Introduction:

In parallel computer systems, branch instructions can cause delays by interrupting instruction processing. Predicting branch directions can minimize delays, but incorrect predictions may increase them, underlining the importance of accurate prediction strategies. This paper reviews existing branch prediction techniques, then introduces new, more accurate and efficient methods. The study uses instruction trace data from six FORTRAN programs, emphasizing the variance in branching behavior across applications. The results are context-specific, but the core concepts have broader applicability. Earlier works in this field are acknowledged. The paper details two primary branch prediction strategies, laying the foundation for deeper discussions on enhancing prediction accuracy.

Two preliminary prediction:

Branch instructions evaluate a condition to determine execution flow. If true, execution starts at the target address; if false, it continues sequentially. Unconditional branches always have consistent outcomes, either always true or false, but were not differentiated in our statistics. Therefore, unconditional branches were included. A simple branch prediction method assumes branches are always taken or never taken. Given that most unconditional branches are always true and loops end with branches taken to their start, predicting all branches as taken often achieves over a 50% success rate.

Programs like ADVAN (99.4%) and SCI2 (96.2%) show high accuracy with this strategy, indicating most branches in these programs are indeed taken. For SINCOS (80.2%) and TBLINK (61.5%), the accuracy is moderate, suggesting a mix of taken and not-taken branches. GIBSON (65.4%) and SORSTT (57.4%) have lower accuracies, revealing that this strategy isn't always optimal.

In summary, the data from Figure 1 reinforces the concept that branching behavior can vary significantly between different programs. Strategy 1, which assumes all branches will be taken, can be incredibly effective for some programs but less so for others. The variation in prediction accuracy across different programs underscores the importance of having adaptable and diverse branch prediction strategies to cater to the unique behavior of different programs.

Strategy 2, a branch prediction mechanism, operates on the premise of predicting a branch's decision based on its last execution. If previously unexecuted, the branch is predicted to be taken. Analyzing the accuracy data from Figure 2, Strategy 2 yields notable accuracy across programs: ADVAN (98.9%), GIBSON (97.9%), SCI2 (96.0%), SINCOS (76.2%), SORTST (81.7%), and TBLINK (91.7%). Compared to Strategy 1, Strategy 2 typically offers superior prediction accuracy. However, it's not without challenges. Theoretically, its feasibility is contested due to the potential infinity of individual branch instructions a program might contain. Strategy 2 also demonstrates an inherent second-order program sensitivity, as it predicts unexecuted branches as taken. A distinguishing observation is that Strategy 1 occasionally trumps Strategy 2 in scenarios where frequently taken branches aren't. This occurs because

Strategy 2, in such cases, makes two successive incorrect predictions. Despite its high accuracy, these nuances indicate that other strategies might achieve even higher success rates than Strategy 2.

Strategy 1a involves predicting that certain operation codes will always result in a branch being taken, while others will not. This strategy was developed based on analyzing six CYBER 170 FORTRAN programs. It found that operations like "branch if negative", "branch if equal", and "branch if greater than or equal" are commonly taken. Applying this strategy improved prediction accuracy in most programs, with the GIBSON program seeing the biggest increase from 65.4% to 98.5%. However, some inaccuracies were observed, particularly when predicting the "branch if plus" operation.

Strategy 3 predicts that all backward branches will be taken and all forward branches will not be taken, based on the idea that loops typically end with backward branches. While this strategy often performed well, sometimes even outdoing Strategy 2, its accuracy was significantly low at about 35% for the SINCOS program. This highlights that program-specific nuances can heavily influence prediction accuracy. A downside of Strategy 3 is the potential delay in prediction since the target address might need computation or comparison with the program counter, making it slower than other strategies.

Branch prediction strategies use past branch history for decision-making. Strategy 2 relies on the most recent branch outcome, while Strategy 7 considers multiple recent executions, predicting based on majority outcomes. Another method uses only the first execution, though it's slightly less accurate. One common method, used in Strategy 2, utilizes an associative memory storing recent branch instructions and their outcomes. If a branch isn't in this table, a default prediction (usually "taken") is applied. This approach is memory-efficient as only 'not taken' branches need storage. For table replacement, both FIFO and LRU strategies are viable. However, due to the repetitive nature of branch instructions, LRU aligns well with methods where only 'not taken' branches are stored.