Study Notes: Parallel Programming and Dark Silicon Problem

I. Parallel Programming (Continued)

A. Recap: Modern Processors

- **CMP/SMT:** Modern processors utilize Chip Multiprocessing (CMP) and Simultaneous Multithreading (SMT) to enhance performance.
- **Cache Coherency:** Maintaining data consistency across multiple cores with caches.

B. Shared Memory and Cache Coherency

- **Shared Virtual Address Space:** Threads share a common memory space.
- **Cache Coherency Protocol:** Ensures that all cores have a consistent view of the data.
  - Example: Code snippet demonstrating shared memory access and potential cache coherency issues.

C. Cache Misses (4Cs)

- **Compulsory:** First-time access to a block.
- **Conflict:** Multiple blocks mapping to the same cache location.
- **Capacity:** Cache is full, and a block must be evicted.
- **Coherency:** A block is invalidated due to sharing among processors.
  - **True Sharing:** Processor A modifies data X, and Processor B needs to access the updated X.
  - **False Sharing:** Processor A modifies data X, and Processor B modifies data Y, but X and Y reside in the same cache block, leading to unnecessary invalidations.

D. Out-of-Order (OoO) Scheduling

- Processors and compilers may reorder memory operations/instructions.
- Cache coherency protocols guarantee eventual data consistency, but not the timing of updates.
- Processor behavior is non-deterministic.
- Threads may not execute immediately after being spawned.

E. Memory Fences

- **mfence (x86):** Instruction to prevent reordering of memory operations across the fence. All updates before `mfence` must complete before subsequent instructions proceed.
- Ensures a relaxed consistency model.

F. Takeaways

- Cache coherency is challenging to support.
- Processor behaviors are non-deterministic, making precise timing predictions difficult.

II. The Dark Silicon Problem

A. Introduction

- **Dark Silicon:** The phenomenon where, due to power constraints, not all transistors on a chip can be powered on simultaneously.

B. Moore's Law and Dennard Scaling

- **Moore's Law:** The number of transistors in a dense integrated circuit doubles approximately every two years.
- **Dennard Scaling (Broken):** As transistors get smaller, power density remains constant because voltage and current decrease proportionally. This scaling has stopped.

C. Power Consumption and Density

- If transistor density increases (Moore's Law) but power consumption per transistor remains constant:
  - Total power consumption per chip increases.
  - Power density increases.
  - Given a fixed power budget, not all chip areas can be powered on at the same clock rate.
  - Clock rates may need to be lowered to power on all chip areas.

D. Implications of Dark Silicon

- Even with more cores, not all can be active simultaneously.
- Operating all cores at a slower speed becomes necessary.

E. Optimizing CMPs under Dark Silicon

- Maximize efficiency per chip within a given power budget.
- Solutions:
  - Aggressively adjust the frequency of processor cores. Run compute-intensive programs at full speed while reducing the frequency or turning off other cores.

F. Design Options for CMPs

- **Option A:** Few powerful cores with aggressive frequency adjustment.

- **Option B:** Many wimpy, slower cores.

  - **Single-Threaded Applications:** A delivers better performance.
  - **Multi-Threaded Applications:** B delivers better performance.

G. Power vs. Energy

- **Power:** Direct contributor to heat.
- **Energy:** Related to electricity bills and battery life (Power x Execution Time).

H. Heterogeneous CMPs (big.LITTLE)

- **big.LITTLE Architecture:** Combines high-performance cores (P-cores) with energy-efficient cores (E-cores).
  - P-Cores: Higher performance, higher power consumption.

    - E-Cores: Lower performance, lower power consumption.
- Examples: ARM's big.LITTLE, Intel Alder Lake (P-cores/E-cores), Apple M1.

I. GPU Architectures

- Designed for high throughput and parallel processing.
- Originally for displaying images (e.g., HD video).
- Not latency-oriented by design.
- Utilize vector processing and many-core designs.

J. Vector Processing

- Uniform operations across elements.
- Good spatial locality.
- Simplified processing elements.

K. Parallelism in Modern Computers

- Instruction-Level Parallelism (ILP): Pipelining, OoO execution.
- Data-Level Parallelism (DLP): SIMD instructions, vector processing.
- Thread-Level Parallelism (TLP): Multicore/SMT processors.

L. Takeaways

- Dark silicon era requires innovative architectural solutions.
- Heterogeneous CMPs and GPU architectures are key strategies.

III. Glossary

- **CMP (Chip Multiprocessor):** A processor with multiple cores on a single chip.
- **SMT (Simultaneous Multithreading):** A technique that allows a single processor core to execute multiple threads concurrently.
- **Cache Coherency:** Ensuring that all processors in a multi-processor system have a consistent view of shared memory.
- **Moore's Law:** The observation that the number of transistors in a dense integrated circuit doubles approximately every two years.
- **Dennard Scaling:** The observation that as transistors get smaller, their power density stays constant. This is no longer true.
- **Dark Silicon:** The phenomenon where, due to power constraints, not all transistors on a chip can be powered on simultaneously.
- **Heterogeneous Computing:** Using different types of processors (e.g., CPU and GPU) or cores (e.g., big.LITTLE) in the same system.
- **ISA (Instruction Set Architecture):** The interface between hardware and software, defining the instructions that a processor can execute.
- **GPU (Graphics Processing Unit):** A specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display device.
- **SIMD (Single Instruction, Multiple Data):** A type of parallel processing where a single instruction operates on multiple data points simultaneously.

- **Vector Processing:** A type of SIMD processing that operates on vectors of data.
- **mfence:** A memory fence instruction that enforces ordering constraints on memory operations.