Instruction Scheduling and Global Register Allocation for SIMD Multiprocessors

Benjamin Hao* and David Pearson**

Department of Computer Science
Cornell University, Ithaca, NY 14853 USA
{bhao,pearson}@cs.cornell.edu

Abstract. Current trends in system design are pointing to using more and more processing units and storage units in a single system. In order to generate programs for these types of distributed memory machines, the challenge is to coordinate and schedule multiple functional units to perform computations efficiently.

In this paper, we describe how our compiler can automate the process and generate good parallel programs from sequential programs. We show how to turn the straight-line code into a task graph which exhibits maximum parallelism possible. Then we give an algorithm for assigning computation to processors to minimize communication cost. Finally, we give an algorithm to allocate registers across processors using an interference graph.

1 Introduction

For SIMD (Single Instruction and Multiple Data) distributed memory multiprocessors, VLIW (Very Long Instruction Word) microprocessors, or ILP (Instruction Level Parallelism) styled machines, to obtain the maximum level of parallelism and the best possible performance, the challenge is to coordinate and schedule the multiple functional (processing) units in the machine to execute desired computations efficiently. Such coordination involves two important tasks: instruction scheduling and global register allocation. Instruction scheduling is the task of mapping computations onto the processing units to achieve maximum parallelism. Global register allocation is the placement of data into each processing unit’s local memories such that all processing units executing the same instruction can execute in lock step with all the data ready in the right registers. Instruction scheduling and global register allocation are interrelated. It is impossible to optimize one without any consideration for the other.


** Supported by a Fannie and John Hertz Foundation Fellowship
Our goal is to automate the entire optimization process via our compiler and free the programmers from such a tedious task. We concentrate on compiling and parallelizing straight line scalar code (SLS code) to run on SIMD multiprocessors (but the techniques developed can be applied to VLIW or any other ILP styled machines with little modifications). We define SLS code as a program segment free of branches, entered at the first statement, and exiting at the last.

We are interested in parallelizing SLS code because: (1) although nested FOR loops are natural for regular problems, irregular problems, large systems of ODE's and so forth can better be handled by synthesized straight-line code for each iteration; (2) statically unrolled loops can give us large basic blocks to explore instruction level parallelism via instruction scheduling across different processing units; (3) removal of conditional branches can be justified in real life through the technique of speculative execution (this technique is often used to keep a high speed super-pipelined RISC processor's pipeline and execution units busy). In other words, we want to keep as many processing units busy as possible to gain maximum parallelism from a sequential program.

We chose to target our compiler for SIMD machines to demonstrate our instruction scheduling and global register allocation algorithms. SIMD was chosen because: (1) SIMD machines have thus far only proved useful for regular problems, but our techniques promise to effectively utilize them on irregular problems as well; (2) they have much more processing and storage units than any other parallel or VLIW processors, hence it is more challenging to manage all of them; (3) they require our algorithms and solutions to be truly scalable to take advantage of all the units; (4) the local memory per node in a SIMD machine is usually very small, thus requires very efficient register allocation.

In this paper, we will present how our compiler performs register allocation and instruction scheduling to generate parallel code from a normal SLS program. But, first we will describe related instruction scheduling and global register allocation studies, and how they are different from ours.

2 Prior and Related Work

Instruction scheduling is very important for machines with multiple functional units. Most of the recent studies concentrate on speculative execution, the execution of instructions before the conditional branch instruction before them is resolved [1, 2, 3], on VLIW machines. The main goals of these studies are to have superscalar processors looking beyond branch boundaries to achieve more ILP in non-numeric applications, and to be able to recover gracefully from exceptions caused by speculative instructions, instructions being executed during speculative execution. The goal of our compiler-controlled instruction scheduling is, however, to attain maximum possible ILP on SIMD while minimizing overall communication cost.

Recent research on global register allocation [4, 5] focuses on problems of improving existing standard register allocators via clever data structures and integrating these new data structures into other parts of the compiler, especially