AN EVOLUTIONARY APPROACH TO THE PARALLEL TASK SCHEDULING PROBLEM

ESQUIVEL S.C., GATICA C. R., GALLARD R.H.

Proyecto UNSL-338403[1]
Departamento de Informática
Universidad Nacional de San Luis (UNSL)
Ejército de los Andes 950 - Local 106
5700 - San Luis, Argentina.
E-mail: {esquivel, crgatica, rgallard}@unsl.edu.ar
Phone: + 54 652 20823
Fax : +54 652 30224

Abstract

A parallel program, when running, can be conceived a set of parallel components (tasks) which can be executed according to some precedence relationship.
In this case efficient scheduling of tasks permits to take full advantage of the computational power provided by a multiprocessor or a multicomputer system. This involves the assignment of partially ordered tasks onto the system architecture processing components.
This work shows the problem of allocating a number of nonidentical tasks in a multiprocessor or multicomputer system. The model assumes that the system consists of a number of identical processors and only one task may execute on a processor at a time. All schedules and tasks are nonpreeptive. The well-known Graham's [8] list scheduling algorithm (LSA) is contrasted with an evolutionary approach using the indirect-decode representation.

[1] The Research Group is supported by the Universidad Nacional de San Luis and the ANPCYT (National Agency to Promote Science and Technology).

1. Introduction

The precedence relationships between tasks are commonly delineated in directed acyclic graph known as the tasks graph. Nodes in the graph represent tasks and their duration and arcs represent the precedence relationship. Factors, such as number of processors, number of tasks and task precedence contribute to make difficult a good assignment.
The problem to find an schedule on $m > 2$ processors of equal capacity, that minimizes the whole processing time of independent tasks has been shown as belonging to the NP-complete class.
Task scheduling can be classified as static and dynamic. In the case of static scheduling some strong reasons make it applicable. First, static scheduling sometimes results in lower execution times than dynamic scheduling. Second static scheduling allows only one process per processor, reducing process creation, synchronization and termination overhead. Third, static scheduling can be used to predict speedup that can be achieved by a particular parallel algorithm on a target machine, assuming that no preemption of processes occur.
The current presentation shows the problem of allocating a number of nonidentical tasks in a multiprocessor or multicomputer system. The model assumes that the system consists of a number of identical processors and only one task may execute on a processor at a time. All schedules and tasks are nonpreeptive. The well-known Graham's [8] list scheduling algorithm (LSA) is contrasted with an evolutionary approach using the indirect-decode representation..

2. A deterministic model

A parallel program is a collection of tasks, some of which must be completed before than others begin. In a deterministic model, the execution time for each tasks and the precedence relations between them are known in advance. This information is depicted in a directed graph, usually known as the task graph.
In Fig. 1 we have eight tasks with the corresponding duration and their precedence relations

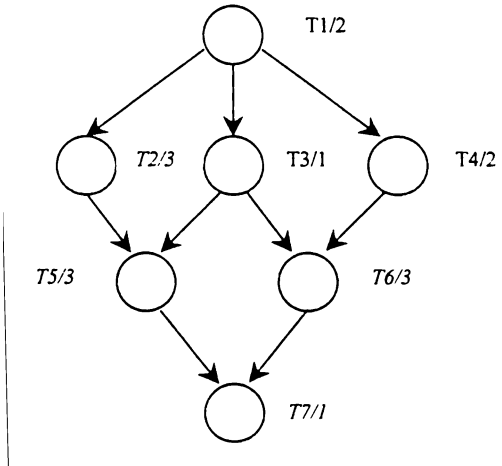
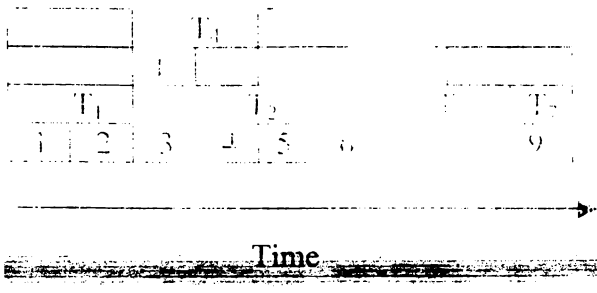


Fig. 1 . The model task graph

Even if the task graph is a simplified representation of a parallel program execution, ignoring overheads due to interrupts for accessing resources etc., it provides a basis for static allocation of processors.
A schedule is an allocation of tasks to processors which can be depicted by a Gantt chart.
In a Gantt chart, the initiation and ending times for each task in the available processors is indicated and the makespan (total execution time of the parallel program) of the schedule can be easily derived.



schedule of the parallel tasks of the task graph of figure 1 onto two processors. By simple observation we notice a makespan of 10 and an utilization of a 100% for processor P_1 and an utilization of 60% for processor P_2. Also an speed-up of 1.6 can be stablished.

Fig. 2. Scheduling 7 tasks onto 2 processors by LSA

When looking to the makespan an optimal schedule is such that the total execution time is minimized. Other performance variables, such as individual processor utilization or evenness of load distribution can be considered.
As we can see some simple scheduling problems can be solved to optimality in polynomial time while others can be computationally intractable.

3. The List Scheduling Algorithm (LSA)

As we are interested in scheduling of arbitrary tasks graphs onto a reasonable number of processors we would be content with polynomial time scheduling algorithms that provide good but no optimal solutions.

For a given list of tasks ordered by priority, it is possible to assign tasks to processors by always assigning each available processor to the first unassigned task on the list whose predecessor tasks have already finished execution.
Let be:
$T=\{T_1,.....Tn\}$ a set of tasks,
$\tau: T \rightarrow (0, \infty)$ a function which associates an execution time to each task,
$\leq$ a partial order in T and
L a priority list of tasks in T.

Each time a processors is idle, it immediately removes from L the first ready task; that is, an unscheduled task whose ancestors under $\leq$ have all completed execution. In the case that two or more processors attempt to execute the same task, the one with lowest identifier succeed and the remaining processors look for another adequate task.
The Gantt chart of Fig. 5.2, resulted of applying the list scheduling algorithm to the task graph of Fig. 5.1, with the priority list $L = [T_1, T_2, T_3, T_4, T_5, T_6, T_7]$.

Using this heuristic, contrary to the intuition, some anomalies can happen. For example, increasing the number of processors, decreasing the execution times of one or more tasks, or eliminating some of the precedence constraints can actually increase the makespan. In his work Graham presented different examples to show this problem.

4. Evolutionary algorithms to provide near-optimal solutions

The task allocation problem has been investigated by many researchers [3] ,[4], [5], [6], [7], [9], [10]. Several heuristics methods has been proposed, such as mincut-based heuristics, orthogonal recursive bisection, simulated annealing, genetic algorithms and neural networks.

From the representation perspective many evolutionary computation approaches to the general scheduling problem exists. According to solution representation these methods can be roughly categorized as indirect and direct representation (Bagchi et al, 1991 [1]).

In the case of indirect representation of solutions the algorithm works on a population of encoded solutions. Because the representation do not directly provides a schedule a scheduler builder is necessary to transform a chromosome into a schedule, validate and evaluate it. The scheduler builder guarantees the feasibility of a solution and its work depends on the amount of information included in the representation. This is the Bagchi et al. strongly recommended approach.

In direct representation (Bruns 93 [2]) a complete and feasible schedule is an individual of the evolving population. The only method that performs the search is the evolutionary algorithm because the represented information comprises the whole search space.

In our work we devised different evolutionary computation approaches to task scheduling. First we addressed two new different representation schemes; direct-ASAP (direct-as-soon-as-posible) and indirect-decode. Second, we addressed the question of attempting to improve performance by means of different recombination and mating approaches.

5. Summary of the work

A genetic approach was compared against the List Scheduling Algorithm.

By analysing results the following comparison can be done:

- The genetic approach found many and no a single optimal solution for any case.
- All the anomalies observed with LSA do not hold when GA is applied, because:
  - When the number of processors is increased the minimum (optimum) makespan is also found.
  - When the duration of tasks is reduced this reduction is reflected in a reduced optimum makespan.
  - When the number of precedence restrictions is reduced the optimum makespan is preserved.

A more detailed analysis on each run detected that in most of the cases alternative solutions do not include, or include a low percentage, of non-optimal alternative solutions. That means that the final population is composed of many replicas of the optimal solutions due to a loss of diversity. This fact stagnate the search and further improvements are difficult to obtain.

settings and recombination approaches.

6. Conclusions

The allocation of a number of parallel tasks in parallel supporting environments, multiprocessors or multicomputers, is a difficult and important issue in computer systems.

In this work we approached allocation attempting to minimize makespan. Other performance variables such as individual processor utilization or evenness of load distribution can be considered.

As we are interested in scheduling of arbitrary tasks graphs onto a reasonable number of processors, in many cases we would be content with polynomial time scheduling algorithms that provide good but no optimal solutions. The list scheduling algorithm (LSA) satisfy this requirement.

A genetic approach was undertaken to contrast its behaviour against the LSA.

Preliminary results on the selected test suite showed two important facts. Firstly, GA provides not a single but a set of optimal solutions, providing fault tolerance when system dynamics must be considered. Secondly, GA is free of the LSA anomalies.

This facts do not guarantee finding optimal solutions for any arbitrary task graph but show a better approach to the problem.

Consequently further research is necessary to investigate potentials and limitations of the GA approach under more complex test suites, different representations, and convenient genetic operators.

A new *as-soon-as-possible* (ASAP) approach oriented to direct representation is now being implemented.

7.Bibliography

[1] Bagchi S., Uckum S., Miyabe Y., Kawamura K. – *Exploring problem-specific recombination operators for job shop scheduling*- Proceedings of the Fourth International Conference on Genetic Algorithms, pp 10-17, 1991.

[2] Bruns R. – *Direct chomosome representation and advanced genetic operators for production scheduling*. Proceedings of the Fifth International Conference on Genetic Algorithms, pp 352-359, 1993.

[3] Cena M.,Crespo M., Gallard R., - *Transparent Remote Execution in LAHNOS by Means of a Neural Network Device*- ACM Press, Operating Systems Review , Vol. 29, Nr. 1, pp 17-28, January 1995

[4] Ercal F.- *Heuristic approaches to Task Allocation for parallel Computing*- Doctoral Dissertation, Ohio State University, 1988.

[5] Flower J., Otto S., Salama M. – *Optimal mapping of irregular finite element domains to parallel processors*- Caltech C3P#292b, 1987.

[6] [Fox G. C. – *A review of automatic load balancing and decomposition methods for the hipercube*- In M Shultz, ed., Numerical algorithms for modern parallel computer architectures, Springer Verlag, pp 63-76, 1988.

[illegible] *A dynamic load balancer* - Proc. the 2nd Conf. on Hipercube multiprocessors, pp 114-121, 1987.

[8] Graham R. L. - *Bounds on multiprocessing anomalies and packing algorithms*.- Proceedings of the AFIPS 1972 Spring Joint Computer Conference, pp 205-217, 1972.

[9] Kidwell M. - *Using genetic algorithms to schedule tasks on a bus-based system*. - Proceedings of the Fifth International Conference on Genetic Algorithms, pp 368-374, 1993.

[10] **Mansour N., Fox G.C. – *A hybrid genetic algorithm for task allocation in multicomputers*.** Proceedings of the Fourth International Conference on Genetic Algorithms, pp 466-473, 1991.