A Randomized Parallel Branch-and-Bound Algorithm

Virendra K. Janakiram, Edward F. Gehringer, Dharma P. Agrawal, and Ravi Mehrotra

Received July 1988; revised February 1989

Randomized algorithms are algorithms that employ randomness in their solution method. We show that the performance of randomized algorithms is less affected by factors that prevent most parallel deterministic algorithms from attaining their theoretical speedup bounds. A major reason is that the mapping of randomized algorithms onto multiprocessors involves very little scheduling or communication overhead. Furthermore, reliability is enhanced because the failure of a single processor leads only to degradation, not failure, of the algorithm. We present results of an extensive simulation done on a multiprocessor simulator, running a randomized branch-and-bound algorithm. The particular case we consider is the knapsack problem, due to its ease of formulation. We observe the largest speedups in precisely those problems that take large amounts of time to solve.

KEY WORDS: Branch-and-bound; randomized algorithms; parallel algorithms; speedup.

1. INTRODUCTION

Much of the interest in parallel processing derives from a desire to solve large problems. Algorithms that perform well on small problems do not always exhibit acceptable performance as the problem size grows. Algorithms whose complexity increases rather slowly with problem

---

1 This work has been supported by the U.S. Army Research Office under Contract No. DAAG 29-85-K-0236.
2 Author’s current address: AT&T Bell Labs, Crawford’s Corner Road, Holmdel, NJ 07733.
3 Computer Systems Laboratory, Department of Electrical and Computer Engineering, Box 7911, North Carolina State University, Raleigh, North Carolina 27695.
4 Arthur Andersen and Company, 33 West Monroe, Chicago, IL 60603.
size—less than linearly, for example—can effectively be “scaled up” to handle large problems. This is true for serial processing, and usually also for parallel processing, provided that the algorithm can be effectively decomposed for multiple processors.

Some algorithms decompose more effectively than others. An algorithm that requires frequent interprocessor communication will perform more poorly as processors are added. If the processors can work more or less independently, however, the total amount of work tends to remain nearly constant, and the benefit of adding processors is more apparent. A computation using \( n \) processors then finishes in about \( 1/n \)th the time that a single processor would need; so that linear speedup has been achieved. Thus, to perform parallel processing efficiently, we need to concentrate on both computational complexity and speedup. In this paper, we show that a particular class of algorithms known as randomized algorithms offer benefits related to both speedup and complexity: the calculations require less elapsed time, and the code for decomposing the problem into concurrent processes is much simpler.

2. RANDOMIZED ALGORITHMS

A randomized algorithm,\(^1\) is an algorithm obtained by introducing randomness into the solution procedure. The use of randomized algorithms can yield a significant reduction in the average time to perform certain problem-solving searches. A randomized algorithm operating on a fixed input may take any one of several possible computation paths, depending on some internally controlled random process derived from the problem structure. As an illustration, consider the simple problem \( P \) of finding a path from a root node \( r \) of a tree \( T \) of \( v \) nodes to a node \( e \) such that \( e \) is a leaf node of \( T \). It is perhaps convenient to think of the problem as follows. Node \( r \) represents a starting node and leaf nodes are the target or goal nodes. All goal nodes are equally desirable and we need to find a path to any one of them.

A deterministic algorithm to solve \( P \) will start from node \( r \) and (deterministically) select an edge (e.g., the leftmost) by which to proceed to an adjacent node \( x \). If \( x \) is not a leaf node, this process will be repeated, and continued until ultimately a leaf node is reached. The algorithm will find the same leaf node \( e \) each time it is executed (e.g., node \( a \) in Fig. 1).

In contrast, a randomized algorithm will start from node \( r \) and randomly select an edge to some adjacent node \( x \). If \( x \) is not a leaf node, the step is repeated by randomly selecting one of the remaining edges connected to node \( x \). This process is continued until ultimately a leaf node is reached. The algorithm follows a completely random solution path,