Data flow Pre-Scheduling in Out of Order Processors

Abhigna Bheemineni, Manikandeshwar Sasidhar, Nishka Sathisha, Swetha Chiliveri
Department of Electrical and Computer Engineering, Portland State University.

Abstract – This paper introduces novel approaches to enhance the performance of out-of-order processors by utilizing advanced data-flow prescheduling techniques. By exploiting parallelism and dependencies in program execution, we build upon previous work and present a comprehensive exploration of a novel strategy. This strategy optimizes instruction dispatch, reducing resource bottlenecks and execution time. We also investigate the impact of various factors, such as instruction mix, memory access patterns, and branch behavior, on the effectiveness of our techniques. Through extensive simulations on diverse benchmark suites, our experimental results demonstrate significant performance gains. This research contributes to the optimization of out-of-order processor performance, providing valuable insights into the practical applicability and benefits of data-flow prescheduling. Our proposed algorithm leverages sophisticated data-flow analysis and techniques, maximizing the utilization of hardware resources and advancing the field.

I. INTRODUCTION

Modern computing systems strive to deliver high-performance capabilities to meet complex computational workload demands. Out-of-order processors, characterized by their ability to dynamically rearrange instructions to maximize resource utilization, have emerged as a vital component in efficient program execution. One crucial aspect of out-of-order processors is the efficient utilization of large instruction windows, which allows for extensive instruction-level parallelism. In this paper, we aim to extend and enhance the existing body of knowledge on dataflow prescheduling for large instruction windows in out-of-order processors. By leveraging advanced data-flow analysis and techniques, we strive to maximize hardware resources and improve overall execution performance.

To accomplish this, we propose a scheduling algorithm that optimizes the scheduling of instructions within the instruction window. By prioritizing instruction dispatch based on their data dependencies and considering the dynamic nature of program execution, we aim to minimize potential resource bottlenecks. We also aim to reduce execution latency. Furthermore, we investigate the impact of various factors, such as instruction mix, memory access patterns, and branch behavior, on the effectiveness of our proposed techniques.

Our research stems from the increasing complexity of modern workloads, which demand efficient out-of-order processor utilization. As programs become more intricate and data-intensive, the ability to exploit parallelism and manage data dependencies becomes crucial for high performance computing. By enhancing data-flow prescheduling techniques for large instruction windows, we aim to unlock increased performance potential and enable more efficient execution of diverse workloads. Through comprehensive analysis, we demonstrate the significant performance gains achieved in terms of both IPC (Instructions Per Cycle) and overall execution time. These results demonstrate the efficacy and scalability of our approach to enhancing out-of-order processor performance.

Section II briefly describes Pre-Scheduling. The functionality of the Data flow Pre-Scheduler is presented in Section III. The Implementation and experimental set up are presented in Section IV. The Results and are then presented in Section V. Section VI presents the observations and analysis. The project's work is concluded in Section VII.

II. Pre-Scheduling

Pre-scheduling, as a key optimization technique in out-of-order processors with large instruction windows, has garnered significant attention in recent research. Its primary objective is to streamline instruction flow and enhance parallelism by carefully orchestrating instruction execution order. By analyzing dependencies and considering available resources, pre-scheduling mitigates data hazards and resource conflicts, leading to improved performance. Furthermore, it enables instruction-level parallelism and helps achieve higher instruction throughput. This research paper presents a comprehensive study of pre-scheduling. Our results provide valuable insights into the benefits of pre-scheduling, paving the way for further advancements in this area.

III. Functionality

Superscalar architecture aims to improve instruction throughput and performance. IPC determines the effectiveness of instruction execution within a single clock cycle. It displays the typical number of instructions finished in each cycle. A higher IPC suggests that more instructions are performed concurrently or more effectively, improving performance. In superscalar processors, the issue buffer is crucial for out-of-order execution and helps handle instruction dependencies. By enabling instructions to be issued and executed independently and optimizing parallelism, it improves Instruction Per Cycle (IPC). IPC is affected by the issue buffer size. The issue buffer is the hardware structure that materializes the instruction window. Issue buffers hold instructions until they are ready to be launched to execution units. In today's processors, instructions are pushed in the issue buffer in sequential order, therefore instructions depending on a long dependency chain occupy the issue buffer for a long time.

A larger issue buffer can hold more instructions, thereby increasing the probability of independent instructions being found and executed. This may result in increased instruction-level parallelism and enhanced IPC. But because instructions must wait longer to be issued, a very large issue buffer could also contribute to latency. Depending on the architecture and workload conditions, a balance between buffer size and IPC must be found.

Because of the distribution of instruction latencies, the IPC for future processors could decline. We must make the processor instruction window larger to avoid this. Two approaches are possible: physical (increasing the capacity of the issue buffer and physical registers) and logical (improving branch prediction accuracy). To prepare for the issue stage, we introduce the "Pre-Schedule Stage". Instructions are provided to the Issue Buffer from the Pre-Schedule Stage not in sequential order but rather in the expected dataflow sequence. In addition to keeping the Issue Buffer's size constant with Instruction Window Size, this helps prevent utilizing entries with operands that are not accessible.

To facilitate the pre-scheduling phase, our team has developed a data flow pre-scheduler. The principle of prescheduling is depicted on Figure 1. A data flow pre-scheduler is an advanced technique used in out-of-order processors with extensive instruction windows. Its purpose is to optimize instructions scheduling by examining their data dependencies and rearranging them accordingly. The pre-scheduler leverages inherent data flow patterns in a program to maximize instruction-level parallelism and enhance the overall performance of modern processors.

We present an implemented algorithm for instruction scheduling that aims to optimize modern processor performance and enhance parallelism. The algorithm incorporates several key strategies to effectively schedule instructions. Firstly, the algorithm assigns priority to WAR (Write-After-Read) dependencies among all other dependencies. This prioritization ensures that instructions with WAR dependencies are given precedence during scheduling.

To facilitate efficient scheduling, instructions with dependencies are placed in the pre-scheduler buffer, with a separation distance equal to the execution latency. This approach ensures that dependencies are satisfied within the required time frame, minimizing potential bottlenecks and stalls in the instruction pipeline. Moreover, the algorithm addresses WAR dependencies between instructions by scheduling the affected instruction at the location with the minimum weight. This strategy minimizes data hazards and ensures efficient execution by reducing dependency delays. Furthermore, the algorithm enforces a constraint that prevents two instructions of the same type from being placed in the same row of the pre-scheduler buffer. By distributing instructions of the same type across different rows, the algorithm maximizes instruction-level parallelism, enabling multiple instructions to be executed simultaneously.

The implemented instruction scheduling algorithm optimizes modern processor performance by effectively managing data dependencies. Through the prioritization of WAR dependencies, appropriate separation of dependent instructions, intelligent scheduling of affected instructions, and avoidance of same-type instruction clustering, the algorithm enhances parallelism and improves instruction execution efficiency. In today's processors, instructions are pushed in the issue buffer in sequential order, therefore instructions depending on a long dependency chain occupy the issue buffer for a long time.

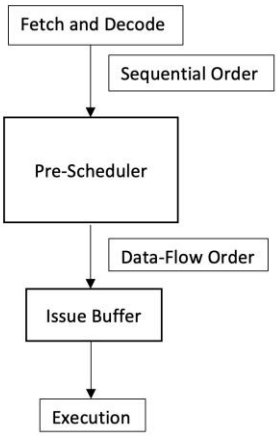

Figure 1: The pre-scheduler sends instructions to the issue buffering the order defined by data dependencies.

IV. Implementation and Experimental Setup

We will discuss the Implementation and Experimental Setup, which looks at a simulated out-of-order superscalar processor based on RV32I integer instructions. This simulated processor serves as the foundation for our study, and its functionality is driven by well-defined steps.

To begin, instructions are fetched sequentially, allowing for the systematic retrieval of the program's instructions. Once fetched, the instructions undergo a decoding process. Critical information, including instruction type, source and destination registers, and execution time, is extracted and made available at subsequent stages. The decoded instructions are then stored in an array, forming groups of instructions ready to be scheduled. This organization of instructions allows for efficient management and analysis of the instruction set, setting the stage for our implementation steps.

To optimize instruction scheduling within the stored instruction array, we have developed a sophisticated instruction scheduling algorithm. This algorithm is designed to operate on the instruction array, strategically processing the instruction sets to maximize instruction-level parallelism and enhance the overall performance of the out-of-order super-scalar processor. By effectively managing dependencies and exploiting parallel execution opportunities, the scheduling algorithm significantly improves processor efficiency and execution throughput.

The experimental setup, driven by the implementation of this advanced scheduling algorithm, provides a robust platform for evaluating and analyzing its impact on our simulated processor's performance. Through rigorous experimentation and measurement, we can gain valuable insights into the effectiveness and efficiency of the algorithm. This will contribute to a deeper understanding of instruction scheduling in out-of-order super-scalar processors.

V. Results

To optimize the execution process, these instructions are fed into a data flow pre-scheduler, which incorporates a proposed algorithm. Figure 2 illustrates a set of instructions to be executed sequentially. The pre-scheduler's primary function is to analyze the dependencies among the instructions and rearrange their order to create a data flow order that eliminates all dependencies. By doing so, the pre-scheduler ensures that instructions can be executed concurrently and efficiently.

Once the instructions are transformed into data flow order, they are transmitted to the issue buffer. The issue buffer acts as an intermediate storage space, where instructions are temporarily held before dispatch to their respective execution units. This buffering mechanism facilitates the smooth flow of instructions through the system, enabling efficient resource utilization and reducing potential bottlenecks.

Figure 2 shows the transformed data flow order and the subsequent execution of instructions. By rearranging the order based on dependency analysis, the proposed algorithm employed by the data flow pre-scheduler maximizes parallelism and minimizes instruction execution time. Ultimately, this approach improves the overall performance and throughput of the system, leading to enhanced efficiency in handling complex computational tasks.

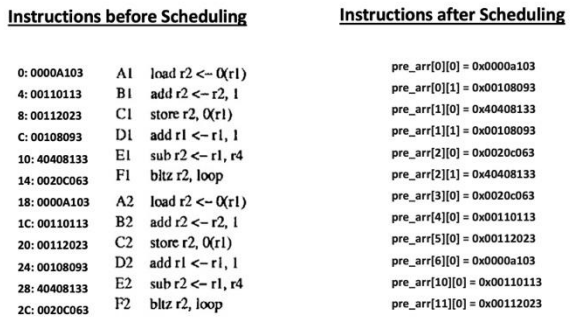

Figure 2: The figure depicts the instructions before Scheduling (Sequential order) and after Scheduling (Data flow order)

In addition to optimizing the execution order of instructions, the proposed approach also provides valuable insights into the dependencies existing among the instructions. These dependencies are identified, as illustrated in Figure 3.

By examining the dependences, the proposed approach offers a comprehensive understanding of the relationships and interconnections between instructions. This information is crucial for ensuring the correct and coherent program execution. Dependencies may arise when an instruction relies on the completion of another instruction or when data produced by one instruction is required as input by another. Overall, the proposed approach enhances the understanding of instruction dependencies, facilitating better decision-making and improving the overall efficiency and performance of the system.

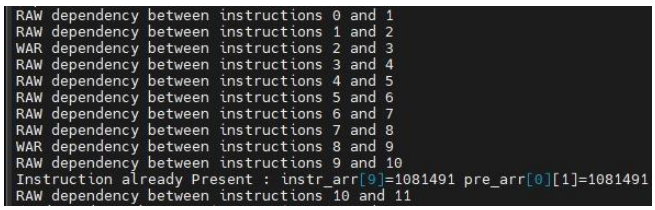

Figure 3: Illustrates the results of the dependences between instructions.

VI. Observations and Analysis

The following observations are based on experimental analysis of our approach. This allowed us to gain valuable insights into the performance and efficiency enhancements achieved through the preschedule buffer. The pre-schedule buffer plays a critical role in enhancing processor performance by allowing instructions to be issued and executed out of their original program order. This capability enables the processor to exploit available parallelism, executing independent instructions concurrently and improving overall performance.

By decoupling instruction scheduling from execution, the prescheduled buffer effectively mitigates instruction latencies and memory access delays. It ensures that instructions can continue to be scheduled and executed, even if previous instructions are awaiting resources or data dependencies to be resolved. This approach helps hide these delays and ensures that the processor remains occupied, effectively utilizing its resources.

One of the key benefits of the pre-schedule buffer is its ability to dynamically reorder instructions based on data dependencies and resource availability. This dynamic reordering optimizes instruction execution order, enhancing instruction throughput and minimizing stalls. By adapting the execution order to the specific requirements of the program at runtime, the pre-schedule buffer enables the processor to achieve better performance.

Moreover, the pre-schedule buffer significantly improves processor resource utilization. By promptly scheduling instructions to available execution units and resources as they become available, it reduces idle time and maximizes resource utilization. This efficient resource allocation further enhances the processor's efficiency and performance. Importantly, the pre-schedule buffer allows out-of-order instruction completion.

In summary, the pre-schedule buffer is a crucial component that enhances processor performance by enabling out-of-order execution, hiding latencies, optimizing instruction throughput, improving resource utilization, and ensuring correct program semantics. Its inclusion in the processor architecture significantly contributes to overall efficiency and performance gains.

VII. Conclusion

In conclusion, our research has provided significant insights into the role of the issue buffer as a critical pipeline stage in out-of-order processors. While it is a crucial component, it is critical to acknowledge that the traversal time of the issue stage increases with the size of the issue buffer. This consideration limits the implementation of large issue buffers due to potential performance drawbacks.

Through our experimental analysis, we have demonstrated the power of dataflow prescheduling as a technique for dynamically reordering instructions. By prioritizing the data-flow order over the sequential order when pushing instructions into the issue buffer, we have demonstrated the ability to achieve the same level of instructions per cycle (IPC) performance using a smaller issue buffer. This finding has practical implications for processor design, as it suggests a more efficient hardware utilization.

It is crucial to recognize that our proposed implementation represents only one point in the vast design space of processor architectures. Prescheduling, alongside other techniques that address the same problem, should be considered a means to tolerate long instruction latencies effectively. The necessity for a large instruction window arises when there is inadequate instruction parallelism to fully utilize the execution units. This is often observed in code sections with frequent data cache misses.

Our simulated data-flow pre-scheduler predicted all dependencies and scheduled instructions in the data-flow order, enhancing overall performance. These results testify to the effectiveness of dataflow prescheduling in improving processor efficiency. Additionally, we firmly believe that the benefits of data-flow prescheduling can extend beyond the specific architecture studied in our research, offering promising solutions to other architectures confronted with similar challenges.

Moreover, our research findings emphasize the importance of considering trade-offs when implementing dataflow prescheduling or similar techniques. While data-flow prescheduling can enhance performance by rearranging instructions and reducing dependencies, it may also introduce additional overhead in terms of complexity and computational resources. Therefore, future studies should aim to strike a balance between the benefits gained from dataflow prescheduling and the associated costs, ensuring that the overall system performance is optimized. By carefully evaluating and refining these techniques, we can continue to advance the field of processor design and drive innovation in high-performance computing.

In conclusion, our research highlights the significant potential of dataflow prescheduling and related techniques for mitigating instruction latencies and enhancing out-of-order processor efficiency. Further exploration and investigation into these methods has the potential to yield valuable insights and advancements in modern processor design and optimization. By leveraging data-flow prescheduling, future processor architectures can continue to evolve and meet increasing computational workload demands.

VIII. REFERENCES

[1] Michaud, P., Seznec, A. Data-Flow Prescheduling for Large Instruction Windows in Out-of-Order Processors. In Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture, Monterrey, Mexico, 2001, pp. 27-36. doi: 10.1109/HPCA.2001.903249.

[2] Butler, M., Patt, Y. An investigation of the performance of various dynamic scheduling techniques. In Proceedings of the 25th International Symposium on Microarchitecture, 1992.

[3] Aragón, J.L., González, J., González, A., Smith, J.E. Dual Path Instruction Processing.

[4] Palacharla, S., Jouppi, N., Smith, J.E. Complexity effective superscalar processors. In Proceedings of the 24th International Symposium on Computer Architecture, 1997.

[5] Agarwal, V., Hrishikesh, M.S., Keckler, S.W., Burger, D. Clock rate versus IPC: the end of the road for conventional microarchitectures. In Proceedings of the 27th Annual International Symposium on Computer Architecture, 2000.

[6] Leibholz, D., & Razdan, R. (1997). The Alpha 21264: a 500 MHz out-of-order execution microprocessor. In Proceedings of IEEE COMPCOM.

[7] Michaud, P., Seznec, A., & Jourdan, S. (1999). Exploring instruction-fetch bandwidth requirement in wide-issue superscalar processors. In Proceedings of the International Conference on Parallel Architectures and Compilation Techniques.

[8] Kessler, R. E. (1999). The Alpha 21264 microprocessor. IEEE Micro, March 1999.

[9] Henry, D. S., Kuszmaul, B. C., Loh, G. H., & Sami, R. (2000). Circuits for wide-window superscalar processors. In Proceedings of the 27th Annual International Symposium on Computer Architecture.

[10] Ranganathan, N., & Franklin, M. (1998). An empirical study of decentralized ILP execution models. In Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems.

[11] Rotenberg, E., Jacobson, Q., Sazeides, Y., & Smith, J. (1997). Trace processors. In Proceedings of the 30th International Symposium on Microarchitecture.

[12] Smith, J. E., & Pleszkun, A. R. (1985). Implementation of precise interrupts in pipelined processors. In Proceedings of the 12th Annual International Symposium on Computer Architecture.

[13] Srinivasan, S. T., & Lebeck, A. R. (1998). Load latency tolerance in dynamically scheduled processors. In Proceedings of the 31st Annual International Symposium on Microarchitecture.