Competitive Implementation of Parallel Programs

X. Deng, E. Koutsoupias, and P. MacKenzie

Abstract. We apply the methodology of competitive analysis of algorithms to the implementation of programs on parallel machines. We consider the problem of finding the best on-line distributed scheduling strategy that executes in parallel an unknown directed acyclic graph (dag) which represents the data dependency relation graph of a parallel program and which is revealed as execution proceeds. We study the competitive ratio of some important classes of dags assuming a fixed communication delay ratio \( \tau \) that captures the average interprocessor communication measured in instruction cycles. We provide competitive algorithms for divide-and-conquer dags, trees, and general dags, when the number of processors depends on the size of the input dag and when the number of processors is fixed. Our major result is a lower bound \( \Omega(\tau / \log \tau) \) of the competitive ratio for trees; it shows that it is impossible to design compilers that produce almost optimal execution code for all parallel programs. This fundamental result holds for almost any reasonable distributed memory parallel computation model, including the LogP and BSP model.

Key Words. Parallel computation, Competitive analysis, Communication delay, Scheduling, Compiler.

1. Introduction. The execution profile of a program can generally be represented as a directed acyclic graph (dag): Nodes represent instructions, or sets of instructions, and edges represent dependencies between individual nodes. An edge \((u, v)\) denotes that the results of node \(u\) are required for the execution of node \(v\). Usually, the dag of a program is not known at the compile time, since it depends on the input data and the runtime conditions. This, however, is no problem at all for a uniprocessor system because nodes can be executed as they become available. Any scheduling which would not intentionally idle achieves the optimal completion time. Even for the system performance metric of mean response time, the Round-Robin scheduling strategy guarantees a mean response time at most twice the optimum, without using any information about the actual executed dag [9].

The situation becomes more complicated when we deal with parallel programs, since one of the intricacies of parallel computation is that the optimum algorithm may depend critically on the profile of the parallel machine. Nevertheless, we may classify parallel models into two major categories: models with shared memory, and models with distributed memory. Our work focuses on a simplified distributed memory model, the
communication delay model introduced in [11]. We must stress, however, that our main negative result applies to most models with distributed memory.

1.1. Distributed Memory Models. In this paper we assume the Papadimitriou–Yannakakis communication delay model [10], [11] to study the implementation problem of parallel programs on general purpose multicomputer systems. In this model, a universal parameter $\tau$, the communication delay between processors measured in instruction cycles, is used to abstract communication in parallel machines. A processor can execute a node of the dag if every predecessor node has been executed either by the same processor or by some other processor $\tau + 1$ or more time units before. Putting it differently, there is a delay of $\tau$ steps for the result of a node to become available to other processors.

Other models of parallel computation assume a communication delay and usually have more parameters for the communication cost. The BSP model, proposed as a bridge between software and hardware [14], and the LogP model, developed by Culler et al. [2], are two such models. These models are not limited to parallel computing but they try to capture the locality properties of a machine, a factor that is also important in distributed systems, especially for clusters of workstations. Similar trends for unifying parallel and distributed computing are observed in parallel architecture and parallel programming [2], [13]. Even for shared memory machines, there is a factor similar to communication delay: It is the ratio of the cost for accessing the shared memory over the cost for accessing private caches of individual processors (a widely used technology to improve performance of parallel computers).

1.2. Competitive Analysis. It was shown in [11] that there exists a polynomial-time algorithm that approximates the optimum execution time of any dag within a factor of 2 when the number of available processors is unbounded. The approach of [10] and [11] assumes that the computation dag is known before the algorithm actually runs, essentially at compile time (the algorithm in [11] can be made on-line, but only by using a very high number of processors [5]). This assumption may not be realistic for all the parallel programs since conditional statements and loops are important parts of any programming language. First, crucial parameters such as the actual shape of the dag and the size of its nodes are found out by processors executing various parts of the program on-line. Second, runtime parameters such as the number of available processors during execution may not be known beforehand and may change dynamically. Because of communication delays, each processor has to make its own decision on how to proceed based on incomplete information. Thus, the situation naturally falls into a currently very active field: Competitive analysis of on-line algorithms. More specifically, competitive analysis requires that the compiler outputs not just code, but a distributed on-line algorithm for the execution of the program in hand, hopefully one that minimizes the worst-case ratio of the completion time of the strategy on a given source program, to the completion time of the optimal execution of the program (with complete information concerning runtime conditions and execution paths).

The concept of competitive analysis has been introduced recently by Sleator and Tarjan in [12] (see also [6] and [8]). For the scheduling problem, however, Graham [4] recognized almost 30 years ago that, without knowing job lengths, a scheduler can achieve twice the optimum. To define the competitive ratio for our problem formally, we