Handling DRAM Contention Between Multiple Threads

18-740 Poster Session

David Jia, Dennis Liang
Motivation

- Chip multiprocessor and parallel computing are becoming prevalent
- Current DRAM memory controller implementation
  - Each bank in DRAM contains a row buffer for memory accesses
  - Bank request conflicts incur row buffer thrashing and serialized requests, resulting in increased latency
  - DRAM memory accesses optimized for throughput, using First-ready First-come-first-serve (FR-FCFS) scheduling policy
  - Accesses to addresses currently in buffers are prioritized
- Unfairness of parallel memory requests
  - Threads with higher row locality will be prioritized
  - Threads with high frequency memory accesses and low memory locality have high stall-time penalizations
  - Can lead to significant thread starvation
- Ramifications of unfair memory access scheduling
  - No quality of service (QoS) guarantees for parallel systems
  - System prone to denial of service (DoS) attacks that are not preventable through operating system or compiler, since they have no control over thread memory access scheduling
Contributions

- Used simulator to implement existing scheduling policies
- Successfully integrated the Priority Based Fair Scheduler (PBFS)
- Unable to complete the Dynamic Miss Handling Architecture (DMHA) scheduler
- Simulated various scheduling policies using several benchmarks with varying memory access patterns
- Analyzed results of simulation between PBFS and three other scheduling policies
- Evaluated performance benefits and disadvantages of different policies, which can only be accurately observed when comparisons occur on a common simulation platform
Benchmarks

- **libquantum**
  - In each bank, consecutive memory accesses are all within the same row, utilizing spatial locality
  - Hit to miss ratio is 63.0 to 1 when running by itself
  - When run in parallel with 3 other benchmarks using FR-FCFS, the number of row buffer hits is only reduced by 11.2%

- **omnetpp**
  - Extremely high frequency memory accesses
  - Memory addresses change drastically, resulting in a large number of row buffer misses
  - Even when running by itself, hit to miss ratio is 0.573
  - Number of hits reduced by 61.1% when running with other threads using FR-FCFS

- **xalancbmk**
  - Periodically frequent but sparsely located memory accesses
  - Number of hits reduced by 79.7% when run with other threads using FR-FCFS
- **gobmk**
  - Very infrequent memory accesses
  - Would be low priority for FR-FCFS policy
  - Row locality highly differs across different banks

- **sphinx3**
  - Each bank contains frequent memory accesses to different rows, where memory addresses alternate between different rows
  - Bad spatial locality
  - Row buffer misses increased by 136.0% when run in parallel with 3 other threads using FR-FCFS

*Please note that the graphs represent the addresses of memory accesses across 50,000 clock cycles. Our simulations were done with eight banks, the first four of which are shown for each test.*
Scheduling Policies

- **First-ready first-come-first-serve (FR-FCFS)**
  - **Ideas**
    - Baseline scheduling policy
    - Issues requests based on if a row hit can be serviced
    - Prioritizes oldest arrival time, regardless of who is the requestor
    - In a parallel setting, the number of requests serviced is for a processor proportional to the number of requests the processor makes
  - **Results**
    - Heavily favors libquantum, due to its high row hit potential and great spatial locality
    - Causes disproportionate slowdowns while running with programs with less frequent accesses

- **Stall-time fair memory (STFM)**
  - **Ideas**
    - Attempts to compute slowdown based on comparing its memory stall time and an estimated stall time if it were running alone
    - Ranks each processor based on its slowdown calculation
    - Prioritizes the requests from the processor if its slowdown reaches a predetermined threshold
    - If the threshold is not reached, FR-FCFS is used
  - **Results**
    - Balances memory access latency, but can may not be as fair in terms of overall slowdowns in memory intensive programs when run with non-memory-intensive programs
• Parallelism-aware batch scheduling (PAR-BS)
  o Ideas
    • Aims to increase throughput by finishing requests for a single processor
    • Forms batches of memory requests for each processor
    • Ranks batches based on lowest load of the processor
  o Results
    • Erratic batch handling between processors can cause row buffer thrashing
    • In similarly low memory intensive program groups, can cause slowdowns
    • Ignores potential row hits altogether, increasing overall memory latency

• Priority based fair scheduling (PBFS)
  o Ideas
    • Aims to increase the fairness of issued requests during times of contention
    • Ranks processors based on duration since its last request
    • Priority resets as a processor becomes idle
    • Does not attempt to provide fairness between periods of idleness
  o Results
    • Programs with different behavior “phases”, like namd, are not balanced well
    • Punishes programs that make frequent requests with a disproportionate slowdown when other programs send requests
    • Ignores potential row hits altogether, increasing overall memory latency
Results – Memory Access Latencies

Results – Slowdown
Results – Row Hits

Acknowledgements

We would like to thank Dr. Todd Mowry for his support, encouragement and assistance. In addition, we would like to express our gratitude to Dr. Onur Mutlu, Lavanya Subramanian and Yoongu Kim for providing us with the simulation environment.