PERFORMANCE BOUNDS ON MULTIPROCESSOR SCHEDULING STRATEGIES FOR STATICALLY ALLOCATED PROGRAMS

LARS LUNDBERG

Department of Computer Engineering,
Lund University, P.O. Box 118, S-22100 Lund, Sweden
email: larsl@dit.lth.se

Abstract.

In multiprocessors with static allocation of processes to processors, scheduling can be done locally for each processor. The scheduling strategy may have dramatic effect on the execution time of a parallel program. It is an NP-hard problem to find an optimal schedule, and very little is known of how close the heuristic solutions get.

The major result here is a theorem stating that if certain program parameters, which can be obtained from an execution of the program on a single-processor, are known, the execution time of the optimal schedule can be calculated within a factor equal to the largest number of border processes on one processor. Border processes are processes which communicate with other processors. The program parameters are obtained using a previously developed tool.

Due to the generality of this theorem, the proof is rather complex because it has to cover a large range of situations. The theorem itself, however, is easy to apply, making it possible to compare the performance of different scheduling strategies with the optimal case. The proof also gives important hints on how to design efficient scheduling algorithms for statically allocated programs.


Key words: Performance Bound, Static Allocation, Local Scheduling, Theory.

1. Introduction.

In a previous paper [8], a method for calculating optimal performance bounds on a restricted class of parallel programs (chain structured programs) has been presented. Here, this result is generalized to programs with an arbitrary structure. To make it possible to understand the results reported here, a large part of the introductory definitions from the previous paper had to be duplicated.

The way processes are scheduled to processors affects the performance of multiprocessor systems. The performance criteria are not the same for all kinds of systems; e.g. in real-time systems, the most important criterion is the schedule's
ability to meet the deadlines imposed by the outside world. Rate monotonic scheduling is one of the best known results in this field [10, 11].

Here we are going to look at another type of systems where only one parallel program, consisting of a number of processes, is executed on a multiprocessor. In such systems, the most important performance criterion is the completion time of the parallel program. We assume that processes are statically allocated to processors, and that the allocation of processors has been decided. Overhead for context switching and external memory references is disregarded.

2. Parallel program characteristics.

A process in a parallel program can be in either of the three states Running, Ready or Blocked (see figure 1). An active process is a process which is in the Running or the Ready state. Processes communicate through Send and Receive operations, which are connected in pairs. A process is blocked if it reaches a Receive before the corresponding Send has been executed. Similarly, a process is blocked if it reaches a Send before the corresponding Receive has been executed. Transitions between the Ready and the Running states are controlled by the schedule. All processes are created at the start of the execution. Some processes, however, may be initially blocked.

The scheduling of processes within each processor affects the total execution time of a parallel program. Figure 2 shows three schedules, using two (A and B) or three processors (C), of one parallel program with three processes (P1, P2 and P3). These processes synchronize through two pairs of Send and Receive operations. Each arrow in figure 2 corresponds to a Send, i.e. the program contains the following two Send operations:

1. Process P1 activates process P3 with a Send, i.e. P3 is initially blocked in the corresponding Receive.
2. Process P1 activates process P2 with a send, i.e. P2 is initially blocked in the corresponding Receive.