Distributed Scheduling with Decomposed Optimization Criterion: Genetic Programming Approach

Franciszek Seredyński[1,2], Jacek Koronacki[2] and Cezary Z. Janikow[1]

[1] Department of Mathematics and Computer Science
University of Missouri at St. Louis, St. Louis, MO 63121, USA
[2] Institute of Computer Science, Polish Academy of Sciences, Ordona 21, 01-237 Warsaw, Poland
e-mail: sered@arch.umsl.edu, korona@ipipan.waw.pl, janikow@radom.umsl.edu

Abstract. A new approach to develop parallel and distributed scheduling algorithms for multiprocessor systems is proposed. Its main innovation lies in evolving a decomposition of the global optimization criteria. For this purpose a program graph is interpreted as a multi-agent system. A game-theoretic model of interaction between agents is applied. Competetive coevolutionary genetic algorithm, termed loosely coupled genetic algorithm, is used to implement the multi-agent system. To make the algoritm trully distributed, decomposition of the global optimization criterion into local criteria is proposed. This decomposition is evolved with genetic programming. Results of succesive experimental study of the proposed algorithm are presented.

1 Introduction

In this paper, we propose a new approach to develop parallel and distributed algorithms for the problem of multiprocessor scheduling [1, 4]. Parallel and distributed scheduling algorithms represent a new direction in the area of multiprocessor scheduling [2, 8, 9]. Here we propose an approach based on a multi-agent system paradigm, which offers both parallelism and distributed control of executed algorithms. To apply a multi-agent system methodology we will interpret the graph of a parallel program as a multi-agent system, and we will investigate the global behavior of such a system in terms of minimization of the total execution time of the program graph on a given multiprocessor system. To implement such a multi-agent system, we will use competetive coevolutionary genetic algorithm termed *loosely coupled genetic algorithm* (LCGA) [8], which provides evolutionary algorithm for a local decision making, and a game-theoretic model of interaction between agents.

The specific open question in the area of multi-agent systems is the problem of incorporating a global goal of the multi-agent system into local interests of individual agents. We propose to apply an evolutionary technique, known as genetic programming (GP) [7], to decompose a global optimization criterion of the scheduling problem into local optimization criteria of agents.

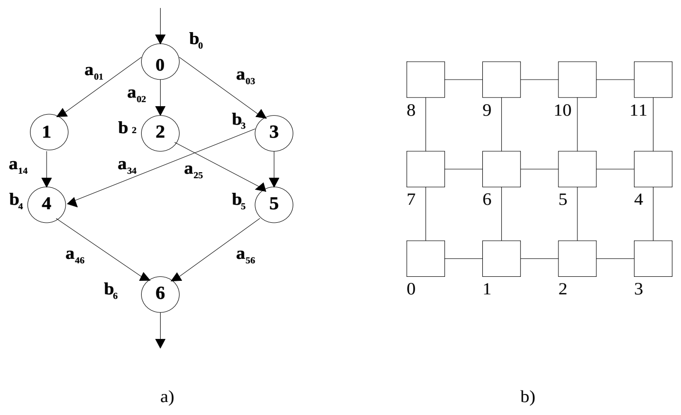

Fig. 1. Examples of a system graph (a), and a precedence task graph (b)

The paper is organized as follows. Section 2 discusses accepted models of parallel programs and systems in the context of scheduling problems, and proposes a multi-agent interpretation of the scheduling problem. Section 3 presents a parallel and distributed algorithm interpreted as a multi-agent system and implemented with LCGA. Section 4 describes an approach to decompose the global scheduling optimization criterion into local optimization criteria – based on GP. Results of experimental study of the proposed scheduler are presented in Section 5.

2 Multi-agent Approach to Multiprocessor Scheduling

A multiprocessor system is represented by an undirected unweighted graph $G_s = (V_s, E_s)$ called a *system graph*. V_s is the set of N_s nodes representing processors and E_s is the set of edges representing bidirectional channels between processors.

A parallel program is represented by a weighted directed acyclic graph $G_p =< V_p, E_p >$, called a *precedence task graph* or a *program graph*. V_p is the set of N_p nodes of the graph, representing elementary tasks. The weight b_k of the node k describes the processing time needed to execute task k on any processor of the homogeneous system. E_p is the set of edges of the precedence task graph describing the communication patterns between the tasks. The weight a_{kl}, associated with the edge (k, l), defines the communication time between the ordered pair of tasks k and l when they are located in neighboring processors. Figure 1 shows examples of the program graph and the system graph representing a homogeneous multiprocessor system with a grid topology.

The purpose of *scheduling* is to distribute the tasks among the processors in such a way that the precedence constraints are preserved, and the *response time* T (the total execution time) is minimized. The response time T depends on

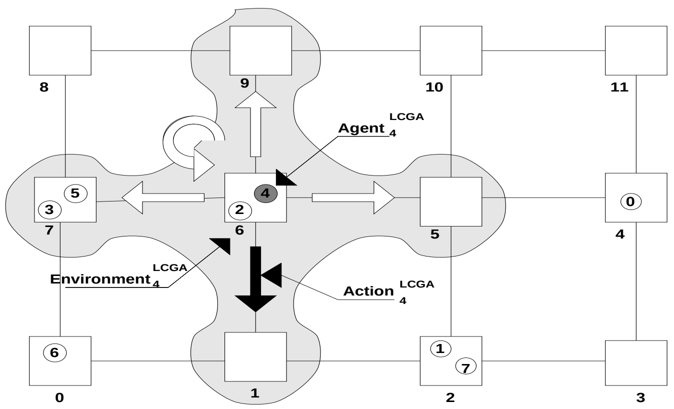

Fig. 2. Multi-agent interpretation of the scheduling problem.

the *allocation* of tasks in the multiprocessor topology and on *scheduling policy* applied in individual processors:

$$T = f(allocation, scheduling\_policy). \tag{1}$$

We assume that the scheduling policy is fixed for a given run and the same for all processors.

We propose an approach to multiprocessor scheduling based on a multi-agent interpretation of the parallel program. An interaction between agents is based on applying a game-theoretic model termed a game with limited interaction [8]. The agents migrate in the topology of the parallel system environment, searching for an optimal allocation of program tasks into the processors. To evolve strategies for for the migration of agents in the system graph, we use competetive coevolutionary genetic algorithm LCGA (see Section 3).

We assume that a collection of agents is assigned to tasks of the precedence task graph in a such way that one agent is assigned to one task. An agent A_k, associated with task k, can perform a number of migration actions S_k on the system graph (see Figure 2). These local actions change the task allocation, thus influence the global optimization criterion (1).

The scheduling algorithm can be described in the following way:

– Agents are assigned to tasks of the program graph and located on the system graph (e.g., randomly).
– Agents are considered as players taking part in a game with limited interaction. Each agent-player has $r_k + 1$ possible actions interpreted as migrations to r_k nearest neighbors (plus 1 for stying in place). Figure 2 shows the precedence task graph from Figure 1 allocated in the system graph of the same Figure 1.
– Each player has a global criterion (1) describing the total execution time of the scheduling problem.

– The objective of players in the game is to alocate their corresponding tasks to processors in such a way that the global execution time is minimized (1).
– After a predefined number of single games, players are expected to reach an equilibrium state corresponding to a Nash point - a solution of this noncooperative game, providing the maximal payoff of the team of players in the game.

3 Distributed Scheduling with LCGA

The LCGA [8] implementing the parallel and distributed algorithm interpreted as a multi-agent system can be described in the following way:

#1: For each agent-player A_k, associated with task k and initially allocated in node n of the multiprocessor system, create an initial subpopulation of size N with its actions. S_k^n defines the chromosome phenotype, as each chromosome is randonly initialized with an action from S_k^n.
#2: Play a single game
- in a discrete moment of time, each player randomly selects one action from the set of actions represented in its subpopulation and not used until now
- calculate the output of the game: each player evaluates its payoff in the game; the payoff corresponds to the value of the criterion (1)

#3: Repeat step #2 until N games are played.
#4: Use GA operators on individual subpopulations
- after playing N individual games, each player knows the value of its payoff received for each action from its subpopulation,
- these payoff values become fitness values for GA; genetic operators [6] of selection (S), crossover (Cr) and mutation (M) are applied locally to subpopulations of actions; the new subpopulations of actions will be used by players in games played in the next game horizon.

#5: Return to step #2 until the termination condition is satisfied.

Potential possibilities of this multi-agent based scheduler are not fully explored because of the nature of the global criterion (1). To make the algorithm fully distributed, decomposition of the global criterion (1) into local interests of agents is proposed in the next section. This decomposition will be accomplished with use of genetic programming.

4 Decomposing Global Function with Genetic Programming

4.1 Genetic Programming

Genetic programming [7] was proposed as an evolutionary method for evolving solutions represented as computer programs. Its main innovation lies in variable-length, tree-like representation. For a particular problem, sets of *functions* and *terminals* are determined. They define labels for the nodes of the trees – terminals can labels leaves, and functions can label nodes according to their arities. Initial structures (programs), with randomly generated labels, are created. These undergo silumated evolution, with the same Darwinian selection, and with tree-based mutation and crossover. The evolutionary process is continued until either a solution is found or the maximum number of generations is reached.

4.2 Genetic Programming Environment

To design a fully distributed scheduling algorithm we need to decompose the global scheduling criterion (1) into local criteria of individual agents of the multi-agent based scheduler. This means that we need to express (1) as

$$T \approx g(h_0(X_{env_0}, sp), h_1(X_{env_1}, sp), ..., h_k(X_{env_k}, sp), ..., h_{N-1}(X_{env_{N-1}}, sp)), \quad (2)$$

where X_{env_k} is the set of local variables of agent A_k, defined in its local environment env_k, sp is the homogeneous scheduling policy, $h()$ is the local scheduling criterion, and $g()$ is some composition of functions $h()$, approximating the criterion (1).

To simplify a computational complexity of the problem we will assume that

$$h_0() = h_1() =, ... = h_k() = ... = h_{N-1}() = h(). \quad (3)$$

We will assign agent A_k^{GP} to task k, which will execute in some local environment env_k.

4.3 Terminals and Functions

The set of *terminals* is composed of

- **StartTime** - beginning time of execution of the visited task.
- **MrtTime** - a Message Ready Time, i.e. time of receiving last data from some predecessor, by a task actually visited by A_k^{GP}.
- **ExecTime** - a computational time b_l (see Section 2) of a task actually visited by agent A_k^{GP}.
- **PredStartTime** - the sum of time-starts of execution of predecessors of a task actually visited by an agent A_k^{GP}.
- **SuccStartTime** - the sum of the time-starts of successors of a task actually visited by an agent A_k^{GP}.

The set of *functions* contains the following function:

- **add**, **sub** - two argument function returning a sum, a difference of argument, respectively.
- **min**, **max** - two argument function returning smaller, greater value from the set of arguments, respectively.

- **forward** - one argument function which causes moving an agent from a current position at the actually visited node (task), corresponding to some incomiming or outcoming edge, to a node adjacent by this edge.
- **right**, **left** - one argument function which causes turning right, left respectively, from an actual position corresponding to some incoming or outcoming edge of the currently visited node, to the next edge of the precedence task graph.
- **ifMRTNodeAhead** - two argument function which returns the value of the first argument if the agent currently visiting a given node is in a position corresponding to some edge pointing a Message Ready Time node (MRT Node), i.e. the node which is last one from which the visited task receives data, and returns a value of the second argument, if an agent has another position.
- **ifMaxDynLevelNodeAhead** - two argument function which returns the value of the first argument if the agent currently visiting a given node is in a position pointing to a node with the locally greatest value of a dynamic level, and returns a value of the second argument if an agent has another position.
- **ifMaxDynCoLevelNodeAhead** - two argument function which returns the value of the first argument if the agent currently visiting a given node is in a position pointing to a node with the locally greatest value of a dynamic co-level, and returns a value of the second argument if an agent has another position.

5 Experiments

In this section we report some initial results of an experimental study of the proposed scheduling system. The system is composed of two modules: the LCGA scheduler working with either a global criterion (1) or local functions $h()$, and the GP system evolving local functions. The system was written in C++, with use of a library [5] to implement the modul of GP.

Because of large computational requirements of the GP module, the conducted experiments were limited to program graphs with the number of tasks not exceeding 40, population size of GP not exceeding 50, and the number of generation of GP not exciding 30. The size of each subpopulation of the LCGA was 80, the crossover probability was $p_c = 0.6$, and the mutation probability $p_m = 0.001$.

A precedence task graph [9] referred as *gauss*18, corresponding to a parallel version of the gaussian elimination algorithm was used in the experiment aiming to decompose a global criterion (1) into local criteria. It was assumed that this program is executed in a multiprocessor system of the MIMD architecture consisting of 8 processors arranged into *cube* topology.

Figure 3a presents an evolutionary GP process of decomposition a global function into local functions. Each individual of GP, representing a local scheduling function, is used by the LCGA module. The total execution time T (a global

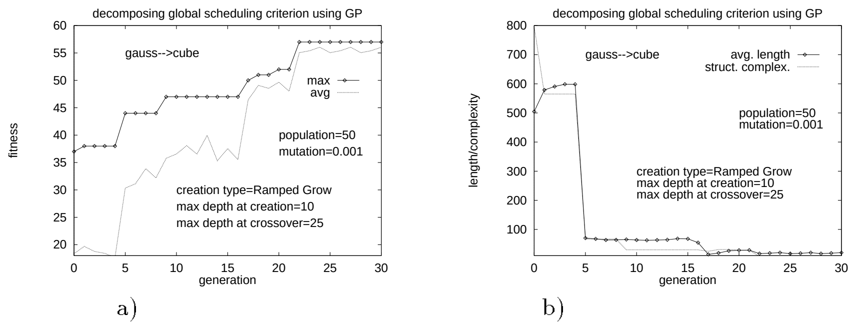

Fig. 3. Changing fitness function during GP evolutionary process (a), changing complexity of discovered local functions during evolutionary processing (b)

scheduling criterion) is evaluated for the final allocation found by the LCGA module. $T^* = const - T$ serves as the fitness function of a given individual from the population of GP. Figure 3a shows the best (max) and the average (avg) values of the fitness function in each generation of GP.

Figure 3b shows how the complexity of evaluated local functions are changing during the evolutionary process. An initial randomly created population of programs describing local scheduling functions, expressed in a LISP-like notation, contains programs with the average length ($avg.lenght$) equal to 504.75, and with structural complexity ($struct.complex.$) (the number of terminals and functions) of the best program in a population equal to 797. After 30 generation of GP the average length and the complexity drop to values 20.2 and 7, respectively. The best program in the last generation corresponds to the following local function $h()$:

((left(left(+(right SuccStartTime)(forward SuccStartTime)))).
The function can be interpreted in the following way:

1. Turn the agent right, to the next edge
2. Compute the sum of the time-start (**SuccStartTime**) of successors from the environment of the task visited by the agent.
3. Move the agent to a task, according to the current position of the agent.
4. Compute the sum of the time start (**SuccStartTime**) of succesors from the environment of the task curently being visited
5. Calculate the sum of the values from the points 2 and 4, and use it as the value of h().
6. Turn left two times; these operations do not influence the value of h().

How good is the found local function $h()$? To find it out, we conducted a number of experiments with the LCGA scheduling module using both global and local scheduling optimization criteria.

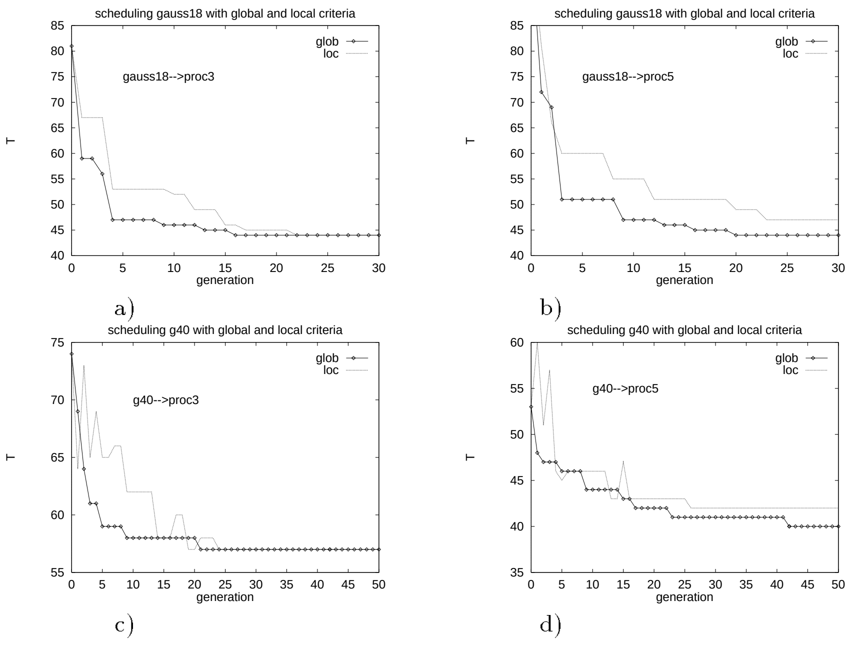

Fig. 4. Performance of scheduling algorithms with global and local optimization criteria

Figure 4a illustrates the work of the LCGA module using both optimization methods to schedule *gauss*18 in 3-processor system. Both methods find the optimal solution equal to 44, after relativelly small number of generations. The speed of convergence of the algorithm with local optimization criteria is slightly slower than that of the algorithm with the global optimization criterion. Increasing the number of processors to 5 (fully connected system) results in achieving a suboptimal solution by the algorithm with local criteria (see Figure 4b).

The purpose of the next experiment was to see if the evolved local criterion is suitable for scheduling programs different that *gauss*18. For this purpose, we used the precedednce task graph *g*40 [9] with 40 tasks. Behavior of both algorithms (see Figure 4c,d) for this new problem is similar as in the previous experiment. The algorithm with local criteria finds optimal (3 processors) and suboptimal (5 processors) solutions.

Results of experiments conducted with other program graphs available in the literature show similar behavior of the scheduling algorithm. It finds either optimal or suboptimal solution, with slightly more generations than the algorithm with the global criterion. To evaluate each individual from the population of the LCGA working with the global criterion, calculation global value of T is required. Time of calculation of T increases with groving number of tasks. The algorithm with local optimization is based on locally performed evaluation of local criteria, which can be done in parallel at least partiall, and may speed up the scheduling algorithm.

6 Conclusions

We have presented a new approach to develop parallel and distributed scheduling algorithms. The approach is based on decomposing the global optimization criterion into local criteria, with use of genetic programming. Both global and local criteria are used by the LCGA - based scheduling algorithm. The preliminary results indicate that the algorithm with the local criteria is able to find optimal or near optimal solutions in a number of generations comparable with the number required by the algorithm with the global criterion. The main advantage of the proposed approach is parallelization and distributed control of evaluated optimization criterion, what may speed up the scheduling algorithm implemented on a parallel machine.

Acknowledgement
The work has been partially supported by the State Committee for Scientific Research (KBN) under Grant 8 T11A 009 13.

References

1. I. Ahmad (Ed.). Special Issue on Resource Management in Parallel and Distributed Systems with Dynamic Scheduling: Dynamic Scheduling, *Concurrency: Practice and Experience*, 7(7), 1995
2. I. Ahmad and Y. Kwok, A Parallel Approach for Multiprocessing Scheduling. *9th Int. Parallel Processing Symposium*, Santa Barbara, CA, April 25-28, 1995
3. V. C. Barbosa, *An Introduction to Distributed Algorithms*, The MIT Press, 1996
4. J. Błażewicz, M. Dror and J. Węglarz, Mathematical Programming Formulations for Machine Scheduling: A Survey, *European Journal of Operational Research* 51, pp. 283-300, 1991
5. A. P. Fraser, *Genetic Programming in C++. A manual in progress for gpc++, a public domain genetic programming system*, Technical Report 040, University of Salford, Cybernetics Research Institute, 1994
6. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989
7. J. R. Koza, Genetic Programming, MIT Press, 1992
8. F. Seredynski, Competitive Coevolutionary Multi-Agent Systems: The Application to Mapping and Scheduling Problems, *Journal of Parallel and Distributed Computing*, **47**, 1997, 39–57.
9. F. Seredynski, Discovery with Genetic Algorithm Scheduling Strategies for Cellular Automata, in *Parallel Problem Solving from Nature-PPSNV*, A. E. Eiben, T. Back, M. Schoenauer and H.-P. Schwefel (Eds.), Lecture Notes in Computer Science 1498, Springer, 1998, 643–652.