Assignment 4\_Part 1: ILP **Date: 27th OCTOBER 2024**

Shafan Nazeer Ahmed

005030047

### **Introduction**

Instruction Level Parallelism (ILP) has been a critical factor in the development of current computer architectures. In order to satisfy the increasing requirements of a variety of applications, ILP has significantly improved computational performance by allowing processors to execute multiple instructions simultaneously. ILP methodologies have developed over the years, incorporating increasingly sophisticated approaches to optimize processor efficiency. Nevertheless, new opportunities and challenges arise as we approach the physical and practical limits of traditional ILP implementations. The historical development of ILP its core concepts and limitations, performance metrics, current challenges, and novel approaches and future directions in the field are all explored in this review.

Between the late 1980s and early 1990s, superscalar architectures grew out of pipelining. These designs added more than one execution unit to a single CPU, which let the processor send and carry out more than one instruction per clock cycle. This development necessitated sophisticated instruction fetch and decode units that could manage multiple instructions and resolve dependencies in real-time.  
  
Another milestone that addressed the constraints imposed by instruction dependencies and latency variations was out-of-order execution. Processors could optimize the utilization of available resources and minimize the idle times of execution units by permitting instructions to execute immediately upon the completion of their operands regardless of their original order. To make sure the program was correct this method needed complicated hardware like reservation stations and reorder buffers to be put in place.  
  
In order to mitigate control hazards resulting from conditional branches advanced branch prediction techniques and speculative execution were implemented. Processors could ensure a consistent flow of instructions through the pipeline by anticipating the results of branches and executing instructions in advance. The design was further complicated by the necessity of mechanisms to roll back and correct the processor state in response to inaccurate predictions.  
  
These advancements were motivated by the necessity to maintain performance enhancements in the presence of Moore's Law's anticipated slowdown in advancements and the discontinuation of Dennard scaling. The number of transistors on a chip would double approximately every two years, resulting in performance gains as Moore's Law observed. Dennard scaling proposed that the power density would remain constant as transistors became smaller, thereby enabling the use of higher frequencies without an increase in power consumption. Architects were compelled to pursue alternative methods, such as ILP, in order to enhance performance, as a result of the slowing of these trends (Dally, Turakhia, & Han, 2021).

**Core Concepts and Limitations of ILP**

By finding separate operations in a program, ILP takes advantage of the fact that it is possible to run multiple instructions at the same time. The main things that make ILP possible are listed below:-  
  
The breaking up of the instruction execution process into separate stages that can handle different instructions at the same time at each stage.  
  
The use of multiple execution units to issue and execute numerous instructions per clock cycle.  
  
The process of dynamically reordering instructions to execute them as soon as their operands are available, rather than adhering to the strict order of the program.  
  
The process of predicting the paths of branches in order to execute instructions in advance, thereby enhancing pipeline utilization.  
  
ILP is restricted by numerous inherent limitations despite the implementation of these sophisticated techniques:  
  
 Parallel execution becomes difficult when instructions are contingent upon the outcomes of previous instructions. There are three categories of data dependencies:  
  
True dependencies in which an instruction necessitates a value that a preceding instruction generates.  
Anti-dependencies in which an instruction overwrites a location that was previously read by a previous instruction.  
Output dependencies that occur when two instructions write to the same location.  
  
Originate from branch instructions that modify the execution sequence. The unpredictability of branches can result in pipeline stalls or the necessity of flushing incorrectly speculated instructions.  
  
The capacity to execute multiple instructions in parallel can be impeded by the limited memory bandwidth and execution units.

**Performance Metrics**

Discussing on the ILP performance, there are several performance matrices that are introduced in this scenario. Major performance matrices like throughput have been used to measure the number of instructions executed per cycle. Higher throughput ensures the better handling of multiple instructions concurrently by the processor. Another metric is latency where the total amount of time taken for a single or a sequence of instructions executing can be determined. Reduced latency is required to complete individual task execution. Another metric is Cycles Per Instruction (CPI) which calculates the number of clock cycles required to process an instruction. Here also, the lower CPI indicates efficient ILP exploitation. Apart from that, matrices like power consumption and determination of frequency are used to identify the performance of ILP.

**Current Challenges**

In recent years, the architecture of the processors has rapidly evolved. There are several significant challenges that come with designing processors to achieve ILP. Most of these challenges come from the increasing complexities related to hardware and micro-architectural approaches where recent research approaches like clustered microarchitectures, and optimization of hardware components dynamically help overcome these sorts of challenges. Another challenge is to limit the utilization of power consumption where approaches like Dynamic Voltage and Frequency Scaling (DVFS), Heterogeneous Multicore Architectures etc. are very helpful in mitigating these sorts of challenges. There are also limitations related to branch prediction, pipeline stalls and management of hazards where value prediction, bypassing of loads, and hybrid and machine learning-based branch predictors are proposed to overcome those limitations.

**Future Directions**

With the help of emerging technologies and computer architectural innovations, researchers engage themselves to create futuristic ILP to extend its effectiveness. They proposed heterogeneous architectures like ARM’s big.LITTLE design which is very high performing and energy efficient and has the ability to allow processors to adapt workload requirements dynamically (Mascitti, 2020). Additionally, the incorporation of specialized processing units like GPUs, AI accelerators etc. is used to offload workload that enhances ILP. On the other hand, using machine learning-based optimization, domain-specific architecture, 3D stacking and advanced memory technologies, clustered microarchitecture etc. helps ILP to increase its effectiveness. All of these sorts of promising architectural approaches focus on growing complexities, diverse workloads, managing power constraints and integration of emerging technologies to help in significant ILP evolution with a promising future direction.

**References**

Borkar, S., & Chien, A. A. (2011). The future of microprocessors. *Communications of the ACM*, 54(5), 67–77. <https://doi.org/10.1145/1941487.1941507>

Chien, A. A., & Borkar, S. (2021). Emerging trends in computer architecture. *Communications of the ACM*, 64(3), 93–102. <https://doi.org/10.1145/3446382>

Dally, W. J., Turakhia, Y., & Han, S. (2021). Domain-specific hardware accelerators. *Communications of the ACM*, 63(7), 48–57. <https://doi.org/10.1145/3418293>

Hennessy, J. L., & Patterson, D. A. (2019). *Computer Architecture: A Quantitative Approach* (6th ed.). Morgan Kaufmann.