A Hierarchical Parallelizing Compiler for VLIW/MIMD Machines

C. Brownhill and A. Nicolau
University of California at Irvine

Abstract
Hierarchical architectures attempt to provide the benefits of both VLIW/superscalar and MIMD machines by combining multiple VLIW or superscalar processors as parallel, asynchronous processors. An example of these architectures is the Intel Touchstone multicomputer. These machines provide the opportunity to execute a program in parallel at both the machine instruction level and the source statement level.

In this paper, we present some initial results from our research into exploiting hierarchical parallelism with the help of an optimizing compiler. The prototype compiler schedules code for a VLIW/MIMD target architecture. The compiled programs were executed on simulated MIMD and VLIW/MIMD machines. These preliminary results indicate that performing instruction level parallelization on loops that have already been parallelized at the statement level can provide an additional multiplicative speedup in the range of 3.5 to 5.5. Moreover, this factor is independent of the number of VLIW processors used.

1 Introduction
To create fast and efficient parallel programs, it is important to choose the correct task size, or granularity, based on the target architecture. In general, the granularity of the parallelism that can be successfully exploited is directly proportional to synchronization and communication costs. Synchronous machines, such as VLIW (Very Long Instruction
Word) machines, have essentially no communication costs. On these machines, fine grain (instruction level) parallelism can be used to exploit virtually all of the parallelism available at that level. Asynchronous machines, such as shared memory MIMD (Multiple Instruction Multiple Data) machines, incur communication and synchronization costs, but provide a greater flexibility in program control flow, and allow task sizes to vary dynamically among processing units. For these architectures, medium grain (source statement or loop level) parallelism, or even larger, may be necessary, depending on actual machine characteristics. The fine grain parallelism available in the application code is then necessarily neglected.

A hierarchical architecture may be one solution to the synchronous vs. asynchronous trade-off. Machines such as the Intel Touchstone multicomputer attempt to provide the best of both worlds by combining VLIW or superscalar processors together as parallel, asynchronous processors. (For convenience, the term processor will hereafter be used to refer to a VLIW/superscalar processor capable of instruction level parallelism.) Each individual processor has its own memory and multiple functional units which execute in lockstep. In addition, the system may include shared memory, which can be used to implement synchronization between processors. Alternatively, processors may communicate through message passing. This type of hierarchical organization machines provide the opportunity of executing a program in parallel at both instruction level and statement level.

The task of analyzing and exploiting fine grain (for this paper, defined as machine instruction level) and medium grain (defined as source statement level) parallelism has generally been accomplished with the help of parallelizing compilers. Analyzing and scheduling at these levels is usually prohibitively time-consuming and error-prone for programmers. There have been many different techniques used in these compilers. Trace Scheduling [4] and Percolation Scheduling [8] has been used to parallelize non-loop instruction level code. Loop level parallelization techniques include Doacross [3], Dopipe [9], Doall, wavefront methods, and N-Dimensional Perfect Pipelining [5] among others [2, 11, 12]. A variety of loop transformations have been used to exploit instruction level parallelism [1, 6]. To fully exploit the capabilities of a hierarchical architecture, the compiler must be able to exploit multiple levels of parallelism, combining multiple these techniques in a productive way.

Some of the parallelism available in a program may be exploited with either instruction