ABSTRACT

Vector architectures have long been the architecture of choice for numerical high performance computing. Their large memory bandwidth and the ability to tolerate relatively long memory latencies have been central to their success in executing numerical applications with very large data sets.

Recently, though, vector processors have been slowly displaced by other supercomputing paradigms based on large or medium scale multiprocessing that use as a building block commodity superscalar CMOS microprocessors. Vector machines have not improved fast enough to keep a good price/performance ratio and have been relegated to those types of problems requiring very large bandwidths and/or very large datasets. Moreover, the performance achieved by real programs on vector machines has usually been far from peak speed. This has inclined users to believe that the extreme cost of vector machines is not justified in front of more affordable microprocessor based machines.

The difficulty in bringing vector supercomputers back to a competitive price/performance point is in their memory systems, which, although provide the uttermost performance, are extremely expensive. The use of many interleaved parallel banks and static memory storage leads to very high bandwidths at a reasonable latency but incurs in large system costs. In order to reduce the cost of vector supercomputers, the memory system should be made more economical by moving from static to dynamic storage technology. Besides, vector machines should also provide increased performance to clearly distinguish themselves from commodity superscalar micros.

These two goals are somewhat in contradiction since moving from static to dynamic storage could double or triple memory latency due to the intrinsically larger response time of dynamic memory. Although the vector paradigm does provide some latency

tolerance, we will start by showing that it is not enough for truly large latencies and that a vector processor would suffer a severe performance degradation under such conditions.

This thesis will show that both the lack of tolerance to very large memory latencies and the discrepancy between actual and peak performance of vector machines originate in the conservative in-order instruction execution model currently in use. This in-order model results in a large underutilization of the high memory bandwidth provided by the memory system. The in-order model is also responsible for not requesting data early enough in time to mask large memory latencies.

Three solutions to the latency and performance problems will be presented. Using dynamic scheduling techniques already exploited in superscalar machines, we will show that performance of traditional vector architectures can be greatly improved while, at the same time, providing the necessary latency tolerance to compensate for slow memory systems. The first two techniques, *decoupling* and *out-of-order execution*, look for independent operations of a singe program stream to be executed in parallel. The third technique, *multithreading*, will improve global throughput by interleaving independent instructions from different programs.

Alone or combined, these three techniques can yield better performance and allow the use of cheaper main memory systems which can make vector architectures regain some of their lost prominence.

Spill code is an important problem in architectures with a limited number of registers. Vector machines have traditionally been built using only a few number of registers and, therefore, experience this spill problem. This thesis will also study the amount of spill code present in vector programs and will present three different techniques that dynamically eliminate spill loads and stores, thereby reducing total memory traffic and improving execution time.

Our goal is to present solutions, both to the latency and spill problems, that are fully binary compatible with existing vector instruction sets. Thus, the techniques described

in this thesis are fully transparent to the end user and do not require recompilation nor the addition of special opcodes to the instruction set architecture.