Abstract

Analyzing the running time of a concurrent algorithm can be as important as verifying its partial correctness or termination. A simple technique for analyzing the running time of a concurrent algorithm is presented. To analyze an algorithm with concurrent processes, the interaction between the processes must be considered. This is done by using the communication sequences of the processes as the basis of the analysis. The technique is used for analyzing and comparing three concurrent algorithms for finding the root of a real function.

1. INTRODUCTION

A concurrent algorithm specifies a number of processes $P_1, P_2, \ldots, P_n$ which can be executed in parallel. This paper presents an example of how the running time of such a concurrent algorithm can be estimated. Techniques for estimating the running time of sequential algorithms (only one process) are very well developed [Knuth 1968] and [Aho, Hopcroft, and Ullman 1974]. Analyzing a concurrent algorithm with several processes presents additional problems because the interaction between the processes must be taken into account. Such interaction is for example necessary when the processes exchange intermediate results. The interaction between concurrent processes can be very complex to analyze, which is also why it is difficult to construct and verify concurrent algorithms. The challenge is of course to avoid the complexity in reasoning about the algorithms and still obtain realistic results.

2. ROOT SEARCHING

In this section a concurrent algorithm for finding the root of a continuous function, $H$, is presented. Assume that $H$ is a real continuous function defined on the
closed interval \([a, b]\). Assume furthermore that \(H(a) \cdot H(b) \leq 0\) and that \(H\) has only one root in \([a, b]\).

There are many well known sequential algorithms for finding the root, for example binary search. Let \(T_H\) denote the average time it takes to evaluate \(H\). If \(T_H\) dominates other quantities in the running time, then it is well known that the running time, \(B_T\), for binary search is:

\[
B_T \approx T_H \cdot \log \frac{l_0}{\text{eps}}
\]

where \(\text{eps}\) is the accuracy with which the root is obtained and \(l_0 = b-a\). (For binary search the worst, best, and average case running times are the same).

The above running time can be improved by letting several processes evaluate \(H\) at different interval points concurrently.

2.1 A Two Process Algorithm

The following algorithm [Kung 1976] with only two concurrent processes is simple, but manageable.

Two processes, \(p\) and \(q\), evaluate the function \(H\) at two different interval points: \(x_p\) and \(x_q\).

\[
\text{interval: } [a, x_p, x_q, b]
\]

Like the binary search, the algorithm works by narrowing the interval. Assume that \(p\) finishes its evaluation of \(H\) first and \(H(a) \cdot H(x_p) \leq 0\), i.e. the root is in \([a, x_p]\). The interval is now changed to \([a, x_p]\), therefore the work of \(q\) is wasted and \(q\) must be directed to work in the new interval \([a, x_p]\) as soon as possible. If on the other hand \(H(a) \cdot H(x_p) > 0\), the root is in the interval \([x_p, b]\). In this case the work currently being done by \(q\) is utilized.

As we shall see later, the placement of \(x_p\) and \(x_q\) is crucial for the efficiency of the algorithm. Let \(D\) be a function for calculating \(x_p\) and \(x_q\) from \(a\) and \(b\):

\[
x_p = D(a, b, p) \quad \text{and} \quad x_q = D(a, b, q)
\]

Administration of the interval is the central part of the algorithm. The interval is an abstract date type with two operations, respond and result: