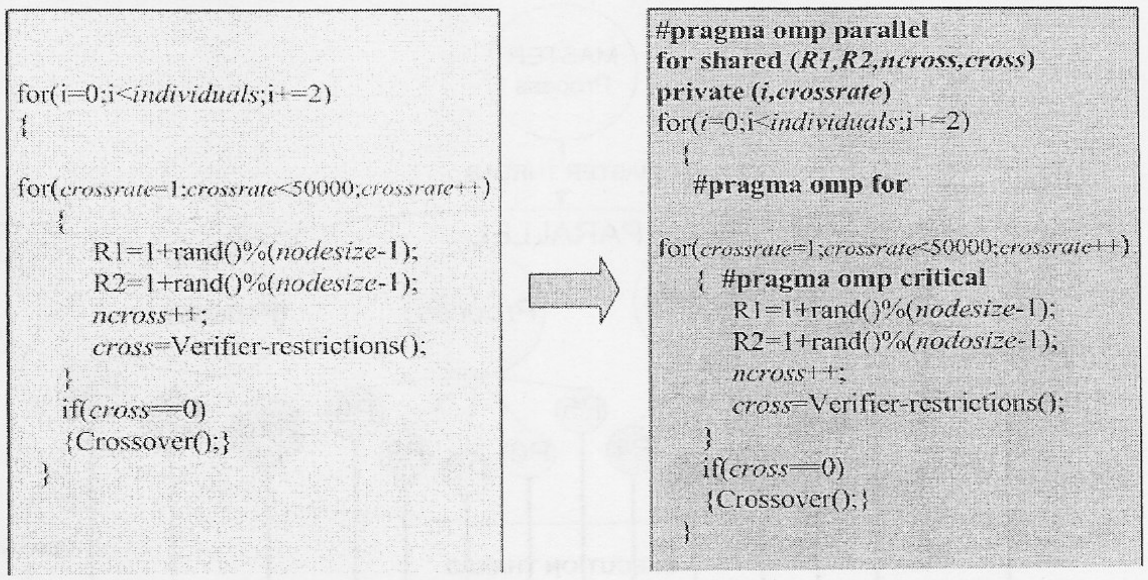
IBM p690 with 32 processors, 1.3GHz, and 32GB RAM. This equipment supports directives based on shared memory as in Open MultiProcessing (OpenMP) and directives based on distributed memory or Message Passing Interface (MPI).

Step 7. According to the structure of the equipment, the programming model can be hybrid. That means that some blocks of the algorithm work with shared memory and other blocks work with distributed memory.

Step 8. The design of the parallel algorithm and its process load distribution will depend much on the style of the programmer and the environment of programming used. Different options of parallel programming exist, using a specialized programming language, using an existing sequential programming language with adapted constructors to establish parallelism, or using an existing sequential programming language and library of procedures to parallelize. Some parallel programming languages are: OpenMP, HPF (High Performance Fortran), jade, and Parallel APL, among others.

Step 9. Because the programming model can be hybrid when it works with shared memory, it is necessary to establish control mechanisms for the access to memory, according to the style of the programmer by means of semaphore, critical section or passage of messages. In this case, the work was done with critical sections using the thread model combined with the passage of messages.

Step 10. The process load distribution can be done by functional decomposition or data decomposition. In this case processes with threads were used, so the distribution was by data decomposition (Fig. 4).



```
for(i=0;i<individuals;i+=2)
{

for(crossrate=1;crossrate<50000;crossrate++)
    {
        R1=1+rand()%(nodesize-1);
        R2=1+rand()%(nodesize-1);
        ncross++;
        cross=Verifier-restrictions();
    }
    if(cross==0)
    {Crossover();}
}
```

```
#pragma omp parallel
for shared (R1,R2,ncross,cross)
private (i,crossrate)
for(i=0;i<individuals;i+=2)
  {
   #pragma omp for

for(crossrate=1;crossrate<50000;crossrate++)
    { #pragma omp critical
        R1=1+rand()%(nodesize-1);
        R2=1+rand()%(nodosize-1);
        ncross++;
        cross=Verifier-restrictions();
    }
    if(cross==0)
    {Crossover();}
  }
```

Fig. 5. The code in Open MP.

Step 11. The construction of the algorithm is done taking into consideration all the previous steps. The following is a fragment of the code in OpenMP (Fig. 5).

Conclusions

One of the reasons for using parallelism for most of the authors who have worked with combinatorial problems is to optimize the time it takes the algorithm to find a solution. The time for a parallel algorithm does not depend on the number of processors used, but on the granularity of each process. The proposed methodology in this paper is a guide for parallelizing algorithms. Its purpose is to provide an auxiliary tool in the process of conversion or construction of parallel algorithms. This methodology, applied to the sequential genetic algorithm, generated information on how to design the parallel genetic algorithm. Future work will show the experimental results of the parallel genetic algorithm.

References

1. GILL S S., Parallel programming. The computer journal,1: 2-10, **1958**.
2. SEYED H., Parallel processing and parallel algorithm. Theory and computation. Roosta, editor. Springer-Verlag, **1999**.
3. CRAINIC T. G., TOULOUSE M., Parallel strategies for metaheuristics, volume Handbook of metaheuristics, pp 475-513. Kluwer academic publisher, **2003**.
4. CANTU-PAZ E., A summary of parallel genetic algorithms. Calculateurs Parallels Reseaux et Systemes Repartis, **10**:141-170, **1998**.
5. CRUZ CORONA C., MORENO J. M., Estrategias cooperativas paralelas en la solución de problemas de optimización, Revista Iberoamericana de Inteligencia Artificial, ISSN 1137 3601, No. **34**, Granada, Espana, pp. 129-143, **2007**.
6. FLYNN M. J., RUDD K. W., Parallel architectures. ACM computing surveys, **28**(1): 67-70, **1996**.