Jeffrey Lau, Giovanni Lupo, Zacharia Hammad
11/25/2024
ECEC 412
Prof. Anup Das
Project 5

# ECE 412 Project 5

## FCFS - 510.parset.trace.processed

End Execution Time: 96151062
Num Bank Conflicts: 131072

## FCFS - 505.mcf.trace.processed

End Execution Time: 16542980
Num Bank Conflicts: 98120

## FCFS - 507.cactusBSSN_r.trace.processed

End Execution Time: 217824869
Num Bank Conflicts: 32645

## OOO - 510.parset.trace.processed

End Execution Time: 27120908
Num Bank Conflicts: 3991468

## OOO - 505.mcf.trace.processed

End Execution Time: 7282848
Num Bank Conflicts: 1018483

## OOO - 507.cactusBSSN_r.trace.processed

End Execution Time: 70789994
Num Bank Conflicts: 4680763

**Execution Time Reduction:**

- The Out-of-Order (OoO) scheduler significantly reduces total execution time compared to the First-Come, First-Served (FCFS) scheduler across all trace files. The execution time reductions are as follows:
- **510.parset.trace.processed:** Execution time reduced by approximately **71.8%**.
- **505.mcf.trace.processed:** Execution time reduced by approximately **55.9%**.
- **507.cactusBSSN_r.trace.processed:** Execution time reduced by approximately **67.5%**.

**Bank Conflicts:**

- Interestingly, the number of bank conflicts increased with the OoO scheduler compared to FCFS. This increase occurs because the OoO scheduler attempts to schedule more requests whenever possible, leading to more instances where requests cannot be processed due to busy banks. However, the rise in bank conflicts did not adversely affect the overall execution time, thanks to better utilization of the available banks.

**Reasons for Performance Improvement:**

- Parallelism: The OoO scheduler exploits parallelism by scheduling requests that target free banks, which reduces idle time.
- Resource Utilization: Banks are kept busier more consistently, improving overall throughput.
- Reduced Waiting Time: Requests do not have to wait unnecessarily behind others targeting busy banks.

**Consideration of Data Hazards:**

- Our current implementation does not account for data hazards. In real-world scenarios, ignoring data dependencies could lead to incorrect program behavior. Nonetheless, the results demonstrate the potential performance advantages of OoO scheduling, assuming there are no data hazards present.

3) The existing memory controller is equipped with an FCFS scheduler, which serves the request in the same order of insertion. Suppose there are three requests, R1 (target to Bank 0), R2 (target to Bank 0), and R3 (target to Bank 1), in the transaction queue. What is the total serving time with the FCFS scheduler (R1 → R2 → R3)?

| Time Unit | Event | Bank 0 Status | Bank 1 Status |
|---|---|---|---|
| 1 | Issue R1 to Bank 0 | Busy | Free |
| 2 | R1 completes; Issue R2 to Bank 0 | Busy | Free |
| 3 | R2 completes; Issue R3 to Bank 1 | Free | Busy |
| 4 | R3 completes | Free | Free |

**Total Serving Time: 4 time units**

**Explanation:**

- R2 cannot be issued until R1 is completed because both target Bank 0.
- R3 waits for R2, even though Bank 1 is free during R2's execution

4) What if an out-of-order memory controller can prioritize requests that target a free bank? Consider the same example; when R1 is issued, Bank 0 stays busy till R1's completion. In the next clock cycle, the memory controller cannot schedule R2 because Bank 0 is busy; however, all other banks stay free, which means R3 can be scheduled out-of-order. What is the total serving time with the OoO scheduler (R1 → R3 → R2)?

| Time Unit | Event | Bank 0 Status | Bank 1 Status |
|---|---|---|---|
| 1 | Issue R1 to Bank 0; Issue R3 to Bank 1 | Busy | Busy |
| 2 | R1 and R3 complete; Issue R2 to Bank 0 | Busy | Free |
| 3 | R2 completes | Free | Free |

**Total Serving Time: 3 time units**

**Explanation:**

- R3 is scheduled out-of-order when R2 cannot be issued due to Bank 0 being busy.
- Both banks are utilized simultaneously in the first-time unit.
- Total serving time is reduced by **1 time unit** compared to FCFS.

5) With an OoO memory controller, do you still need to consider data hazards?

Yes, data hazards must be considered even with an OoO memory controller. Data hazards occur when there are dependencies between memory operations that could lead to incorrect program execution if not appropriately handled. For example:

- **Read-After-Write (RAW) Hazard:** A read request following a write to the same address must ensure the write completes before the read.

- **Write-After-Read (WAR) Hazard:** A write following a read to the same address must not overwrite data before the read completes.
- **Write-After-Write (WAW) Hazard:** Two writes to the same address must be executed to maintain data consistency.

This project demonstrated the impact of memory scheduling policies on system performance. By implementing both FCFS and OoO schedulers, we observed significant execution time reductions with the OoO scheduler across various workloads.

**Key Takeaways:**

- **Out-of-Order Scheduling:** Allows for better utilization of memory banks, reducing execution time.
- **Bank Conflicts:** While the number of bank conflicts increased with the OoO scheduler, efficient scheduling improved overall performance.
- **Data Hazards:** This must be considered in OoO scheduling to ensure correct program execution.