Instruction Scheduling on the M-Machine Multicomputer*

Marco Fillo fillo@ai.mit.edu
Stephen W. Keckler skeckler@ai.mit.edu
Nicholas P. Carter npcarter@ai.mit.edu
Andrew Chang achang@ai.mit.edu
Yevgeny Gurevich yev@ai.mit.edu
Whay S. Lee wslee@ai.mit.edu

MIT Artificial Intelligence Laboratory

Introduction

The M–Machine is an experimental multicomputer being developed to test architecture concepts which are motivated by the constraints of modern semiconductor technology and the demands of programming systems, such as faster execution of fixed sized problems and easier programmability of parallel computers.

Advances in VLSI technology have resulted in computers with chip area dominated by memory and not by processing resources. To address this imbalance, the M–Machine increases the fraction of chip area devoted to processor, to make better use of the critical memory resources. An M–Machine multi-ALU processor (MAP) chip contains four 64-bit three-issue *clusters* that comprise 32% of the 5Gλ^2 chip and 11% of an 8 MByte (six-chip) node.

The M–Machine is designed to extract more parallelism from problems of a fixed size, rather than requiring enormous problems to achieve peak performance. To do this, nodes are designed to manage parallelism from the instruction level to the process level. The 12 function units in a single M–Machine node are controlled using a form of Processor Coupling [1] to exploit instruction level parallelism by executing 12 operations from the same thread, or to exploit thread-level parallelism by executing operations from up to six different threads. The fast internode communication allows collaborating threads to reside on different nodes.

This abstract briefly describes the execution of instructions across the clusters. A more detailed discussion of the M-Machine can be found in [2].

Instruction Scheduling

The amount and granularity of parallelism varies enormously across application programs and even during different phases of the same program. Some phases have an abundance of instruction level parallelism that can be extracted at compile time. Others have data dependent parallelism that can be executed using multiple threads with widely varying task sizes.

The M–Machine is designed to efficiently execute programs with any or all granularities of parallelism. On the MAP, parallel instruction sequences (*H–Threads*) are run concurrently on the four clusters to exploit ILP across all 12 of the function units. Alternatively they may be used to exploit loop level parallelism. To exploit thread-level parallelism and to mask variable pipeline, memory, and communication delays, the MAP interleaves the 12-wide instruction streams from different tasks (*V–Threads*) within each cluster on a cluster-by-cluster and cycle-by-cycle basis, thus sharing the execution resources among all active tasks.

This arrangement of V–Threads (Vertical Threads) and H–Threads (Horizontal Threads) is summarized in Figure 1. Six V–Threads are resident in the cluster register files. Each V–Thread is composed of four H–Threads, one on each cluster. Each H–Thread consists of a sequence of 3-wide instructions containing integer, memory, and floating point operations. On each cluster the H–Threads from the different V–Threads are interleaved over the execution units.

H–Threads

An *H–Thread* runs on a single cluster and executes a sequence of operation triplets (one operation for each of the 3 ALUs in the cluster) that are issued simultaneously. Within an H–Thread, instructions are guaranteed to issue in order, but may complete out of order. An H–Thread may communicate and synchronize via registers with the 3 other H–Threads in the same V–Thread, each executing on a separate cluster. Each H–Thread reads operands from its own register file, but can directly write to the register file of any H–Thread in its own V–Thread.

H–Threads enable multiple execution models. They can execute as independent threads with possibly different control flows to exploit loop-level or thread-level

*This research is supervised by W.J. Dally and supported in part by the Advanced Research Projects Agency and monitored by the Air Force Electronic Systems Division under contract F19628-92-C-0045 and in part by the Air Force Office of Scientific Research under contract F49620-94-1-0462.

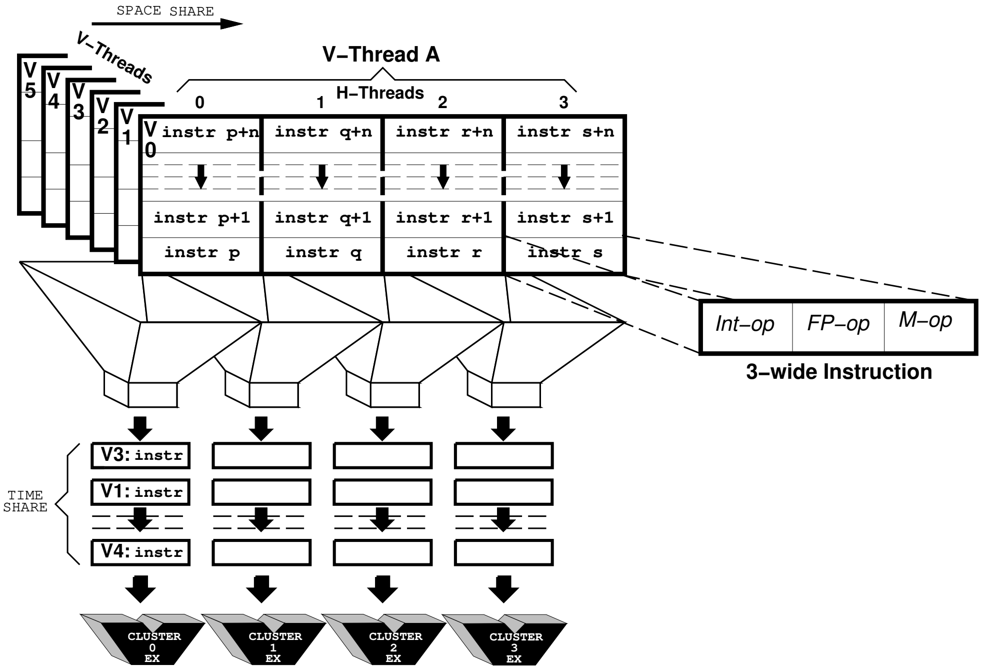

Figure 1: Multiple V-Threads are interleaved dynamically over the cluster resources. Each V–Thread consists of 4 H–Threads which execute on different clusters.

parallelism. Alternatively, the compiler can schedule the four H–Threads in a V–Thread as a unit to exploit instruction level parallelism, as in a VLIW machine. In this case the compiler must insert explicit register-based synchronization to enforce instruction ordering between H–Threads. Unlike the lock-step execution of traditional VLIW machines, H–Thread synchronization occurs infrequently, only when required by data dependencies. While explicit synchronization incurs some overhead, it allows H–Threads to slip relative to each other in order to accommodate variable–latency operations such as memory accesses.

H–Thread Synchronization

Inter-cluster data transfers require explicit register synchronization. To prepare for inter-cluster data transfers, the receiving H–Thread executes an EMPTY operation to mark empty a set of destination registers. As each datum arrives from the transmitting H–Thread (over the inter-cluster switch), the corresponding destination register is set full. An instruction in the receiving H–Thread that uses the arriving data will not be eligible for issue until its data is available.

Four pairs of single-bit global condition code (CC) registers are used to broadcast single bit values across the clusters. Unlike centrally located global registers, the MAP global CC registers are physically replicated on each of the clusters. A cluster may broadcast using either register in only one of the four pairs, but may read and empty its local copy of any global CC register. Using these registers, all four H–Threads can execute conditional branches and assignment operations based on a comparison performed in a single cluster.

References

[1] S. W. Keckler and W. J. Dally, "Processor coupling: Integrating compile time and runtime scheduling for parallelism," in *Proceedings of the 19th International Symposium on Computer Architecture*, (Queensland, Australia), pp. 202–213, ACM, May 1992.

[2] M. Fillo, S. W. Keckler, W. J. Dally, N. P. Carter, A. Chang, Y. Gurevich, and W. S. Lee, "The M-Machine Multicomputer," AI Memo 1532, Massachusetts Institute of Technology Artificial Intelligence Laboratory, Cambridge, MA 02139, 1995.