Module-4
CSEN 3104
Lecture 31
21/10/2019

Dr. Debranjan Sarkar

Vector Memory-Memory versus Vector Register Machines

- Vector memory-memory instructions hold all vector operands in main memory
- The first vector machines, TI ASC (1971), and CDC Star-100 (1973) were memory-memory machines
- Cray-1 (1976) was first vector register machine

- Example Source Code
- for (i=0; i<n; i++)
- {
- c[i] = a[i] + b[i];
- d[i] = a[i] – b[i];
- }

Vector Memory-Memory versus Vector Register Machines

- Vector Memory-Memory code
- ADDV   C, A, B
- SUBV   D, A, B

- Vector Register code
- LV   V1, A
- LV   V2, B
- ADDV   V3, V1, V2
- SV   V3, C
- SUBV   V4, V1, V2
- SV   V4, D

Vector Memory-Memory vs. Vector Register Machines

- Vector memory-memory architectures (VMMA) require greater main memory bandwidth
  - Because all operands must be read in and out of memory
- VMMAs make it difficult to overlap execution of multiple vector operations
  - Because dependencies must be checked on memory addresses
- VMMAs incur greater startup latency
  - Scalar code was faster on CDC Star-100 for vectors < 100 elements
  - For Cray-1, vector/scalar breakeven point was around 2 elements

Cache Coherence

Cache Coherence Problem

- In a memory hierarchy for a multi-processor system, data inconsistency may occur between adjacent levels or within the same level
- For example, cache and main memory may contain inconsistent copies of the same data object
- Multiple copies may possess different copies of the same memory block because multiple processors operate asynchronously and independently
- Caches in a multi-processing environment introduce the cache coherence problem
- When multiple processors maintain locally cached copies of a unique shared-memory location, any local modification of the location may result in a globally inconsistent view of memory
- Cache coherence schemes prevent this problem by maintaining a uniform state for each cached block of data

Cache inconsistency: Causes

- Maintaining cache coherency is a problem in multiprocessor system when the processors contain local cache memory (multiple private caches are used)
- 3 sources of the problem are identified:
    - Sharing of writable data
    - Process migration
    - I/O activity
- Inconsistency in sharing writable data (Show figure)
- Consider a multiprocessor with 2 processors, each with private cache and both sharing the main memory
- Let X be a data element which has been referenced by both the processors
- Before update, the three copies of X are consistent
- Under write-through policy, inconsistency occurs between the 2 copies in the 2 caches

Cache inconsistency: Causes

- Inconsistency due to process migration (Show figure)
- Inconsistency occurs in case of both the write-back and write-through policies
- Special precautions must be exercised to avoid such inconsistencies
- A coherence protocol must be established before processes can migrate safely from one process to another
- Inconsistency during IO operation bypassing cache (Show figure)
- When the IO processor loads a new data X’ into the main memory, bypassing the write-through caches, inconsistency occurs between the cache and Main Memory
- When outputting a data directly from the shared memory (bypassing the caches), the write-back caches also create inconsistency

Cache inconsistency: Solutions

- Solution to IO inconsistency problem (Show figure)
- Attach the IO processor (IOP1 and IOP2) to the private caches (C1 and C2)
- Thus the IO Processors share caches with the CPU
- The IO inconsistency can be maintained if cache-to-cache consistency is maintained via the bus
- Shortcomings:
  - Likely increase in cache perturbations
  - Poor locality of IO data
  - These may result in higher miss rates

Thank you