Computational Complexity of Random Access Stored Program Machines*

by

J. HARTMANIS

Department of Computer Science
Cornell University
Ithaca, New York

ABSTRACT

In this paper we explore the computational complexity measure defined by running times of programs on random access stored program machines, RASP's. The purpose of this work is to study more realistic complexity measures and to provide a setting and some techniques to explore different computer organizations. The more interesting results of this paper are obtained by an argument about the size of the computed functions. For example, we show (without using diagonalization) that there exist arbitrarily complex functions with optimal RASP programs whose running time cannot be improved by any multiplicative constant. We show, furthermore, that these optimal programs cannot be fixed procedures and determine the difference in computation speed between fixed procedures and self-modifying programs. The same technique is used to compare computation speed of machines with and without built-in multiplication. We conclude the paper with a look at machines with associative memory and distributed logic machines.

Introduction

The purpose of this paper is to study the quantitative problems about computation speed of random access stored program machines, called RASP's. These abstract machines were defined [1, 2] to provide theoretical models which look more like real computers and which reflect some aspects of real computing more directly than previously studied Turing machines.

In the following section we given an informal definition of random access stored program machines and derive some elementary results about the computational complexity measure based on the number of operations used by RASP's. Next, by a new technique involving the size of the computed functions, we study the problem of how well we can define and approach the best possible computation time with RASP programs. It is shown that there exist arbitrarily complex computations for which we can write programs whose computation time cannot be improved by any factor. That is, there exist functions $f_j(n)$ which can be computed in time $T_j(n)$ but cannot be computed in time $(1 - \varepsilon)T_j(n)$

*This research has been supported in part by National Science Foundation Grant GJ-155.
for any $\epsilon > 0$. This result contrasts sharply the corresponding results for
time-limited, tape-limited or reversal-limited Turing machine computations [3, 4, 5, 6],
all of which can be improved by a constant factor since these models permit
compression of output data (i.e., the number of tape-symbols is not fixed). It is
also interesting to observe that this result is obtained without using a diagonal
argument which up until now was the only technique to prove results of this
type. Clearly, some results about computation speeds have been obtained pre-
viously by counting arguments but they did not extend to arbitrarily complex
computations [7, 8].

Further analysis of the above result reveals that these optimal programs
cannot be fixed procedures in that they must modify themselves and grow in
length. This proves that for infinitely many computations fixed procedures are
slower than self-modifying programs. Furthermore, it is shown that self-
modification can at best speed-up the computation time by a constant factor
and that for infinitely many, arbitrarily complex computations they are faster
by a factor which depends on the size of the fixed program. Next, we compare
RASP's with and without built-in multiplication and prove that the RASP's
with multiplication are faster and give an upper bound for the maximal speed
difference. It is interesting to note that the above mentioned results are much
sharper than any results we have been able to obtain for Turing machine
computations. For example, it is still not known whether million-tape Turing
machines are faster than two-tape Turing machines for complicated computa-
tions [9]. We only know that for real-time computations two tapes are better than
one tape [7, 8]. Similarly, we do not know whether non-deterministic Turing
machines are faster than deterministic machines nor do we know that Turing
machines with jump operations or multidimensional tapes have any speed
advantage over regular, one-tape machines for arbitrarily complex computations.

We conclude the paper by showing that RASP's with associative memories
can at best improve the computation speed by the square root and raise some
questions about distributed logic machines.

Random Access Stored Program Machines

In this section we describe a random access stored program machine model,
RASP, and derive some results about computation times on RASP's. We are
giving only an informal description of our model to avoid making a rather
simple topic look more complicated.

A random access stored program machine, abbreviated RASP, consists of a
memory $M$ and a finite set of instructions $I$,

$$\text{RASP} = \langle M, I \rangle.$$  

The memory $M$ of the RASP consists of two special registers, called Accumulator,
AC, and Instruction Counter, IC, and an infinite sequence of memory registers:

$$R_1, R_2, R_3, \ldots.$$  

Each register is capable of holding an arbitrary, finite-length binary string.