Study Notes: Programming on Modern Processors - Single Thread Version

I. Modern Processor Architecture Recap

- **Register Renaming:** Redirects instruction outputs to physical registers and inputs from architectural registers to correct physical registers.
- **Out-of-Order (OoO) Execution:** Executes instructions when all operands are ready, regardless of the original program order.
- **Reorder Buffer (RoB):** Stores results of speculative execution before knowing if the instruction should be executed. Instructions are retired in order.
- **SuperScalar:** Fetches and issues multiple instructions per cycle from the same process/thread.

II. Key Concepts

- **Instruction-Level Parallelism (ILP):** Exploiting parallelism by executing multiple instructions simultaneously. Higher Instructions Per Cycle (IPC) or lower Cycles Per Instruction (CPI) are desirable.
- **Speculative Execution:** Executing instructions before knowing if they are needed.
- **Branch Prediction:** Predicting the outcome of branch instructions to avoid pipeline stalls. Modern processors use perceptron and tournament predictors.
- **Data/Instruction Prefetching:** Fetching data and instructions into the cache before they are needed.
- **Cache Locality:** Writing code to maximize cache hits.

III. Hazards and Bottlenecks

- **Structural Hazards:** Occur when multiple instructions require the same hardware resource at the same time.
- **Data Dependencies:** Operations that depend on the results of previous operations (e.g., pointer chasing).
- **Inter-Iteration Dependencies:** Dependencies between loop iterations (e.g., linked lists, trees). These limit ILP.
- **Critical Path:** The longest chain of dependent operations that determines the minimum execution time. Improving performance requires optimizing the critical path.

IV. Popcount Problem

- **Definition:** Counting the number of set bits (1s) in a binary representation of a value.
- **Applications:**
  - Parity bits in error correction/detection code
  - Cryptography
  - Sparse matrix operations
  - Molecular Fingerprinting
  - Succinct data structures (bit vectors, wavelet trees)

V. Popcount Implementations and Performance

Several implementations of the popcount function were analyzed for performance on modern processors. The key takeaway is that different implementations have vastly different performance characteristics due to how well they exploit instruction-level parallelism and avoid branches.

- **Implementation A:** Naive bit counting with a loop and shifts.
- **Implementation B:** Loop unrolling of Implementation A. Reduces branch overhead.
- **Implementation C:** Uses a lookup table to count bits in 4-bit chunks.
- **Implementation D:** Uses a lookup table and loop unrolling.
- **Implementation E:** Uses a switch statement to count bits in 4-bit chunks. Performs poorly due to branch mispredictions.
- **Hardware Acceleration (SSE4.2):** Uses the `_mm_popcnt_u64` intrinsic for the POPCNT instruction. Provides the best performance.

Key Performance Factors:

- **Dynamic Instruction Count:** The total number of instructions executed.
- **Branch Mis-prediction Rate:** The percentage of incorrect branch predictions.
- **Cycles Per Instruction (CPI):** The average number of cycles required to execute an instruction.
- **Instruction-Level Parallelism (ILP):** The ability to execute multiple instructions simultaneously.

Why some implementations are better:

- **B vs. A:** B has lower dynamic instruction count, fewer branch instructions, and better CPI due to loop unrolling.
- **C vs. B:** C has better CPI because it uses a lookup table, reducing data dependencies and enabling more parallelism.
- **D vs. C:** D eliminates branches through loop unrolling.
- **E vs. Others:** E performs the worst due to high branch misprediction rates caused by the switch statement.

VI. Takeaways for Efficient Code

- **Exploit Instruction-Level Parallelism (ILP):** Maximize IPC and minimize CPI.
- **Avoid Data-Dependent Operations:** Minimize pointer chasing and other operations that create long dependency chains.
- **Avoid Inter-Iteration Dependencies:** Design algorithms that minimize dependencies between loop iterations.
- **Use Loop Unrolling:** Reduce control overhead from branches.
- **Use Lookup Tables:** Replace expensive data-dependent operations with table lookups (when appropriate and cache-friendly).
- **Make Code Predictable:** Enable compilers to perform aggressive optimizations, improve branch prediction accuracy, and reduce cache miss rates.
- **Leverage Hardware Features:** Use specialized instructions (e.g., POPCNT) when available.

VII. Parallel Architectures

- **Simultaneous Multithreading (SMT):** Allows multiple threads to share the same processor core.
- **Chip Multiprocessors (CMP):** Multiple processor cores on a single chip.
- **Parallel Programming:** Writing code to execute on parallel architectures.

VIII. Simultaneous Multithreading (SMT)

- **Concept:** Allows multiple independent threads to execute concurrently on the same processor core, sharing resources like functional units and caches.
- **How it works:**
  - Multiple program counters (PCs)
  - Multiple sets of architectural-to-physical register mappings
  - Shared cache and pipeline functional units
  - Tagged reorder buffer to track instruction ownership.
- **Benefits:**
  - Improved pipeline utilization
  - Increased throughput
- **Drawbacks:**
  - Potential for increased cache miss rates due to sharing
  - Possible reduction in single-thread latency due to resource contention.

IX. Glossary

- **ALU:** Arithmetic Logic Unit. Performs arithmetic and logical operations.
- **CPI:** Cycles Per Instruction. A measure of processor performance. Lower is better.
- **IPC:** Instructions Per Cycle. A measure of processor performance. Higher is better.
- **ILP:** Instruction-Level Parallelism. The degree to which instructions can be executed in parallel.
- **OoO:** Out-of-Order execution.
- **RoB:** Reorder Buffer.
- **SMT:** Simultaneous Multithreading.
- **SuperScalar:** A processor architecture that can execute multiple instructions per clock cycle.