CS 533	**Name:** ______________________
Spring 2024	
Homework 3	**NetID:** ______________________
Assigned: 3/19/2024	
Due: 4/4/2024	**UIN:** ______________________

This homework contains 5 pages (including this cover page) and 4 questions.

Please clearly print or typeset your answers.

Note: up to 9 points (out of 90) may be deducted for illegible submissions

Problem Breakdown

Question	Points	Score
1	22	
2	30	
3	20	
4	18	
Total:	90	

1. (22 points) Thread Level Speculation
   (a) (2 points) Define the concept of *Thread Level Speculation (TLS)*.
   (b) (4 points) Compared to a sequential version of a program, explain the effects of the speculatively parallelized version on:
      i. Dynamic instruction count (with and without squashes)
      ii. Performance of the memory system
      iii. Performance of various micro-architectural predictors (such as branch predictors or prefetchers)
   (c) (2 points) What is the effect of task granularity on TLS performance? Explain different ways in which a larger task can positively/negatively affect the performance.
   (d) (2 points) Assume that you are given a sequential program and are asked to parallelize it. You can choose between using traditional multi-threading or using TLS. What factors should you take into account to make that decision? Explain.
   (e) (6 points) Assume the baseline architecture proposed in paper 7a[1], without the performance optimizations given in Section 2.7. Can TLS programs have squashes even if there are no data dependency violations in relation to:
      i. Branch mispredictions? Explain.
      ii. Cache line size? If so, what can a compiler do to avoid/reduce this?
      iii. Cache associativity? If so, what can a compiler do to avoid/reduce this?
   (f) (2 points) Refer to paper 7a . The evaluation shows that the star performers for TLS are **buk** and **equake** both of which are memory bound, as evidenced by the small fraction of busy processor cycles, and the large fraction of memory stall cycles. Explain how **buk** and **equake** were able to achieve such impressive speedups when there was barely any computation to parallelize in the first place.
   (g) (2 points) What are the advantages of forwarding speculative data? What are some of the drawbacks of forwarding speculative data?
   (h) (2 points) What are the main overheads associated with TLS? How does paper 7a try to limit these overheads?

[1] J. Gregory Steffan et al. "A Scalable Approach to Thread-Level Speculation," ISCA, 2000

2. (30 points) Speculative Synchronization
   (a) (4 points) *Transactional Memory (TM)* is a concept closely related to TLS and speculative synchronization. Briefly explain what it is. A good reference about existing TM technologies is "Tim Harris, Jim Larus and Ravi Rajwar. 2010. Transactional Memory (Synthesis Lectures on Computer Architecture), 2 nd Ed. Morgan & Claypool Publishers." [2]
   (b) (2 points) Define the notions of *weak* and *strong atomicity* in TM systems.
   (c) (15 points) Recently, hardware support for TM has been added to selected Intel and IBM machines. The following questions relate to the IBM Power 8 TM implementation[3].
      i. Describe the 3 implicit memory barriers associated with transactions in this system.
      ii. Explain how they augment the *pthreads* library to support Lock Ellision.
      iii. What is the purpose of the Suspended state in their system? How does their TLS scheme leverage the suspend region to ensure correct execution?
      iv. What is the limiting factor for transaction size in this system?
      v. What types of programs will benefit from the Rollback-only transactions?
      vi. Is it possible for two threads to be executing the same critical section concurrently with one thread in the Transactional state and the other in the Non-Transactional state? Why?
   (d) (6 points) The divergent design goals of speculative synchronization and TM have resulted in different trade-offs in terms of complexity and usability.
      i. (2 points) One area of complexity for hardware transactional memory is *transaction virtualization*. Briefly explain what this problem is and how it applies to speculative synchronization.
      ii. (2 points) Another area of complexity for transactional memory is I/O, often referred to as the *output commit* problem. Briefly explain what this problem is and how it applies to speculative synchronization.
      iii. (2 points) Transactional memory solves the deadlock problem and the related lock composability problem. Briefly explain what this problem is and how it applies to speculative synchronization. Can programs deadlock with speculative synchronization? Explain.
   (e) (3 points) Lock based concurrent programs rely on Lamport's happens-before relation for correctness, using memory fences to enforce this relation depending on the memory consistency model. Does speculative synchronization guarantee Lamport's happens before relationship? Explain. (A good explanation of happens-before relation as it relates to data races can be found in the related work section of the Eraser [4] paper.)

[2] Available at https://link.springer.com/book/10.1007/978-3-031-01719-3

[3] H.Q. Le et al. "Transactional Memory Support in the IBM POWER8 Processor," IBM Journal of Research and Development, 2015

[4] S. Savage et al. "Eraser: A Dynamic Race Detector for Multithreaded Programs", ACM Transactions on Computer Systems, 15(4), November 1997.

3. (20 points) Processor in Memory (PIM)

   (a) (2 points) What type of applications benefit most from PIM? What type of applications do you expect to perform better on a traditional (i.e. non-PIM) system?

   (b) (2 points) In a system with prefetching support running a memory-bound application with regular memory accesses, is the idea of PIM still relevant? If so, why?

   (c) (10 points) A recent research topic has been 3D-stacked memories. This seminal paper [5] provides a nice introduction to 3D stacking.

      i. Why does 3D stacking offer both bandwidth improvements and reduction in power when compared to traditional architectures?

      ii. Are there other reasons besides power and performance that it may be necessary to incorporate 3D stacking into future designs (*think of form factors*)?

      iii. In Section 3 of the paper referred to above, what are the 3 configurations used and why do they have different performance and power characteristics? Why do not all benchmarks have performance improvements?

      iv. What are the potential drawbacks of 3D stacking?

      v. Compare and contrast 3D stacking against PIM. What are the advantages and disadvantages of each?

   (d) (6 points) Consider a simple in-order processor with a L1 cache composed of a direct-mapped 8KB cache of 64-byte cache lines and no L2 cache. The latency to the L1 cache is 1 cycle. The additional latency to main memory is 100 cycles. The clock speed is 1 GHz. In addition to the in-order processor, the main memory has a processor-in-memory. The processor in memory has a 1KB direct- mapped cache of 32-byte cache lines. The clock speed of the processor in memory is 250 MHz. The processor in memory can access its cache in 1 cycle and memory in 5 cycles. Both processors can execute only one instruction per cycle. Explain whether each of the following loops is faster on the main processor or the PIM. Compare the steady-state access times for two processor configurations:

```
(1) for (i = 0; i < ∞ ; i++)
        A[i] = B[i]
```

```
(2) for (i = 0; i < ∞ ; i++)
        A[i] = B[i] * B[i + 1]
```

   ASSUMPTIONS: The data caches are empty when the loops start. The arrays A and B are of int type (4 bytes). Think about steady-state conditions, not the boundary conditions. When determining the execution time you can ignore the execution time for all but the memory access (for instance, don't count the multiply; only cache and memory reads and writes).

[5] Bryan Black et al. "Die Stacking (3D) Microarchitecture," MICRO, 2006

4. (18 points) Reliability
   (a) (8 points) In paper 9a [6], the authors use speculative threads for two optimizations.
      i. How does *Efficient fine-grain code monitoring* assist the programmer in minimizing application bugs?
      ii. How do the authors allow multiple monitors to execute in parallel? Make sure to give enough details.
      iii. Why is it advantageous to have *Fine-grain transactional programming* capabilities while executing a program?
      iv. What example applications do they elucidate to prove the merits of their optimizations?
   (b) (10 points) In paper 9c [7], the authors describe a system which is resilient against transient faults
      i. Describe the ways the authors' propose to protect against permanent faults in the SRT scheme.
      ii. Describe the two structures which cause deadlock and for each structure give an example of how they contribute to deadlock.
      iii. How does the CRT processor provide performance improvements over running in lockstep? Are there any drawbacks in comparison to SRT (*think permanent faults*)?
      iv. Why is *slack* fetch not necessary on their SRT system? Futhermore, how can it hurt performance?

[6] Jeffrey Oplinger et al. "Enhancing Software Reliability with Speculative Threads," ASPLOS, 2002

[7] Shubhendu S. Mukherjee et al. "Detailed Design and Evaluation of Redundant Multithreading Alternatives," ISCA, 2002