AUTOMATIC PROGRAM RESTRUCTURING FOR HIGH-SPEED COMPUTATION

D. J. Kuck
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

ABSTRACT

This paper discusses program transformations to allow traditional programming languages to be used on a variety of architectures. A uniform framework is presented in which various array and multiple-scalar machines are all special cases. The methods are implemented in the Parafrase system, which can be used for machine design as well as compiler design studies.

1. INTRODUCTION

Advances in computer speeds in the past have come about more from technology advances than from a better understanding of the compilers and algorithms to be used on the machines being designed. However, the rate of increase in effective computer speeds seems to have dropped off in the past 20 years. Furthermore, raw hardware speeds have become harder and harder to exploit because of the more intricate architectures that have been built.

In the 1980s, we can anticipate the fulfillment of many of the promises of VLSI. But with the potential of using vast amounts of hardware for a single computation, the need to organize a computer system to take advantage of algorithms and compilers becomes even more acute than it has been. Thus, algorithm and compiler researchers must provide ideas and direction to computer architects, in order to exploit the potentially even more complex and faster systems that VLSI should deliver.

If we restrict our attention to the speedup of one computation, i.e., decreasing turnaround time in a monoprogrammed system, there are basically only three architectural approaches that have been proposed. Since traditional machines executed one scalar operation at a time, an obvious idea is to execute many scalar operations at once. We will refer to such machines as multiple execution, scalar (MES) systems; the idea is as old as the CDC 6600 and as new as various data flow machine proposals.

*This work was supported in part by the National Science Foundation under Grant No. US NSF MCS76-81686 and in part by the University of California under Contract No. US DOE SBC UCAL 5498609.
Preserving data and control dependences is a problem in such machines, and this can lead to synchronization overheads (time and hardware).

These overheads may be avoided by executing array instructions that sequence large amounts of hardware from a central synchronization point. Single execution, array (SEA) and multiple execution, array (MEA) machines are also common. For example, the CDC CYBER 205 executes single array operations, whereas the CRAY-1 can execute several array operations at once. The array machines usually have the ability to execute one (or a few) scalar operations at the same time as the array operations. While machines in this class are the fastest in existence, they have a narrower range of potential applications that can achieve high speedup than do MES machines.

Of course, various combinations of MES and MEA architectures are possible. Intuitively, it would seem best to use array instructions whenever possible and operate in an MES mode whenever necessary. But designers have relied on intuition too often in the past. It is essential in the future that highly parallel algorithms be studied before a machine to execute them is built. This follows from the fact that besides synchronization overheads, there are potentially severe time penalties due to memory conflicts and data transmission conflicts in high-performance systems.

One way of studying highly parallel algorithms is to automatically transform serial programs into forms that contain many simultaneously executable operations. The Parafrase system, which will be referred to and described in this paper, has been developed over a 10-year period at the University of Illinois for this purpose. It can produce outputs for SEA or MEA (parallel and pipeline) as well as MES machines. We have analyzed over 1,000 Fortran programs and the ideas presented are based on this analysis.

Our goal is to determine what kind of computer system is best suited to a particular class of computations. This paper will not discuss experimental results in much detail, but the overall methodology will be emphasized. Earlier experimental results appear in [KuMC72] and [KBCD74], for parallel machines, and a wide range of new experiments are now in progress.

While our method may not produce the fastest possible computation (and it does not produce fundamentally new algorithms), it generally discovers more simultaneity than humans can. For almost all programs with a significant running time, large speedups can be achieved on one or another of the architectures we consider. It is important to realize that in addition to determining an architecture that is effective for a class of computations, this design method also leads to optimizing compiler transformations in the course of defining the architecture. Thus, we avoid the bottleneck of generating a compiler to suit an exotic architecture, after the machine is designed. This compiler-design bottleneck has arisen in most of the fastest computers for the past 20 years. Five to ten years are often required to design an optimizing compiler that is well-matched to a high-performance machine. Furthermore,