A Control-Parallel Programming Model Implemented on SIMD Hardware

H. Dietz and W. Cohen
Purdue University, West Lafayette, Indiana

Abstract

Although "data parallelism" has been shown to be an effective and portable way to express some types of parallel algorithms, there are many other problems for which data parallelism seems awkward and inefficient. For example, recursive decompositions and operations on irregular grids are most readily expressed using control parallelism. The problem is that control parallelism has always been associated with MIMD (Multiple Instruction stream, Multiple Data stream) hardware. In this paper, we describe how to make a MIMD programming model execute efficiently on a SIMD (Single Instruction stream, Multiple Data stream) computer.

The efficient execution of control-parallel code on a SIMD machine involves a careful blend of compiler technology and design and semi-automatic construction of the support routines (i.e., the MIMD emulator). This paper discusses how the techniques were applied to give the appearance of a 16,384-processor shared memory barrier MIMD using the hardware of a SIMD MasPar MP-1.

1. Introduction

The differences between data parallelism (SIMD execution) and control parallelism (MIMD execution) are at least superficially quite large. In a data parallel program, parallelism is specified in terms of performing the same operation simultaneously on all elements of a data structure; this naturally fits the SIMD execution model. It is also easy to see that, because the abilities of a MIMD are a superset of the abilities of a SIMD, the data parallel model can be used for MIMD targets. However, the control parallel model suggests that each processor can take its own path independent of all others, and this characteristic seems to require the multiple instruction streams possible only in MIMD execution. Control parallelism is impossible on a SIMD with only one instruction stream... or is it?
If efficiency is not the primary concern, there is nothing particularly difficult about writing a SIMD program that interprets a MIMD instruction set. For example, [20] reported on a simple MIMD interpreter running on a MasPar MP-1 [1]. Wilsey, et. al, implemented an interpreter for the MINTABS instruction set and indicated that work was in progress on a similar interpreter for the MIPS R2000 instruction set. The MINTABS instruction set is very small (only 8 instructions) and is far from complete in that there is no provision for communication between processors, but it does provide basic MIMD execution. In fairness to [20], their MIMD interpreter was built specifically for parallel execution of mutant versions of serial programs — no communication is needed for that application.

Such an interpreter has a data structure, replicated in each SIMD PE, that corresponds to the internal registers of each MIMD processor. Hence, the interpreter structure can be as simple as:

**Basic MIMD Interpreter Algorithm**

1. Each PE fetches an "instruction" into its "instruction register" (IR) and updates its "program counter" (PC).
2. Each PE decodes the "instruction" from its IR.
3. Repeat steps 3a-3c for each "instruction" type:
   a) Disable all PEs where the IR holds an "instruction" of a different type.
   b) Simulate execution of the "instruction" on the enabled PEs.
   c) Enable all PEs.
4. Go to step 1.

The only difficulty in implementing an interpreter with the above structure is that the simulated machine will be very inefficient. There are several reasons for this inefficiency.

1.1. Interpretation Overhead

The most obvious problem is simply that interpretation implies some overhead for the interpreter; even MIMD hardware simulating a MIMD with a different instruction set would suffer this overhead. In addition, SIMD hardware can only simulate execution of one instruction type at a time, hence, the time to execute a simulated instruction is proportional to the sum of the execution times for each instruction type.

1.2. Indirection

Still more insidious is the fact that even step 1 of the above algorithm cannot be executed in parallel across all PEs in many SIMD computers. The next instruction for each PE could be at any location in the PE's local memory, and many SIMD machines do not allow multiple PEs to access different memory locations simultaneously. Hence, on such a SIMD machine, any parallel memory access made will take time proportional to the number of different PE addresses being fetched from or, worse still, time proportional to the size of the address space which could be accessed. For example, this is the case on the TMC CM-1 [9] and TMC CM-2 [19]. Note that step 3b suffers from the same problem if load or store operations must be performed.

Since many operations are limited by (local) memory access speed, inefficient handling of these memory operations can easily make MIMD interpretation on a SIMD machine infeasible. However, this overhead can be averted if the SIMD hardware can indirectly access memory using an address in a PE register. Examples of SIMD machines with such hardware include the PASM Prototype [16] and the MasPar MP-1 [1].