Free Parallel Computer (Variable Number of Instruction Fetch/Variable Number of Instruction Execute Computer)

1. Formulation of the Problem

The good scheduling of the program tasks executed on parallel systems can improve the system performance in terms of execution time of the program and processor utilization. The components of the scheduling systems are the program tasks, the target machine and the scheduling algorithm. According to the scheduling algorithm tasks are assigned for execution to the processing elements, so that that the parallel execution time of the program is minimized. Up till now all the available scheduling techniques depend on the number of processor elements in the multiprocessor systems. These scheduling techniques are used in compilers for optimization of the code generated from the source program in order to exploit the hardware capabilities of the system as much as possible. The so generated machine language program is produced for a specific number of processors and has a fixed degree of parallelism. The following problems result from scheduling techniques depending on the degree of parallelism:

1.1 If the number of the processors in the system is changed, the machine code obtained through scheduling of the program for a fixed degree of parallelism, cannot be executed efficiently or at all on a system with different number of processors. There is a necessity for rescheduling of the user source program and assigning for execution of a newly generated machine language program with fixed degree of parallelism corresponding to the new number of processors (Fig.1).

1.2 However, there is no method for assigning for execution machine language programs with different degrees of parallelism corresponding to the changed number of processors in the multiprocessor system.

2. Proposed Solution of the above Problems

The above problems arise from static scheduling methods depending on the degree of parallelism. Therefore there is a necessity for scheduling not depending on the degree of parallelism and also for an architecture which can support and execute the so scheduled program (called standard program) efficiently (Fig.2). We call this architecture Free Parallel Computer or Variable Number of Instruction Fetch/Variable Number of Instruction Execute Computer. Once the user program is scheduled using this method, it should run with equal success on variable number of processors, i.e. its execution time should be near to optimal when it is run on different number of processors.

In order to justify our idea, we have tried to develop a scheduling algorithm that does not depend on the degree of parallelism and compare its results with those of an existing method but applied separately for different number of processors. In our simulations we have assumed that the latencies of all the tasks are equal. The algorithm consists of two phases. The first phase is scheduling of the program using a classical list scheduling method (CP/MISF) for number of processors equal to the minimal number of processors necessary for execution of the program with schedule equal to the critical path. Thus blocks of tasks are formed in which the tasks can be executed simultaneously. The second phase consists of rearrangement of the tasks in the blocks, according to their distance from their predecessors in the previous block. The tasks are ordered by two functions: the length of the maximal path from the tasks to the exit node, and their distance from their intermediate predecessors in the previous block (Fig.3).

3. Why We Think the Solution Is Good

When this algorithm is applied, the tasks are ordered in a way which allows the execution of the standard program on variable number of processors with schedule length comparable to the optimal one (i. e. comparable to the schedule length obtained if the program is scheduled using the Critical Path Method separately for different number of processors, as shown by the simulation results so far). The results we have obtained for the moment show that our idea makes sense and we would like to continue our work considering the case when the tasks have different latencies and taking in account the communication between the tasks.

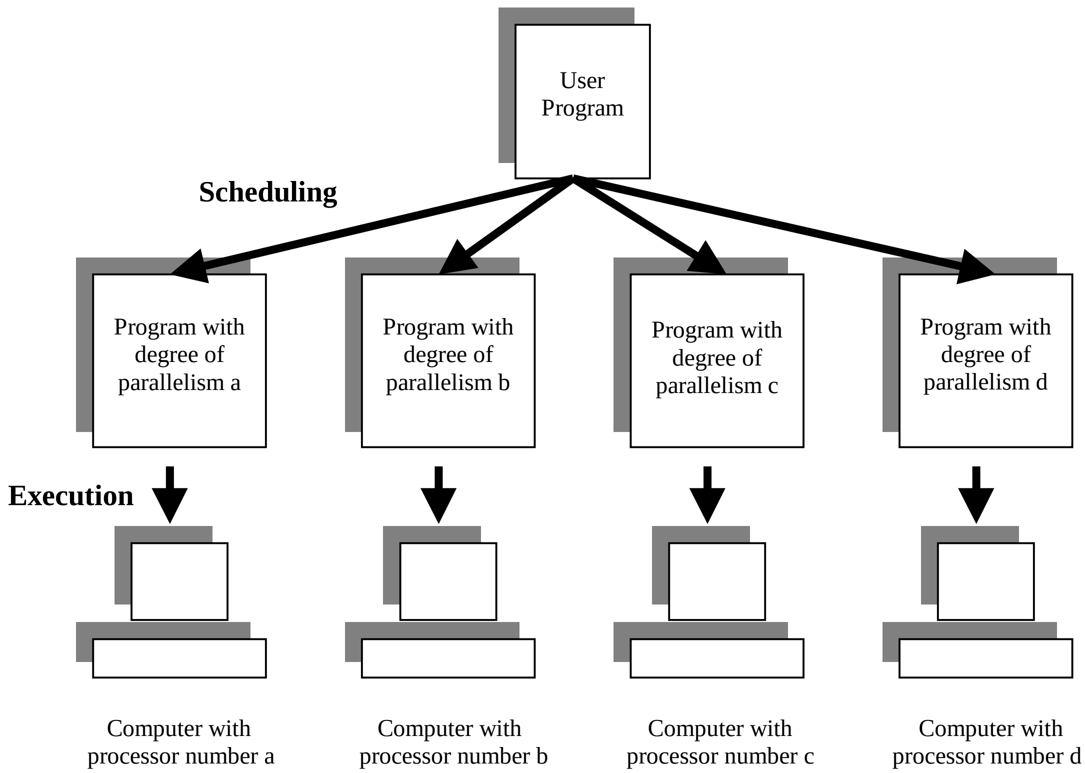

Fig.1 Necessity of rescheduling the user program when the number of processors in the parallel system is changed

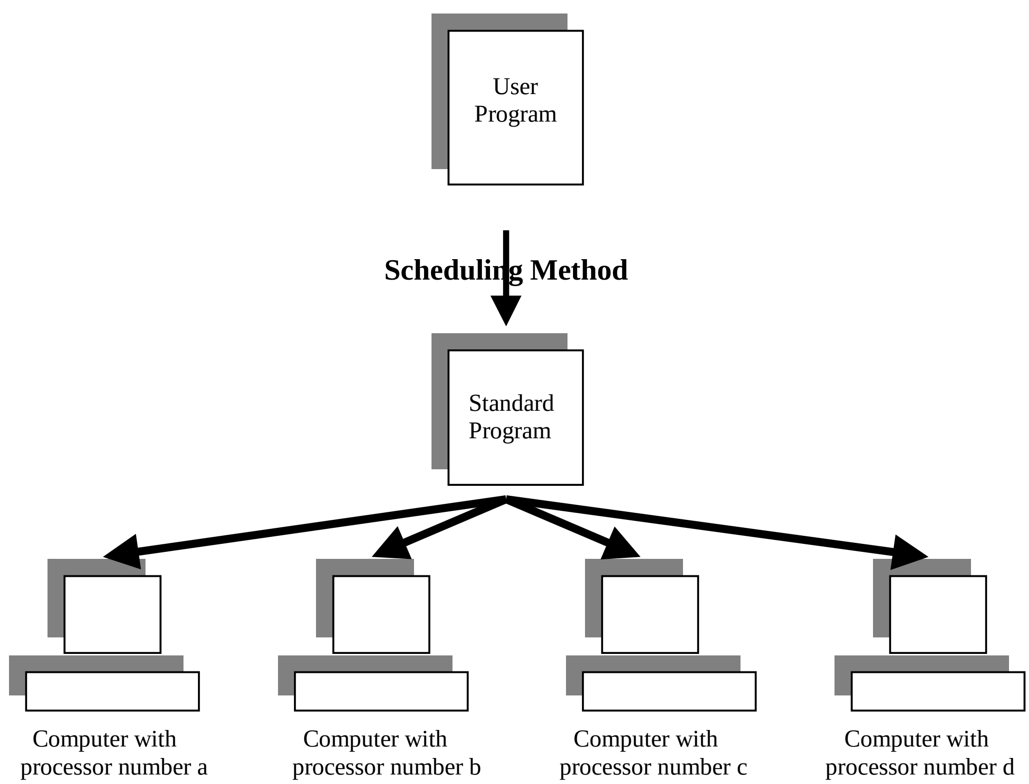

Fig.2 The proposed solution of the problem

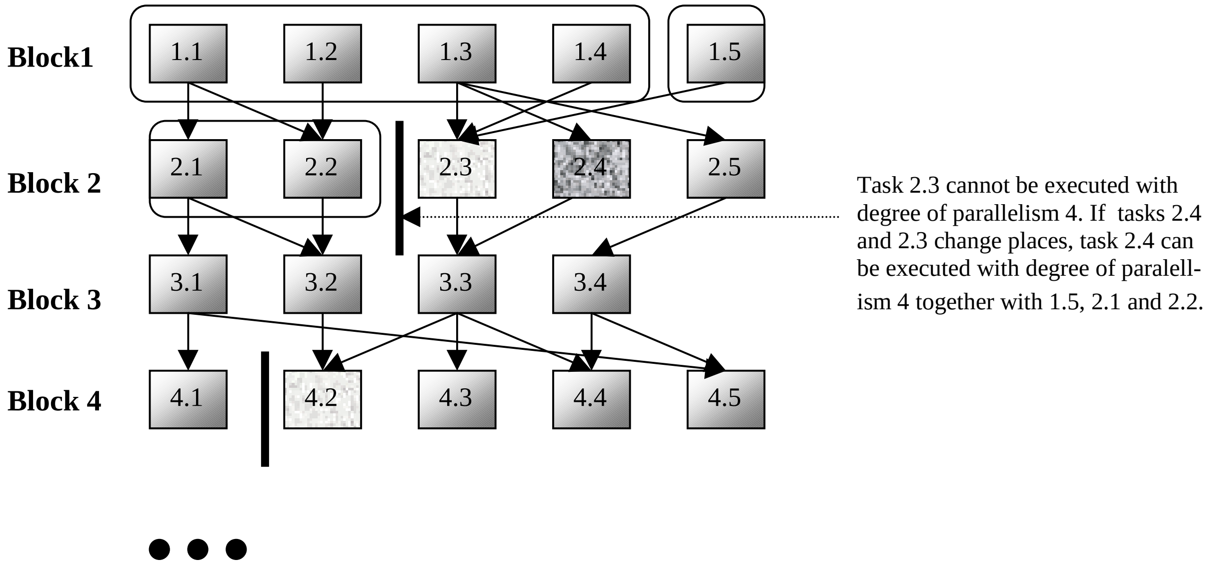

Fig. 3a First phase of the algorithm: forming of blocks using CP/MISF method for processor number p equal to the minimal necessary number for executing the program with schedule length equal to the critical path

- For this example p =5
- The distance between the task and its successor in the next block is equal to the number of tasks which have to be assigned for execution starting from the specific task till its successor. In this case the distance between task 1.1 and its successor 2.1 is 5, the distance between task 1.4 and its successor 2.3 is 4, etc.

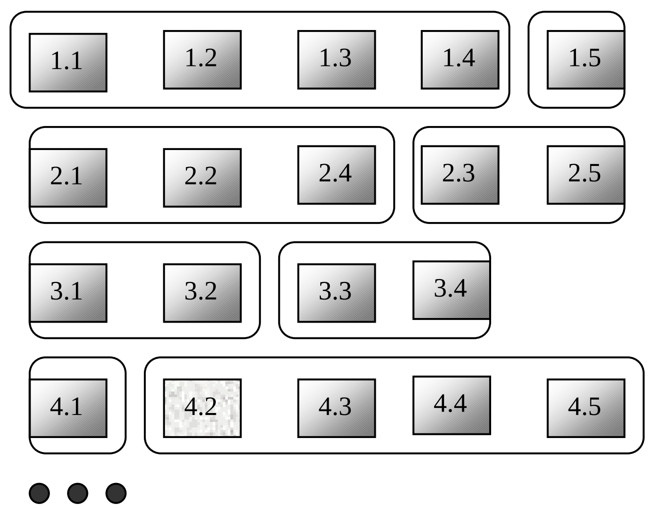

Fig. 3b Second phase of the algorithm: Arranging the tasks according to their distance with their successors in the next block

- The length of the schedule should be close to the optimal when the program is executed on different number of processors – for example for number of processors 5, 4, 3, 2. In this case there is a necessity of rearrangement of tasks only when execution with 4 processors is considered. When the program is executed on 4 processors, first are assigned for execution tasks 1.1, 1.2, 1.3 and 1.4. However the second set of tasks 1.5, 2.1, 2.2 and 2.3 cannot be assigned for execution at one and the same time because 2.3 depends on 1.5. But if the places of tasks 2.3 and 2.4 are changed, then tasks 1.5, 2.1, 2.2, 2,4 can be executed at one and the same time with degree of parallelism 4 because these tasks are independent. After that, tasks 2.3, 2.5, 3.1 and 3.2 are independent and can be assigned for execution at one and the same time. The next set of tasks should be 3.3, 3.4, 4.1 and 4.2. However 4.2 depends on 3.3 and cannot be assigned for execution at one and the same time with tasks 3.3, 3.4 and 4.1. It cannot be substituted with any of the tasks 4.3, 4.4 or 4.5 because they depend either on 3.3 or 3.4, or both.

5 processors	**4 processors**	**ZZZZZZZZ3 processors**	**2 processors**
1.1 1.2 1.3 1.4 1.5	1.1 1.2 1.3 1.4	1.1 1.2 1.3	1.1 1.2
2.1 2.2 2.4 2.3 2.5	1.5 2.1 2.2 2.4	1.4 1.5 2.1	1.3 1.4
3.1 3.2 3.3 3.4 4.1	2.3 2.5 3.1 3.2	2.2 2.4 2.3	1.5 2.1
4.2 4.3 4.4 4.5	3.3 3.4 4.1	2.5 3.1 3.2	2.2 2.4
…	4.2 4.3 4.4 4.5	3.3 3.4 4.1	2.3 2.5
	…	4.2 4.3 4.4	3.1 3.2
		4.5…	3.3 3.4
			4.1 4.2
			4.2 4.4
			4.5…

Fig. 3c The schedules obtained by using scheduling method not depending on the degree of parallelism for different number of processors