EECE7352 Computer Architecture
Homework 5

Part A

In this problem you will review cache coherency and memory consistency protocols.

a. Discuss the tradeoffs between using a snoopy cache coherency protocol and a directory-based cache coherency protocol.
b. Discuss the role of the Exclusive state introduced in the MESI protocol.
c. Discuss the role of the Ownership state introduced in the MOESI protocol.

a. Snoopy protocols utilize a shared bus for communication and rely on broadcast mechanisms to ensure cache coherence. This design is effective in small systems with a limited number of processors, offering simplicity and low latency. However, as the processor count grows, the increased broadcast traffic places a heavy load on the shared bus, limiting the scalability of snoopy protocols. On the other hand, directory-based protocols achieve coherence through a centralized directory that monitors the state of cache lines across processors. While this approach minimizes broadcast traffic and scales better in larger systems, it introduces additional latency due to directory indirection and requires more complex hardware to manage the directory.

Aspect	Snoopy Protocol	Directory-Based Protocol
Scalability	Limited scalability due to broadcast traffic	Scales better due to point-to-point messages
Latency	Low latency for small systems	Higher latency due to directory indirection
Network Traffic	Generates high traffic for large systems	More efficient for large systems
Complexity	Simpler implementation	More complex due to centralized directory
Hardware Cost	Lower cost; relies on shared bus	Higher cost due to directory storage

Table 1: Comparison between Snoopy and Directory-Based Protocols

b. The Exclusive (E) state in the MESI protocol improves efficiency by designating a cache line as present in a single cache and consistent with the main memory. This state removes the need for broadcast invalidations during write operations, as no other caches contain a copy of the line. Transitions between the Exclusive (E) and Modified (M) states are handled locally, reducing latency and avoiding unnecessary coherence traffic. This optimization is especially advantageous in workloads with frequent read-modify-write operations by a single processor.

c. The Ownership (O) state in the MOESI protocol enables a cache with a modified line to share it directly with other caches, avoiding memory access. This reduces latency and boosts overall system performance. The O state is particularly valuable in multi-processor systems where cache-to-cache transfers frequently occur. By facilitating data sharing while preserving coherence, the Ownership state improves protocol efficiency and minimizes traffic to the memory hierarchy.

Part B

In this problem, find the organization for the TLB present on two different microprocessors, one developed by Intel and the other developed by ARM. Make sure to cite your sources. Provide the details of the organization the whole TLB hierarchy, and the format of a single entry in an L1 TLB (either I or D).

In modern processors, the Translation Lookaside Buffer (TLB) plays a crucial role in translating virtual addresses to physical addresses efficiently. Its organization differs significantly between Intel and ARM architectures.

For Intel processors, such as the Intel Core i9-12900K, the L1 TLB is divided into Instruction (I-TLB) and Data (D-TLB). The I-TLB accommodates 64 entries for 4KB pages and 8 entries for 2MB/4MB pages, while the D-TLB supports 64 entries for 4KB pages, 32 entries for 2MB pages, and 4 entries for 1GB pages. Both TLBs are fully associative, enabling rapid lookups. Complementing the L1 TLBs is an L2 TLB, a shared resource with 1024 entries configured as a 4-way set-associative cache. It handles 4KB and larger page sizes, offering extensive coverage and reducing miss rates.

The ARM Cortex-A76 employs a hierarchical TLB structure designed for flexibility and energy efficiency. Its L1 TLB also has separate sections for instructions and data. The I-TLB contains 48 fully associative entries supporting 4KB, 64KB, and 1MB page sizes, while the D-TLB includes 64 fully associative entries for the same page sizes. The unified L2 TLB provides 512 entries, organized as 4-way set-associative, shared between instructions and data to reduce lookup latency and maintain throughput.

In both architectures, TLB entries store the virtual address, physical address, and metadata, including access permissions, cache attributes, and replacement policy. These designs reflect their architectural priorities: Intel targets high performance for large, complex workloads, whereas ARM focuses on power efficiency and compact memory footprints.

Sources:

1. “Documentation – Arm Developer,” *Arm.com*, 2024. https://developer.arm.com/documentation/100798/0401/Memory-Management-Unit/TLB-organization
2. “Intel Core i9-12900K Processor - Product Specifications,” *Intel*. https://www.intel.com/content/www/us/en/products/sku/134599/intel-core-i912900k-processor-30m-cache-up-to-5-20-ghz/specifications.html

Part C

All scripts, codes, and files are attached in the folder 'Part C'.

The experiment was run on the CPU with the following specifications:
Number of cores: 24
Thread (s) per core: 2
Model name: Intel(R) Xeon(R) CPU X5650 @ 2.67GHz

The parallel program of a Taylor series to compute e^x was run for the following input values. The value of x chosen for the experiment is 5. All variables used are double precision.
Number of terms: 100,000, 250,000, 500,000, and 750,000
Number of threads: 1, 2, 4, 8, 16, 32

The following command was used to compile the program.
gcc -o euler compute_euler.c -lrt -lpthread -lm -std=c99

a. The "compute_euler.c" code implementation to compute e^x was executed for various input values using a bash script to automate the process. This script systematically ran the code with different parameters and stored the corresponding output values in a CSV file named "results.csv." Subsequent analysis and visualization of these results was done using a Python script, "analysis.py". The error in the computed values was determined using the following formula.
$Actual\ Value = 148.413159$
$Error = |Computed\ Value - Actual\ Value|$

Additionally, the speedup was calculated to assess performance improvements when parallel processing was employed. The speedup was defined as the ratio of the time taken for a single thread to the time taken for multiple threads, for each specific value of the "number of terms" parameter.

$$Speedup = \frac{Time\ for\ 1\ Thread}{Time\ for\ N\ Threads}$$

Figures 1 and 2 present the results of these analyses. Figure 1 illustrates the relationship between the error and the number of terms, showing how computational accuracy improves with increasing terms. Figure 2 depicts the speedup achieved as the number of threads increases, providing insights into the scalability and performance of the implementation.

Figure 1 illustrates the relationship between the number of terms used in a computation and the resulting error, for different thread configurations. As the number of terms increases, the error in computation generally decreases for all thread configurations. However, the rate of decrease varies. With a higher number of threads, the error decreases more rapidly initially, but then plateaus at a higher level. This pattern indicates that while increasing the number of threads enhances accuracy initially, the improvements diminish beyond a certain point. Additionally, the graph shows that using more threads generally leads to lower error, especially

for larger numbers of terms. However, the benefit of additional threads becomes less significant as the number of terms increases.

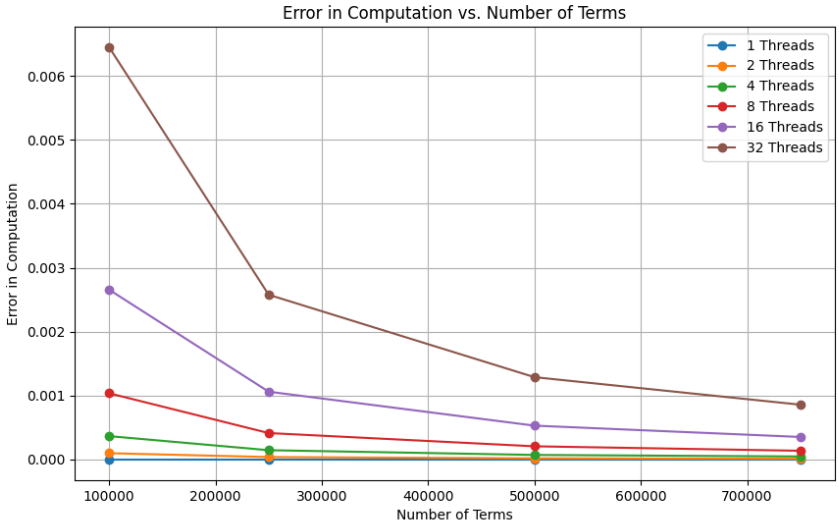

Figure 1: Error in Computation vs. Number of Terms

Figure 2 illustrates the relationship between the number of threads used in a computation and the resulting speedup, for different numbers of terms. As the number of threads increases, the speedup generally improves for all numbers of terms. However, the rate of improvement varies. For smaller numbers of terms, the speedup increases rapidly initially, but then plateaus at a lower level. This suggests that increasing the number of threads can significantly improve performance up to a certain point, beyond which diminishing returns are observed. For larger numbers of terms, the speedup increases more gradually, indicating that the benefit of additional threads is less pronounced.

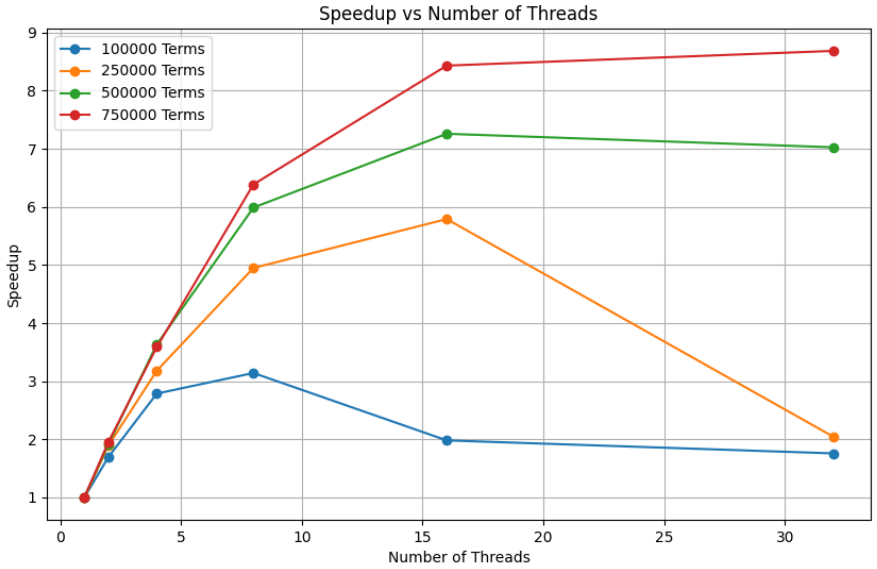

Figure 2: Speedup vs. Number of Threads

b. The above steps were repeated for a program using single precision floating-point variables. Figures 3 and 4 illustrate the resulting plots. A visual comparison of Figures 3 and 4 with Figures 1 and 2 reveals that the changes in results due to the change in precision are negligible. However, a notable exception is the speedup plot for 100,000 terms (Figure 4 – blue line), which exhibits significant differences compared to Figure 2. This indicates that in many practical applications where extreme precision is not essential, single precision provides a suitable balance between accuracy and computational efficiency.

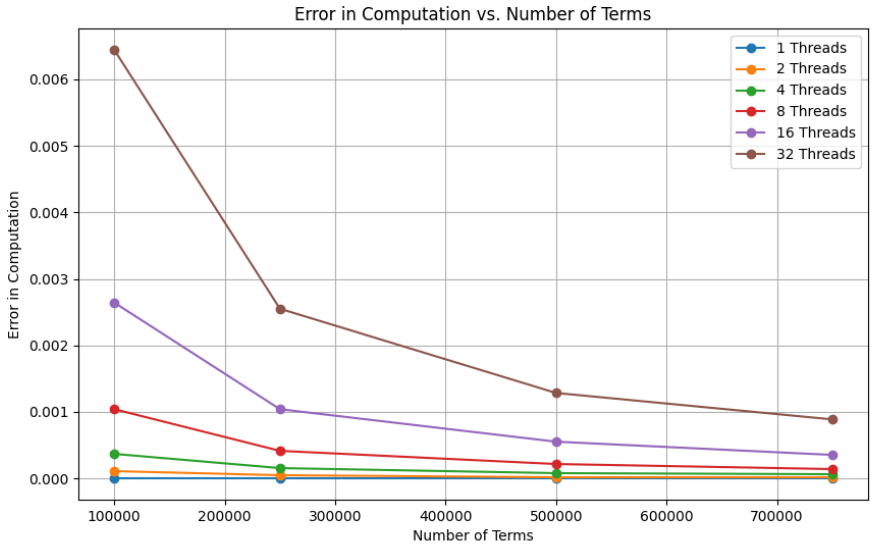

Figure 3: Error vs. Number of Terms – Single Precision

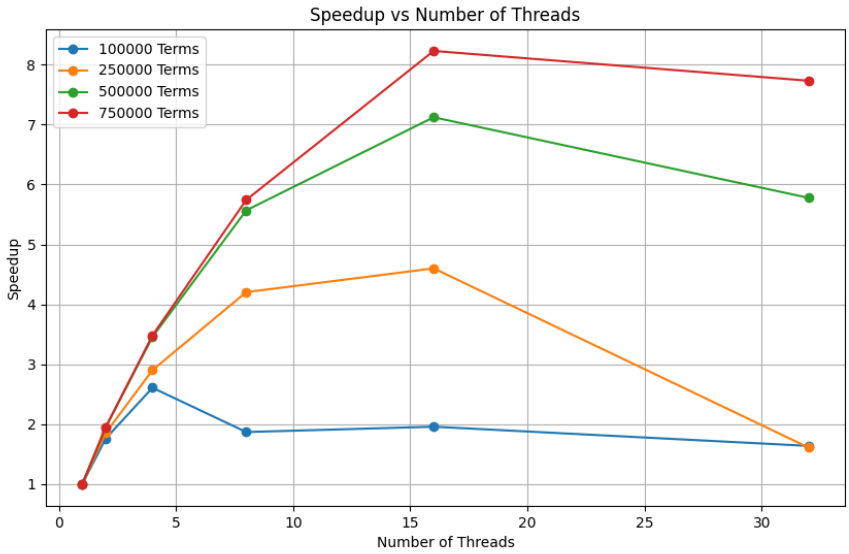

Figure 4: Speedup vs. Number of Threads – Single Precision

The following plots illustrate the impact of using double and single precision on both execution time and result accuracy when performing calculations with increasing numbers of threads. Figure 5 demonstrates that double precision generally requires more execution time than single precision. This is because double precision numbers use twice the memory and computational resources as single precision numbers. However, as the number of threads increases, the execution time for both precision types decreases due to the ability to distribute the workload across multiple processors. Figure 6 highlights the difference in results between double and single precision calculations. Notably, the results obtained remain relatively similar. In conclusion, using single precision can significantly improve execution time compared to double precision, especially for large-scale computations. However, it is important to consider the potential loss of accuracy in the results, particularly when dealing with many floating-point operations or a large number of threads. The choice between double and single precision depends on the specific requirements of the computation.

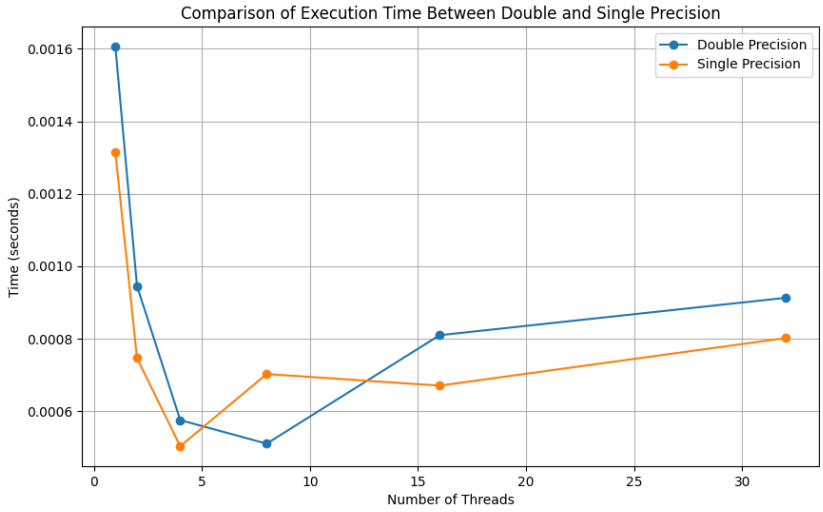

Figure 5: Comparison of Execution Time Between Double and Single Precision

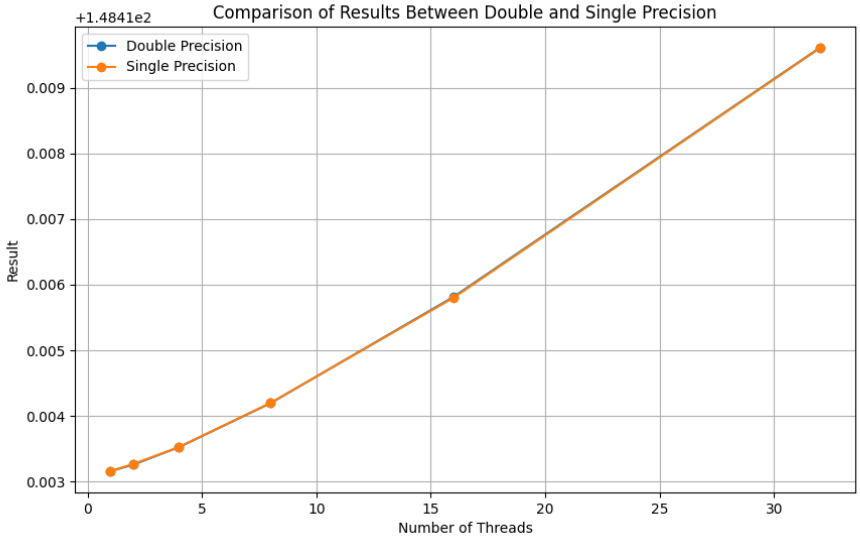

Figure 6: Comparison of Results Between Double and Single Precision

Part D

Using a large language model of your choice, provide a list of five questions you might give as a homework assignment on virtual machines, and then answer at least three of those questions. Please provide details on the LLM you used, and any other details that are relevant. Make sure to cite any source you used for answering the questions.

LLM Model Used: OpenAI ChatGPT

Questions:

1. Explain the role of a hypervisor and differentiate between Type 1 and Type 2 hypervisors.
2. Describe the role of the shadow page table in virtual machine memory management.
3. Discuss the significance of hardware-assisted virtualization (e.g., Intel VT-x or AMD-V) in improving VM performance.
4. How does live migration of virtual machines work, and what are its challenges?
5. Compare and contrast para-virtualization with full virtualization.

Answers:

1. A hypervisor is a software or firmware layer that facilitates the creation, management, and operation of virtual machines (VMs). By abstracting the physical hardware, it enables multiple VMs to share resources such as CPU, memory, and storage while maintaining isolation for security and stability. The hypervisor also optimizes resource allocation dynamically, enhancing efficiency in environments with multiple tenants.
   Hypervisors are categorized into two types. Type 1 hypervisors, or bare-metal hypervisors, operate directly on the physical hardware without an underlying operating system. This direct interaction with hardware makes them highly efficient and well-suited for enterprise applications like data centers and cloud computing. Notable examples include VMware ESXi, Microsoft Hyper-V, and Xen. Conversely, Type 2 hypervisors, or hosted hypervisors, run on top of an existing operating system. While easier to use and ideal for personal or development tasks, they incur higher overhead due to reliance on the host OS. Examples of Type 2 hypervisors include VMware Workstation and Oracle VirtualBox.

2. In virtualized systems, shadow page tables are essential for managing memory. Virtual machines (VMs) behave as if they have exclusive control over memory, translating virtual addresses to physical addresses within their allocated space. However, because multiple VMs share the same physical hardware, the hypervisor must map guest physical addresses to host physical addresses. Shadow page tables handle this by maintaining mappings that directly link guest virtual memory to host physical memory.
   This process ensures VMs remain isolated while allowing the hypervisor to oversee actual memory access. However, shadow page tables introduce performance overhead since the hypervisor must frequently monitor and update these mappings. Modern systems have largely addressed this issue through hardware-assisted technologies like Extended Page Tables (EPT) in Intel VT-x and Nested Page Tables (NPT) in AMD-V, which transfer memory translation tasks to hardware, significantly improving efficiency.

3. Hardware-assisted virtualization, enabled by technologies like Intel VT-x and AMD-V, has significantly enhanced the performance of virtual machines by embedding virtualization capabilities directly into the CPU. Traditionally, hypervisors depended on software-based methods like trap-and-emulate, which added substantial latency and complexity. By allowing the CPU to handle key virtualization tasks natively, hardware-assisted virtualization minimizes these inefficiencies.

   A key feature is the introduction of Extended Page Tables (EPT) or Nested Page Tables (NPT), which optimize memory management by performing virtual address translations at the hardware level. This eliminates the need for shadow page tables, greatly reducing context-switching overhead and accelerating memory access. Another notable capability is "ring deprivileging," which allows guest operating systems to execute privileged instructions safely and efficiently without requiring modification. This facilitates practical full virtualization and enhances security by isolating VMs.

   These innovations have made virtualization more scalable, efficient, and secure, supporting applications like cloud computing, containerization, and high-performance virtual environments. As a result, hardware-assisted virtualization has become a cornerstone of modern data centers and enterprise infrastructure.

Sources:

1. vmware, “What is a Hypervisor? | VMware Glossary,” *www.vmware.com*. https://www.vmware.com/topics/hypervisor.
2. “What is a Bare Metal Hypervisor? | VMware Glossary,” *Vmware.com*, 2024. https://www.vmware.com/topics/bare-metal-hypervisor.
3. VMTN, “Hypervisor? That’s no hypervisor!,” *VMTN Blog*, Mar. 24, 2007. https://blogs.vmware.com/vmtn/2007/03/hypervisor_that.html.
4. “What Are the Basics of Virtualization,” *Vmware.com*, 2024. https://docs.vmware.com/en/VMware-vSphere/8.0/vsphere-vcenter-esxi-management/GUID-ED375B12-7D08-4B7E-81EE-DCE83E51B1AF.html.

Extra Credit

Graphics Processing Units are highly parallel single-instruction multi-threaded architectures that achieve impressive speedups for a range of applications. Using the NVIDIA A100 architecture, describe the memory hierarchy present on this device, and compare how it differs from the memory hierarchy of the AMD Ryzen CPU.

The NVIDIA A100 GPU and AMD Ryzen CPU feature distinct memory hierarchies, each tailored to their specific workloads and performance goals. The NVIDIA A100 is engineered for massively parallel, throughput-intensive tasks, such as deep learning, data analytics, and scientific computing. At the core of its memory hierarchy is a highly parallel structure that ensures efficient data handling across thousands of threads. Each Streaming Multiprocessor (SM) is equipped with 164 KB of shared memory, facilitating fast intra-thread communication and reducing reliance on slower global memory. A large 40 MB L2 cache is shared among all SMs, enabling localized data reuse and minimizing global memory latency. For large-scale data operations, the A100 relies on 40 GB of HBM2e (High-Bandwidth Memory 2E), which provides exceptional bandwidth of up to 1.6 TB/s, crucial for high-performance workloads requiring rapid data transfer.

In contrast, the AMD Ryzen CPU's memory hierarchy is designed for low-latency, general-purpose tasks, such as application processing and multitasking, common in desktop and server environments. Each core features a dedicated 64 KB L1 cache, split evenly between instructions and data, allowing for immediate access to frequently used information. The cores also have private 512 KB L2 caches, offering slightly larger and faster storage for ongoing tasks. The L3 cache, up to 64 MB in size, is shared among cores within a Core Complex (CCX), enhancing performance in multi-threaded applications by reducing contention for main memory. For main memory, Ryzen CPUs utilize DDR4 or DDR5 memory, which provides lower latency compared to HBM2e but with reduced bandwidth, making it suitable for workloads where quick, frequent access to smaller data sets is critical.

The primary difference between the A100 and Ryzen lies in their design goals. The A100 focuses on maximizing parallelism and memory bandwidth, excelling in compute-heavy applications such as AI training and simulations. Its memory architecture is optimized for handling large-scale data operations across thousands of threads simultaneously. In contrast, the Ryzen CPU prioritizes low-latency access, making it ideal for a broad range of tasks that demand quick responses and efficient handling of smaller, diverse data sets. These distinct approaches reflect how memory hierarchies are adapted to the specific demands of their respective workloads, whether for specialized high-performance computing or versatile general-purpose use.