Study Notes: Multithreaded Architectures and Programming

I. Optimizing Code for Modern Processors

- **Minimize Critical Path:** Focus on reducing operations in the most performance-sensitive code sections.
- **Cache/Memory Locality:** Prioritize optimizing data access patterns for better cache utilization. Memory latencies are significant. Consider using arrays or hash tables instead of linked lists for improved locality.
- **Branch Prediction:** Be mindful of branch costs, as deeper pipelines increase the penalty for mispredictions.
- **Instruction-Level Parallelism (ILP):** Exploit multiple functional units by ensuring code can execute instructions in parallel. Hide instructions under the critical path and utilize diverse functional units simultaneously.
- **Long Latency Operations:** Avoid operations with high latency (e.g., modulo).
- **Compiler Optimizations:** Leverage compiler optimizations, but be aware of their limitations.
- **Sorting & Loop Unrolling:** Use these techniques to improve performance.

II. Simultaneous Multithreading (SMT)

- **Concept:** SMT allows multiple threads to share the same processor core, improving resource utilization.
- **Mechanism:** By scheduling another running program instance (thread), the processor avoids wasted issue slots.
- **Benefits:**
    - Better utilization of pipeline resources.
    - Improved execution throughput.
- **Drawbacks:**
    - Potential latency increase for individual threads due to shared functional units and cache.

III. Chip Multiprocessors (CMP) / Multi-Core Processors

- **Moore's Law:** The number of transistors on a chip doubles approximately every 12-24 months, driving CPU performance gains.
- **Transistor Count:** Modern processors pack a large number of transistors, enabling complex microarchitectures.
- **Issue Width:** The number of instructions a processor can issue per cycle.
- **CMP Concept:** Multiple processor cores on a single chip. Each core has its own L1 and L2 caches, while sharing a last-level cache (LLC).

IV. SMT vs. CMP

- **SMT:** A superscalar processor with multiple instruction front-ends.
- **CMP:** Multiple cores, each with its own pipeline and cache.
- **Trade-offs:**
    - **Single Program:** SMT *might* perform better due to resource sharing, but it depends.

    - **Multiple Applications:** CMP *might* have lower cache miss rates, but it depends. Branch misprediction rates also depend.
    - **Parallel Threads:** Cache miss and branch misprediction rates depend on the specific program and data access patterns.
- **Modern Processors:** Often combine both CMP and SMT.

V. Parallel Programming

- **Exploiting Parallelism:** Requires dividing computations into multiple processes or threads.
- **Processes:**
    - Separate programs running concurrently.
    - Own virtual memory space.
    - Require explicit inter-process communication (e.g., MPI, Spark).
- **Threads:**
    - Independent portions of a program running in parallel.
    - Share the same virtual memory space (e.g., pthreads, OpenMP).
- **Shared Memory:** Threads communicate through shared memory.

VI. Coherency & Consistency

- **Coherency:** Guarantees that all processors see the same value for a variable/memory address at the same time.
- **Consistency:** Ensures that all threads see data changes in the same order.
- **Snooping Protocol:** A common cache coherency protocol where each processor broadcasts/listens to cache misses.
- **Cache Line States:**
    - **Invalid:** Data in the block is invalid.
    - **Shared:** Processor can read the data; other processors may also have it.
    - **Exclusive:** Processor has exclusive access to the data and is the only one with the up-to-date copy.

VII. Cache Misses (4Cs)

- **Compulsory:** First access to a block.
- **Conflict:** Multiple blocks map to the same cache set.
- **Capacity:** Cache is too small to hold all required data.
- **Coherency:** A block is invalidated due to sharing among processors.
    - **True Sharing:** Processor A modifies data that Processor B needs.
    - **False Sharing:** Processor A modifies data X, and Processor B modifies data Y, but X and Y are in the same cache line, leading to unnecessary invalidations.

VIII. Parallel Programming Takeaways

- Cache coherency guarantees eventual data consistency, but not *when*.
- Cache coherency can lead to unexpected cache invalidations/misses if not handled carefully.
- Processor behaviors are non-deterministic (scheduling, instruction execution order).

IX. Code Examples and Considerations

- **Volatile Keyword:** Prevents the compiler from optimizing a variable into a register, ensuring it's always read from memory.
- **Threaded Vector Addition (vadd):** Different implementations can have varying cache miss rates due to data access patterns. Version R (block-based) generally performs better than Version L (interleaved) due to reduced false sharing.

Glossary

- **SMT (Simultaneous Multithreading):** A processor design that allows multiple independent threads of execution to run concurrently on a single processor core.
- **CMP (Chip Multiprocessor):** A single integrated circuit containing two or more processors (cores).
- **ILP (Instruction-Level Parallelism):** The degree to which independent instructions can be executed in parallel.
- **Cache Coherency:** A system that ensures that multiple caches in a multi-processor system maintain a consistent view of shared memory.
- **Cache Line:** A small block of memory that is transferred between the cache and main memory.
- **False Sharing:** A performance-degrading situation that occurs when multiple threads access different data items that happen to reside within the same cache line.
- **Critical Path:** The sequence of stages determining the minimum time needed for an operation or other process.
- **Moore's Law:** The observation that the number of transistors in a dense integrated circuit doubles approximately every two years.