# Instruction-level Parallelism

Loic Niragire

School of Technology, Northcentral University

TIM 8121: Distributed Algorithms and Parallel Computing

Dr. Carol Cusano

April 16th, 2022

Instruction-level parallelism achieves computing efficiency by rearranging instruction's execution sequence to maximize hardware utilization. Rather than a top-down execution, the CPU executes multiple independent instructions in parallel to reduce idle time. Modern CPUs contain multiple computing cores capable of processing billions of instructions per second. Multithreading improves computing efficiency by executing tasks into multiple threads of execution concurrently or in parallel across processors. Moreover, the CPU achieves further efficiency within a thread of execution by pipelining and executing multiple instructions in parallel, using a combination of software and hardware techniques such as instruction pipeline, superscalar execution, out-of-order execution, register renaming, and speculative execution.

## Instruction pipeline

Instructions designed for pipeline execution have pre-determined ordered steps that each take an equal amount of clock cycles. For instance, MIPS instructions have five steps; fetch instruction from memory, read registers while decoding the instruction, execute the operation, access an operand in data memory, and write the result into a register. Thus, after the compiler translates a given program into a set of instructions, each instruction must execute all pre-defined steps before starting the execution of the next instruction. Therefore, by overlapping execution sequences, the CPU aims to maximize hardware utilization while improving instruction throughput. For example, table one depicts a static two-issue MIPS pipeline with the five stages described above. A non-pipelined execution of the same instructions would first execute all five stages for the *load word* instruction, followed by the same stages for the *store word* instruction, then the *branch* instruction.

| *Instruction type* | *Pipe stages* | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| *ALU* | IF | ID | EX | MEM | WB | | | |
| *Load* | IF | ID | EX | MEM | WB | | | |
| *ALU* | | IF | ID | EX | MEM | WB | | |
| *Load* | | | IF | ID | EX | MEM | WEB | |
| *ALU* | | | | IF | ID | EX | MEM | WEB |
| *Load* | | | | IF | ID | EX | MEM | WEB |

*Table 1 MIPS static two-issue pipeline*

The hardware utilization that yields the ILP performance improvement stems from the observation that a given instruction class does not need to wait for the completion of its predecessor, where completion indicates the execution of all pre-determined stages. Thereby, the CPU starts the execution of the next instruction as soon as the required computing facility is released by the previous instruction. Therefore, pipelining has a speed-up of approximately the number of stages, assuming the stages are perfectly balanced (Patterson & Hennessy, 2013). Hence, one way to increase the potential of instruction parallelism is to increase the depth of the pipeline to overlap more instructions.

Cases, where the next instruction in a pipeline cannot be executed in the following clock cycle are known as hazards. A structural hazard, for instance, indicates that the hardware cannot support the combination of instructions that we want to execution in the same clock cycle. Meanwhile, a data hazard is the result of a data dependency between instructions. Whereby the pipeline is forced to stall because one step must wait for another to complete. The last hazard is known as a control or branch hazard, which occurs when the

proper instruction cannot execute in the appropriate pipeline clock cycle because of fetching the wrong instruction. These hazards form the primary limitations of how much ILP can be exploited. Scheduling techniques and speculation via prediction, both in the hardware and software, are used to ease these limitations.

Processors employ a combination of software and hardware-based approach to exploit ILP. The x86 microprocessors, for instance, use both dynamic multiple-issue and dynamic pipeline scheduling with out-of-order execution and speculation on a 14-stage pipeline (Patterson & Hennessy, 2013). Meanwhile, the ARM Cortex-A8 processor uses multiple-issue with two instructions per clock cycle, using a 14-stage static in-order pipeline. Although pipelining improves throughput, it does not decrease instruction latency, where latency is defined as the number of clock cycles elapsed between two dependent computations. At the same time, the throughput of an instruction is defined as the maximum number of instances of that instruction that can be executed in parallel in one cycle. Table two categorizes different ILP techniques as software and/or hardware-based techniques.

| ***Technique*** | **Software-based** | **Hardware-base** |
|---|---|---|
| *Superscalar* | False | True |
| *Register renaming* | True | True |
| *Out-of-order execution* | False | True |
| *Speculative execution* | True | True |
| *Multiple-issue* | True | True |
| *VLIW* | True | False |
| *Dynamic scheduling* | False | Hardware |

*Table 2 Categorization of ILP techniques*

## Superscalar execution

Processors that can decide how many instructions to issue in each clock cycle are known as superscalar processors or simply superscalars. Furthermore, these processors may extend their functionalities to include which instructions to issue in a clock cycle through a process known as *dynamic pipeline scheduling*, which involves analyzing the data flow structure of a program. The processor then decides whether to execute the instructions in an order that preserves the data flow of the program.

In a superscalar execution, the CPU decomposes the instructions to execute into one or more micro-operations, µOps, which are then scheduled on execution ports. A process that requires the construction of resource mapping describing resources used by each instruction (Derumigny, et al., 2022). An example of such mapping could be a graph describing the decomposition of instructions into µOps and the assignment of these µOps to execution ports.

## Register renaming

To exploit ILP, register renaming solves false dependency hazards dynamically. The compiler provides the software-centric approaches by introducing additional registers to eliminate dependencies that are not true data dependencies but could lead to potential hazards. Processors that rely on the compiler to assist with hazard handling and instruction packaging are known as *static multiple-issue* processors, which allow multiple instructions per clock cycle. VLIW processors fall under this category. In comparison, hardware-centric mechanisms require a register mapping table, RMT, that associates logical register numbers with physical register number.

There are two limitations associates with ILP through register renaming. It requires a register mapping table and has complicated recovery mechanism in the event of an exception

such as branch misprediction (Mitsuno, et al., 2020). A considerable amount of resources is required to implement a multi-port register mapping table that is accessed frequently. Additionally, the RMT needs to be restored when an exception occurs, which causes an increased hardware complexity. Ongoing research aim to eliminate the need for register renaming by resolving false dependencies statically at compiler time.

## Speculative execution

By letting the processor or compiler make a guess about the properties of an instruction, subsequent instructions can begin executing earlier. However, this strategy requires the implementation of a recovery mechanism to counter wrong guesses. Additionally, it is prone to introducing exceptions that would not have occurred in a normal execution. Therefore, speculation has the potential to decrease performance when not done correctly.

Task speculative execution is used in MapReduce to improve response time by running poorly performing nodes, called outliers, on a separate machine (Xu & Lau, 2013). Hadoop also implements speculative execution to improve performance by using a strategy known as *Longest Approximate Time to End* which speculates whether to run a backup copy based on a task's remaining time to finish.

CPU vulnerabilities can be exploited by tricking the processor into speculatively executing the wrong instruction sequence and exposing sensitive data through branch or memory speculation. (Li, Zhao, Hou, Zhang, & Meng, 2019). This is largely because most rollback processes invoked in the recovery from the wrong speculation do not fully revert cache contents. Thus, exposing the CPU to cache-timing attacks.

## Out-of-order execution

Superscalar processors can override the sequential model imposed by the application and rearrange instruction execution order based on input availability and execution units to improve throughput performance. It is an execution strategy that relies on the data flow graph rather than the compiler scheduled order. Combining out-of-order execution and speculative precomputation can improve memory latency in processors by effectively hiding L1 cache misses and reducing L3 penalty (Wang, et al., 2002). Out-of-order superscalars allow instruction execution to continue while a pipelined L2 cache handles L1 cache misses.

An out-of-order execution machine schedules operations that are independent of cache misses by using an instruction and scheduling window. The instruction window maintains decoded instructions in their original order to assert successful completion. At the same time, the scheduling window contains the logic to distinguish dependent and independent instructions. Therefore, out-of-order processors are limited by the size of their instruction window.

Out-of-order execution is used by high-performance processors to tolerate extended latency operations by buffering the operations in an instruction window. There are other hardware and software prefetching techniques used to tolerate long memory latencies since solely relying on out-of-order execution increases design complexity and power consumption (Mutlu, Stark, Wilkerson, & Patt, 2002). Table three illustrates a twenty-three-year span of microprocessor design from 1989 to 2012. Notice that the Intel Pentium 4 Willamette has the highest power consumption at 103 watts and the deepest pipeline with 31 stages. In comparison, the Intel Core *i*5 Ivy Bridge manages to run a 14-pipeline stage with eight cores at 77 watts.

| ***Microprocessor*** | **Year** | **Clock rate** | **Pipeline stages** | **Out-of-order** | **Cores** | **Power** |
|---|---|---|---|---|---|---|
| *Intel 486* | 1989 | 25MHz | 5 | No | 1 | 5w |
| *Intel Pentium* | 1993 | 66MHz | 5 | No | 1 | 10w |
| *Intel Pentium Pro* | 1997 | 200 MHz | 10 | Yes | 1 | 29w |
| *Intel Pentium 4 Willamette* | 2001 | 2000 MHz | 22 | Yes | 1 | 75w |
| *Intel Pentium 4 Prescott* | 2004 | 3600 MHz | 31 | Yes | 1 | 103w |
| *Intel Core* | 2006 | 2930 MHz | 14 | Yes | 2 | 75w |
| *Intel Core i5 Nehalem* | 2010 | 3300 MHz | 14 | Yes | 1 | 87w |
| *Intel Core i5 Ivy Bridge* | 2012 | 3400 MHz | 14 | Yes | 8 | 77w |

*Table 3 Microprocessors pipeline power consumption*

## References

Derumigny, N., Gruber, F., Iooss, G., Guillon, C., Pouchet, L.-N., Bastian, T., & Rostello, F. (2022). PALMED: Throughput Characterization for Superscalar Architectures. 2022 IEEE/ACM International Symposium on Code Generation and Optimization (pp. 106-117). IEEE.

Li, P., Zhao, L., Hou, R., Zhang, L., & Meng, D. (2019). Conditional Speculation: An Effective Approach to Safeguard Out-out-order execution against Spectre Attacks. 2019 IEEE Internation Symposium on High Performance Architecture (HPCA) (pp. 264-276). IEEE.

Mitsuno, S., Kadomoto, J., Koizumi, T., Shioya, R., Irie, H., & Sakai, S. (2020). A High-Performance Out-of-Order Soft Processor Without Register Renaming. 2020 30th International Conference on Field-Programmable Logic and Applications (pp. 73-78). IEEE.

Mutlu, O., Stark, J., Wilkerson, C., & Patt, Y. N. (2002). Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors. The Ninth International Symposium on High-Performance Computer Architecture (HPCA) (pp. 129-140). IEEE.

Patterson, A. D., & Hennessy, L. J. (2013). An overview of Pipelining. In A. D. Patterson, & L. J. Hennessy, Computer organization and design - fith edition (pp. 272-286). Waltham, MA: Morgan Kaufmann.

Wang, P. H., Wang, H., Collins, J. D., Grochowski, E., Kling, R. M., & Shen, J. P. (2002). Memory Latency-Tolerance Approaches for Itanium Processors: Out-of-Order

Execution vs. Speculative Precomputation. Proceedings Eight International Symposium on High-Computer Architecture (HPCA) (pp. 187-196). IEEE.

Xu, H., & Lau, C. W. (2013). Resource Optimization for Speculative Execution in MapReduce Cluster. 2013 21st IEEE International Conference on Network Protocols (ICNP) (pp. 1-3). IEEE.