

Instruction Level Parallelism

Robert Yung
Sun Microsystems Inc.
yung@sun.com
(415) 336-5954

In the quest for higher performance, state-of-the-art processor designs currently employ various techniques to exploit and increase instruction level parallelism. Some of these optimizations are attempted at compile time while others are handled in hardware during program execution. In general, these hardware or software techniques exploit instruction level parallelism by increasing instruction dispatch rate, by reducing control transfer overhead, and by reducing average operand access latency.

The rate at which instructions can be dispatched is largely affected by the nature of the program and the efficiency of instruction scheduling. Scientific programs generally contain much instruction parallelism and few control transfers. For these programs, the instruction dispatch rate is usually limited by hardware resources such as the number of execution units and available memory bandwidth. In contrast, an integer or commercial program contains many data dependent control transfers. Unless correctly predicted and properly handled, data dependent control transfers reduce the efficiency of instruction scheduling and in turn reduce instruction dispatch rate.

Control transfer overhead can be minimized by speculative execution and branch reduction techniques. Speculative execution is a technique that allows execution of instructions following a branch prior to its resolution while maintaining the sequential execution model. Branches can be reduced by conditional execution of instructions following them. For example,

- trace scheduling is a compile time technique that reschedules a program based on branch statistics collected from a sample run. The drawback of such

a scheme is the potential larger code size and its reliance on a representative sample.

- conditional, guarded, or predicated execution may be used to eliminate an if-then-else decision branch by conditionally executing one or more instructions after the branch. This technique requires support via a new instruction set architecture (ISA) or changes in an existing one. Besides the compatibility issue, the merit of these techniques is not conclusive in running real world programs. Generally, conditional branches are difficult to eliminate in integer and commercial programs.
- loop unrolling is a compile time technique that eliminates loop-terminating branches by concatenating multiple iterations of the loop in the code. This is a common technique used in vectorizable numeric programs.
- dynamic branch prediction is a general hardware technique that predicts the outcome of branches based on the history of previous branches. In general, dynamic prediction techniques achieve very good hit rates. Their disadvantages are increased hardware complexity and increased die area.

These techniques can be applied at compile time by software or handled during program execution by hardware.

As integrated circuit process technology advances, there is a growing gap between the processor and the memory sub-system. Memory access latency can be reduced with a larger set of programmer visible registers, higher cache hit rate, faster memory access, and by software techniques such as loop unrolling and software pipelining. Memory access overheads can be reduced by increasing the

distance between a memory access and its dependent instruction.

- One or more levels of caches have been employed to reduce the average memory access latency. Caches work well when there is a high locality of references. Most integer and commercial programs exhibit both temporal and spatial locality of references. A fully- or set-associative cache has a better hit rate than a direct-mapped cache, but the associative cache has a larger access time and takes up more area.
- Loop unrolling and software pipelining are compile time techniques that increase the load-use distance between a load and its dependent instructions. The limitation is that they require a large set of programmer visible registers. Windowed register files have been used to increase the number of addressable registers in a register file. Hardware register renaming can also be used in conjunction with these techniques to reduce the register bottleneck. This is done by remapping the programmer-visible register set to a larger set of physical registers implemented in a design.

Today processor designs employ these and other techniques to obtain higher performance. The main differences in these different designs lies in the hardware and software interface.

- Superscalar and dataflow designs place heavy emphasis on hardware for efficient instruction scheduling. A compiler first compiles a program into a sequence of instructions. After the instructions are fetched into the processor, the dataflow information is rediscovered by elaborate hardware. This information is used to schedule hardware resources and resolve data dependencies in the program during execution. Because the complexity for instruction scheduling is $O(n^2)$, a high instruction issue rate becomes increasingly difficult to realize with a fast cycle time.
- A Very Long Instruction Word (VLIW) design exposes the underlying hardware resources and constraints to the compiler. Instruction scheduling is handled entirely in software. The advantage of this design approach is potentially simple hardware which may lead to faster cycle time. Its disadvan-

tages are increased software complexity, binary incompatibility, and reduced ability to handle dynamic situations such as cache misses during program execution.

In this architecture track, we will examine these techniques and designs that exploit instruction level parallelism in a program.

Acknowledgment

This minitrack will not be possible without the help of John Fu, Scott Mahlke, Trevor Mudge, Nancy Warter and the referees. Thanks also go to Renee Beauvais, Dennis Duprie, Dave Roberts and Neil Wilhelm for the administrative supports.