SCHEDULING MULTIPROCESSOR TASKS WITH GENETIC ALGORITHMS

MARIN GOLUB
Department of Electronics, Microelectronics, Computer and Intelligent Systems
Faculty of Electrical Engineering and Computing
University of Zagreb
Unska 3,HR-10000 Zagreb, Croatia
Phone:(+385-1) 6129 967 E-mail: marin.golub@fer.hr,

SUAD KASAPOVIC
Department of Telecommunications
Faculty of Electrical Engineering and Computing
University of Zagreb
Unska 3,HR-10000 Zagreb, Croatia
Phone:(+387-35) 760019 E-mail: casse@inet.ba

ABSTRACT – In this paper, an efficient method based on genetic algorithms is developed to solve the multiprocessor scheduling problem. To efficiently execute programs in parallel on multiprocessor scheduling problem must be solved to determine the assignment of tasks to the processors and the execution order of the tasks so that the execution time is minimized. Even when the target processors is fully connected and no communication delay is considered among tasks in the task graph the scheduling problem is NP-complete. Complexity of scheduling problems dependent of number of processors *(P)*, task processing time T_i and precedence constraints. This problem has been known as strong NP-hard intractable optimisation problem when it assumes arbitrary number of processors, arbitrary task processing time and arbitrary precedence constraints. We assumed fixed number of processors and tasks are represented by a directed acyclic graph (DAG) called "task graph".

Keywords – DAG, parallel processing, multiprocessor scheduling, genetic algorithms, optimisation, heuristics.

I. INTRODUCTION

The telecommunication networks are one of good example parallel and distributed systems and can be considered as parallel call and service processing systems where scheduling tasks in that system can be done with genetic algorithms. A major challenge in parallel processing is task scheduling. In the static scheduling a parallel program, is represented by a directed acyclic graph (DAG). DAG consists of:

- a collection of "vertices" $V=\{T_i\}$, $i=1,2,...,N$, and where V is a set of v nodes, representing the tasks;
- a collection of "edges", $E=\{e_{ij}\}$, each connecting some two vertices and a directed, representing the precedence relationship among the computational tasks. An edge goes from one of the vertices towards the other.

Here we can draw a graph in which a directed edge says we have to do one task before the other. Let us assume that the task T_j cannot execute until T_i completes if $i<j$. This problem can be reformulated as graph problems. The general problem of multiprocessor scheduling can be stated as scheduling a set of partially ordered computational tasks onto a multiprocessor system so that a set of performance criteria will be optimised. Task scheduling is the process of deciding which instructions will be run by which processor, and in which order. We assume that the multiprocessor system is uniform (homogenous) and nonpreemptive etc. the processors are identical, and a processor completes the current task before executing a new one. Task execution time can be nonuniform. Every processor can communicate with each other and every has own memory. In this paper we deal with the scheduling problem where number of processors is fixed.

Other problem which can be discussed is find the minimum number of processors in order to execute tasks in a time not exceeding the length of the critical path of task graph. Complexity of scheduling problems dependent of number of processors *(P)*, task processing time T_i and precedence constraints. Example, when number of processors is arbitrary, task processing time is equal and precedence constraints is tree complexity this problem is $O(N)$ where N is number of computational tasks or example, when number of processors is arbitrary, task processing time equal and precedence constraints arbitrary complexity this problem is NP-hard, which is evidence that there doesn't exist a good algorithm to solve it exactly [10]. Consequently, research effort has focused on finding polynomial time algorithms, which produce optimal schedules for restricted cases of the multiprocessor scheduling problem. However we can find a solve that's pretty close to the optimal solution. This paper presents an efficient method based on genetic algorithms to solve multiprocessor scheduling problem.

This paper is organized as follows. First we present the model for multiprocessor scheduling (Section 2), after that the principles of genetic algorithms are precisely defined in Section 3. In Section 4, our proposed genetic algorithm is reviewed and analysed. Results and analyses are shown in Section 5 and we close the paper with concluding remarks and ways for further research – Section 6.

II. THE MULTIPROCESSOR SCHEDULING PROBLEM

The number of available processors is unlimited. A multiprocessor system is composed of a set *P* of *p* identical processors (homogeneous), that is, their processing speeds are the same. The processors are fully connected without any regard to link contention and nonpreemptive, that is, each processor can execute at most one task at a time. A parallel program is characterized by a directed acyclic (task) graph $G=(V, E)$ where $V=\{T_i\}$, $i=1,2, \ldots,N$ represent the set of tasks of the program with associated weights $\{r_i\}$ where r_i denotes the computation (execution) time of T_i, and $E=\{e_{ij}\}$ is the set of directed edges which define a partial order or precedence constraints on *V*. In our case of task graphs, intertask communication is negligible etc. the communication cost between two tasks assigned to the same processor is assumed to be zero. In task graph, tasks without predecessors are known as entry tasks and tasks without successors are known as exit tasks. Every task is present and appears only once in the schedule (completeness and uniqueness). The problem of optimal scheduling a task graph onto a multiprocessor system with p identical processors is to assign the computational tasks to the processors so that all of the tasks are completed in the shortest possible time. The time that the last task is completed is called the finishing time (FT) of the schedule. A simple task graph with 10 tasks and a schedule is illustrated in Fig.1.

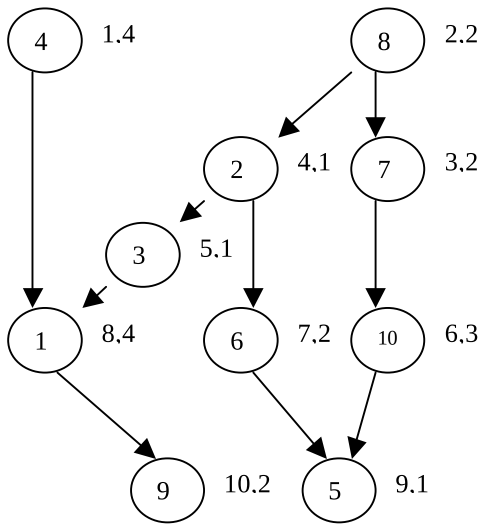

Fig.1. Example of directed acyclic graph with execution time and without communicating cost

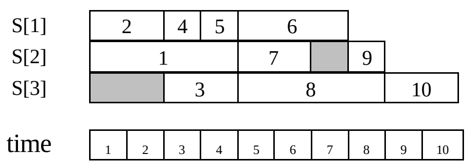

Fig.2. Example of optimal scheduling with precedence constraints.

Inside of circle is the marks of tasks and from the right side of circle are first indexes getting by topological sorting and after comma execution time of task.

III. EVOLUTION AND GENETIC ALGORITHM

Evolutionary algorithms are optimisation and search procedures inspired by genetics and the process of natural selection. This form of search evolves throughout generations improving the features of potential solutions by means of biologically inspired operations. On the ground of the structures undergoing optimisation the reproduction strategies, the genetic operators adopted, evolutionary algorithms can be grouped in: evolutionary programming, evolution strategies, classifier systems, genetic algorithms and genetic programming.

The genetic algorithms behave much like biological genetics. The genetic algorithms are an attractive class of computational models that mimic natural evaluation to solve problems in a wide variety of domains. A genetic algorithm comprises a set of individual elements (the population size) and a set of biologically inspired operators defined over the population itself etc. a genetic algorithms manipulate a population of potential solutions to an optimisation (or search) problem and use probabilistic transition rules. According to evolutionary theories, only the most suited elements in a population are likely to survive and generate offspring thus transmitting their biological heredity to new generations. A genetic algorithm maps a problem onto a set of strings (the chromosomes), each string representing a potential solution. The three most important aspects of using genetic algorithms are: (1) definition of the objective function, (2) definition and implementation of the genetic representation, and (3) definition and implementation of the genetic operators.

Table 1. Nature-to-computer mapping

Nature	*Computer*
Individual	Solution to a problem
Population	Set of solutions
Fitness	Quality of a solution
Chromosome	Representation for a solution
	(e.g. set of parameters)
Gene	Part of the representation of a solution
	(e.g. parameter or degree of freedom)
	Decoding of the representation of
	solutions

There are a lot of list heuristic methods which using to scheduling tasks onto parallel processors. Most of them give a good solution problem. Example, each task graph is assigned a priority, then added to a list of waiting tasks in order of decreasing priority. As processors become available the task with the highest priority is selected from the list and assigned to the most suited processor. If more than one task has the same priority a task is selected randomly. The basic list scheduling heuristic is shown in pseudocode :

```
begin
  repeat
    select a task
    select a processor to run the task
    assign the task to the processor
  until all task s are scheduled
end.
```

Fig.3.List scheduling heuristic

Initialisation - an initial population of the search nodes is randomly generated. The strings encoding mechanism should map each solution to a unique string. The encoding mechanism depends on the nature of the problem variables and it use for representing the optimisation problem's variables. The representation is unique. In some cases the variables assume continuous values, while in other cases the variables are binary. It can be integer parameters, real-valued parameters, vectors of parameters, Gray code, dynamic parameter encoding etc. The fitness values of each node are calculated according to the fitness function (objective function). The fitness function provides the mechanism for evaluating each chromosome in the problem domain. It is always positive. Three operators are needed to achieve this *selection, crossover and mutation.* The *selection* criterion is that string with higher fitness value should have a higher chance of surviving to the next generation. A quality measure for the solutions (fitness function) of the problem is known. Fitter solutions survive, while weaker ones perish. There are many different models of selection. The most popular selection in genetic algorithms is fitness proportionate selection, rank selection, tournament selection and elitist selection. After selection comes crossover.

The *crossover* operator takes two chromosomes (parents) and swaps part of their genetic information to produce new chromosomes (child). The offspring (child) keep some of the characteristics of the parents. One point crossover involves cutting the chromosomes of the parents at a randomly chosen common point and exchanging the right - hand – side sub-chromosomes. In two – point crossover chromosomes are thought of as rings with the last and the first gene connected. The rings are cut in two sites and the resulting sub-parts are exchanged. In uniform crossover each gene of the offspring is selected randomly from the corresponding genes of the parents. Crossover is applied to the individuals of a population with a constant probability. Usually from 0.5 to 0.95.

Mutation consists of making (usually small) alterations to the values of one or more genes in a chromosome. In genetic algorithms, mutation is considered a method to recover lost genetic material. Our proposed algorithm we call Turnir genetic algorithm. When we have task graph, we need use topological sorting. Topological sorting consist of finding some global ordering consistent with these local constraints.

IV. PROPOSED ALGORITHM

Let be *G(V,E)* directed acyclic graph shown by list. Task graph is topological sortie if *index (I)<index (J)* if the node T_i is predecessor from node *Tj* . If the number of nodes (tasks) is *N* them index can be the natural number from *[1..N]*. Include order *Number [1..N]* and order *R* of index of tasks. Algorithm of topological sorting by width looking as follows (Algorithm how we can give to the tasks indexes by width):

```
for(I=1;I<= N;I++) {
  Number[I]:=Number_incoming_edges[I];
  if(Number[I]==0) put_in_order(R,I);
  J:=1
  While R nonempty {
    L=first_index_taken_from_order(R);
    Index[L]:=J
    J:=J+1;
    for all K from list_outgoing_edges[L]{
    Number[K]:=number[K]-1;
    if(Number[K]==0){put_in_order(R,K);}
  }
}
```

Fig.4. Topological sorting by width

When we got indexes of tasks, now, structure of our genetic algorithm look as follows:

```
genetic algorithm {
  initialisation(P(0));
  for(i=0,i<I;i++) {//I-number of iterations
    randomly choose three individuals from
      P(i);
    mark the worst  individuals for
      elimination;
    crossover surviving individuals;
    mutation child;
    evaluation child;
    replace eliminated individuals with new
      child
  }
}
```

Fig.5 . Proposed genetic algorithm

Now, we will consider initialisation, crossover, mutation and evaluation algorithm.

With initialisation we will make population of solutions. Let be *N* population size and *Z* will be number of tasks from directed acyclic graph. Randomly we choose the one of processors from set of *[1,P]* where *P* is total number of processors and then add the task from the list of task sortie by indexes on increasing order. Pseudocode of initialisation is:

```
initialisation(P(0)) {
  for (i=0;i<N;i++){//N-population size
  for(j=1;j<=Z;j++){//Z-number of tasks
    set of tasks which do r-th processor
      extend with task $T_j$;
      //r-randomly chosen number: r∈[1,P]
      //P-number of processors
  }
  evaluation(i-th individual);
}
```

Fig.6. Pseudocode of initialisation

The chromosome is consisting from *P* sorted arrays. Example, let be *P*=3 and *Z*=10. One example of string looking as:

S[1]	2	5	6		
S[2]	1	4	7	8	9
S[3]	3	10			

Fig.7. Example of string (work with indexes (sorted by width) of tasks)

This is only chromosome (string) *but not scheduling*. On that way, and with number of iterations we have defined algorithm of initialisation.

The crossover operator use two strings randomly choose (choose one at random task) T_i from one of two sets and put on the new string. Precedence relation must be kept and whole time we work with indexes of tasks (getting by topological sorting). If T_i element the same set at both parents then T_i is coping on the same place for new string (child). If we randomly choose two same parents (strings A=B) and if one of parents e.g. A the best string we use operator mutation on the second B string. It is elitism. Else, we mutate the first set and child generate randomly.
This algorithm performs the crossover operation on two strings (A and B) and generates new string.

```
crossover (A;B) {
  if(A==B){ //elimination duplicate
    if(A is the best string) mutation(B);
                                // elitism
    else mutation(A);
    random generate child;
    return
  }
  for(i=1,i<=Z;i++){
    if($T_i$∈P on both parents)
      $T_i$ copy on the same place on the child
    else
      $T_i$ below one from sets of parents
                                (randomly);
  }
}
```

Fig.8. Pseudocode of crossover

Increasing order indexes of the tasks must be kept.

Before crossover operation:

A[1]	2	5	6		
A[2]	1	4	7	8	9
A[3]	3	10			

B[1]	1	5			
B[2]	3	4	8		
B[3]	2	6	7	9	10

After crossover operation (child):

C[1]	1	5			
C[2]	3	4	8	9	
C[3]	2	6	7	10	

Fig.9. Example of crossover operator (work only with indexes)

Next, what we must to do is define the mutation operator. We first generate two randomly chosen numbers *r* i *q* from the sets *[1,P]*. The condition for that is that: a) *r # q*, and b) set *r* aren't empty

After that from set *r*, choose one task at random and remove him in the set *q*. We must take in the account that the task which we move, must be put on the place that indexes of task be ordered by increasing.
Algorithm mutation:

```
mutation() {
   generate  two numbers randomly r,q∈[1,P]
   with conditions:
     a)r # q, and
     b)set r aren't empty
   from set r, randomly pick  a task and
   removing him in the set q;
}
```

Fig.10. Pseudocode of mutation

Let be r=1 and q=2 and randomly choose the task has the index 5.

Before mutation:

S[1]	2	5	6			
S[2]	1	4	7	8	9	
S[3]	3	10				

After mutation:

S[1]	2	6				
S[2]	1	4	5	7	8	9
S[3]	3	10				

Fig.11. Example of mutation operator

Figure 12 shows the evaluation algorithm.

```
evaluation(){
  for(i=1;i<=P;i++) FTP[i]=0;
      // reset FTP for all processors
  for(i=1;i<=Z;i++){
        // $T_i \in$ {sets of tasks processor p},
           $p \in [1,P]$
    FTP[p] += duration($T_i$);
    for(j=1;j<=P;j++){
      if(j == p) continue;
      // the precedence relations are
      // maintained in this line:
      if((($T_x \in$ j)<$T_i$ ) && (FTP[j]>pom))
      // x is the biggest index of tasks set
      // p (tasks which execution onto
      // processor p), and that content the
      // condition  x<j.
         FTP[p] = FTP[j] + $T_i$;
    }
  }
  FT=max$_{i \in [1,P]}${FTP[i]};
}
```

Fig.12. Pseudocode of evaluation

The fitness function for the multiprocessor scheduling problem in our genetic algorithms is finishing time a besides it can be also throughput and processor utilization. Finishing time of a schedule is defined as follows: $FT=max_{i\,î[1,P]}\{FTP[i]\}$ where *FTP[i]* is the finishing time for the last task in processor *i*.

V. RESULTS

The experiments have been done on two computers:

a) SUN ULTRA on 140 MHz with 64 MB RAM, and
b) ALPHA STATION 600 on 333 MHz and 256 MB RAM

Test problem consists in the work with two problems: Problem with 452 tasks, which should be scheduling onto 20 processors. Number of tasks generated randomly as processing times for every task and the precedence constraints took from [12]. For this problem, optimal solution is 537 time units. And it has been also taken from [12]. To that solution we have done with our GA with next parameters: mutation probability 1%, population size is 20 individual solutions and it has 10000 iterations with 85% probability. We have done a few tens experiments. In the 15% cases we have got local optimum. The result of 540 time units doesn't look as bad solution because it's only three time units worst solution from optimal solution from the site [12]. This problem, on the computer b), for 1000 iterations spent 7 minutes and 40 seconds. We have graphically showed evolution process for first problem (Fig.13).

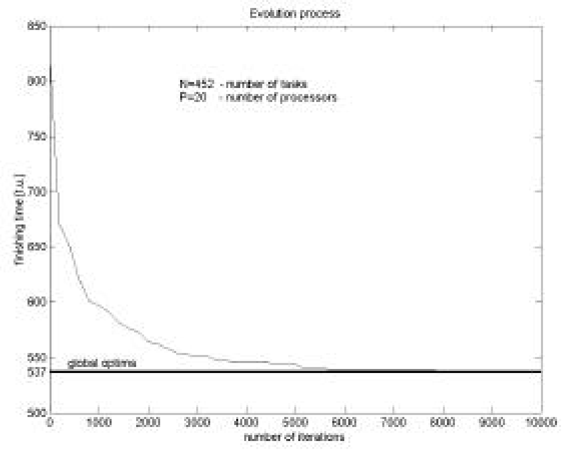

Fig.13. Evolution process

Second, we tested the problem with 473 tasks, which should be scheduling onto 4 processors. This problem, for 1000 iterations on the computer b) spent 1 minute. Optimal solution for this problem is 1178 time units (from the site). The best solution with our genetic algorithm was 1182 time units and the worst solution was 1187 time units. Average solution with 1184 time units has got with the next parameters: population size=50, number of iterations=30000 and mutation probability=0.01 or 1%. For better results, it would be to adjust genetic algorithm and for that we need a few hundred experiments, which take a few months on this computers.
We measured the quality the schedules using a quantity called speedup where:
Speedup = (completion time of a task graph using sequential schedule) / (completion time of the task graph on multiprocessor according to scheduling algorithm). For the first problem with 452 tasks, total processing time is 4584 t.u., speedup = 8.54, until for the second problem with 473 tasks, total processing time is 4713 t.u., speedup = 3.98.

VI. CONCLUSION

In this paper we present an efficient genetic algorithms for scheduling precedence constrained task graphs with negligible intertask communication onto multiprocessors without taking contention in the communication channels into consideration. This means that for such problems, without limiting onto communication cost and contention of the communication channels has to be designed a different type of algorithm. Experimental results on the relatively hard problems that have been taken from Internet [12] show that genetic algorithm without optimisation of parameters comes to optimum or near optimum solutions. In order to get better results, concerning larger probability of the global optimum, an optimisation of the parameters genetic algorithm should be done. It is possible to make a parallel of the described genetic algorithm, in a very simple way, by using multithreading as described in [14,15,16].

VII. APPENDIX

Table 2 : Complexity of scheduling problems

Number of Processors (*m*)	Task Processing Time T*i*	Precedence Constraints	Complexity
Arbitrary	Equal	Tree	O (*n*)
2	Equal	Arbitrary	O (*n*^2)
Arbitrary	Equal	Arbitrary	NP-hard
Fixed (*m*>=2)	T*i*=1or2 for all *i*	Arbitrary	NP-hard
Arbitrary	*Arbitrary*	*Arbitrary*	*Strong NP-hard*

REFERENCES

[1] Keith Grant, An Introduction to Genetic algorithms, CC++ Journal, march 1995

[2] Greg P. Semeraro, Evolutionary Analysis Tools For Real-Time Systems, The sixth International symposium on Modeling, analysis and Simulation of Computer and Telecommunication Systems, Montreal, Canada, July 1998

[3] M.Srinivas, Lalit M. Patnaik, Genetic Algorithms: A Survey, IEEE, 1994

[4] Jose L.Ribeiro Filho, Philip C. Treleaven, Genetic Algorithm Programming Enviroments, IEEE, 1994

[5] Eduardo B. Fernandez, Bertram Bussell, Bounds on the Number of Processors and Time for Multiprocessor Optimal Schedules, IEEE Transactions on Computers, Vol. C-22, No.8, August 1973

[6] G.N. Srinivasa Prasanna , B.R. Musicus, Generalized Multiprocessor Scheduling and Applications to Matrix Computations, IEEE Transactions on Parallel and Distributed systems, Vol.7, No.6, June 1996

[7] Ishfaq Ahmad, Yu-Kwong Kwok, On Parallelizing the Multiprocessor Scheduling problem, IEEE Transactions on Parallel and Distributed systems, Vol.10, No.4, April 1999

[8] S. Selvakumar and C.Siva Ram Murthy, Scheduling Precedence Constrained Task Graphs with Non-Negligible Intertask Communication onto Multiprocessors, IEEE Transactions on Parallel and Distributed systems, Vol.5, No.3, March 1994

[9] Edwin S.H.Hou, Nirwan Ansari, Hong Ren, A Genetic Algorithm for Multiprocessor Scheduling, IEEE Transactions on Parallel and Distributed systems, Vol.5, No.2, February 1994.

[10] Ricardo C. Correa, Afonso Ferreira, Pascal Rebreyend, Scheduling Multiprocessor Task with Genetic Algorithm", IEEE Transactions on Parallel and Distributed systems, Vol.10, No.8, August 1999.

[11] Albert Y. Zomaya, Chris Ward, Ben Macey, Genetic Scheduling for Parallel Processor Systems: Comparative studies and Performance Issues, IEEE Transactions on Parallel and Distributed systems, Vol.10, No.8, August 1999.

[12] Advanced computing systems, available from: **http://www.kasahara.elec.waseda.ac.jp**

[13] Introduction to evolutionary computation, available from: **http://www.cs.bham.ac.uk/~rmp/slide_book/node2.html**

[14] M. Golub, D. Jakobovic, A New Model of Global Parallel Genetic Algorithm, Proceedings of the 22nd International Conference ITI2000, Pula, 2000, pp.363-368

[15] M. Golub, L. Budin, An Asynchronous Model of Global Parallel Genetic Algorithms, Second ICSC Symposium on Engineering of Intelligent Systems EIS2000, University of Paisley, Scotland, UK, 2000, pp. 353-359

[16] M. Golub, D. Jakobovic, L. Budin, Parallelization of Elimination Tournament Selection without Synchronization, Proceedings of 5th IEEE International Conference on Intelligent Engineering Systems