COMPUTER SCIENCES DEPARTMENT
UNIVERSITY OF WISCONSIN—MADISON
PH.D. QUALIFYING EXAMINATION

Computer Architecture
Qualifying Examination

Spring 2023

GENERAL INSTRUCTIONS:

1. Answer each question in a separate book.

2. Indicate on the cover of *each* book the area of the exam, your code number, and the question answered in that book. On *one* of your books list the numbers of *all* the questions answered. *Do not write your name on any answer book.*

3. Return all answer books in the folder provided. Additional answer books are available if needed.

SPECIFIC INSTRUCTIONS:

Answer all of the following **SIX** questions. The questions are quite specific. If, however, some confusion should arise, be sure to state all your assumptions explicitly.

POLICY ON MISPRINTS AND AMBIGUITIES:

The Exam Committee tries to proofread the exam as carefully as possible. Nevertheless, the exam sometimes contains misprints and ambiguities. If you are convinced a problem has been stated incorrectly, mention this to the proctor. If necessary, the proctor can contact a representative of the area to resolve problems during the *first hour* of the exam. In any case, you should indicate your interpretation of the problem in your written answer. Your interpretation should be such that the problem is non-trivial.

1. Prefetching

Prefetching is used to improve the performance of caches while introducing additional power consumption. In this problem, we will focus on performance improvements alone and based on hardware prefetching . For parts a) and b) assume the machine is an OOO processor with a 2-level cache hierarchy, and prefetching only happens into the L1 cache.

(a) A strided prefetching algorithm would prefetch another cache line **B + δ** where **B** represents the current cache line being accessed. Describe some programming / application scenarios where such a prefetcher is effective.
(b) Discuss some of the limitations of such a strided prefetcher.
(c) Now assume prefetching can be done at any level of the cache hierarchy. In recent years, datacenters have become increasingly popular and widely used, using CPUs with large core counts. Independent programs ("jobs") concurrently run on a single server CPU. In such settings, the datacenter operators prefer to disable the CPU's prefetch **into the LLC**, while typically allowing L1 prefetch. Discuss why this decision might have been made.

2. Virtual Caches

Most L1 caches today are physical: indexed and tagged with physical addresses.

(a) How do most L1 caches today avoid the latency penalty of logically performing address translation before an L1 cache lookup? Under what constraints?
(b) What are the pros and cons with switching to a virtual L1 cache? Be sure to consider single-core and multicore issues.
(c) What are the pros and cons with switching from physical to virtual for (i) a private L2, (ii) an L2 shared among a few cores, or (iii) a last-level cache (LLC) among all cores on a chip?

3. Performance Measurements

Multi-threaded applications are frequently used in modern multi-core CPUs to improve performance by taking advantage of the available parallelism. They can also suffer from difficult-to-explain performance cliffs and CPI pitfalls.

(a) One common trick to trying to gauge performance of a multi-threaded application is to run it multiple times on the same system with the same configuration. However, performance often varies from run to run. Describe at least three reasons why the performance could vary between runs.
(b) If you are responsible for evaluating the potential benefits of a multi-threaded program on a multi-core CPU, how might you design your methodology to effectively determine the utility of this program on this hardware?

4. Interconnection Networks

Network-on-Chip (NoC) connects the cores, cache banks, and other IP blocks. Using the right topology and architecture for NoC is crucial for reducing latency and maximizing bandwidth. Meshes are somewhat simple to design and implement, while more complex NoC provide better performance and other characteristics.

(a) Discuss the pros and cons of using mesh topology for NoCs.
(b) How does the flattened butterfly network improve performance over mesh topologies?
(c) Explain how the proposed flattened butterfly networks can be implemented in a scalable manner compared to hypercubes, which offer similar connectivity benefits.

5. Reliability

In the 1990s, microprocessors went from having no memory error detection or correction to using parity to detect single-bit memory errors to using SECDED ECC codes to detect double and correct single-bit memory errors. Similarly, over the past decade, microprocessors have gone through the same progression for L2 caches. Some systems are even using parity bits, ECC codes for L1 caches and registers.

(a) How does the choice of code and/or code word size interact with write-through vs. writeback policies and inclusive vs. exclusive caches?
(b) **As SECDED can only correct one error and detect two errors in a cache line, modern processors prone to soft errors have started using data scrubbing**. Data scrubbing involves running a background task that periodically inspects the entire cache memory for errors. The insight being that before a second error could hit a cache-line, scrubbing would read, correct and rewrite contents (with a newly computed error code and clean data). Thus this "new" data would contain zero errors, making the cache line ready to accommodate another correctable soft error later in time. Describe the tradeoffs between doing scrubbing vs a more complex code that can correct two errors.

6. Pipeline and Microarchitecture

"Classical" and somewhat older OOO processors had relatively small issue widths, while more modern processors like the Zen3, Apple M2, etc. have much larger issue widths.

(a) **Outside** of better branch prediction, better prefetching, larger/better caches, what techniques are needed to accomplish a 6 or 8-wide OOO processor to execute "general purpose" programs at high performance?
(b) One common feature in modern processors is wider SIMD/short-vector instructions. Compared to general purpose programs these require somewhat different techniques. What are the hardware challenges in supporting wide SIMD/short-vector instructions like 256-bit and 512-bit AVX?