Abdul Dakkak

CS533 : HW1

Question 1

- For an application with 10% sequential code, 30% 3-way parallel code, and 60% infinity parallel code the generalized amdahl's law would be:

$$S(n,r) = \frac{1}{\frac{1-f}{perf(r)} + \frac{f}{pref(r)+n-r}}$$

$$S(n,r) = \frac{1}{\frac{0.1}{pref(r)} + \frac{0.3}{pref(r)+min(n,3)-r} + \frac{0.6}{pref(r)+n-r}}$$

$$= \frac{1}{\frac{0.1}{\sqrt{r}} + \frac{0.3}{\sqrt{r}+min(n,3)-r} + \frac{0.6}{\sqrt{r}+n-r}}$$

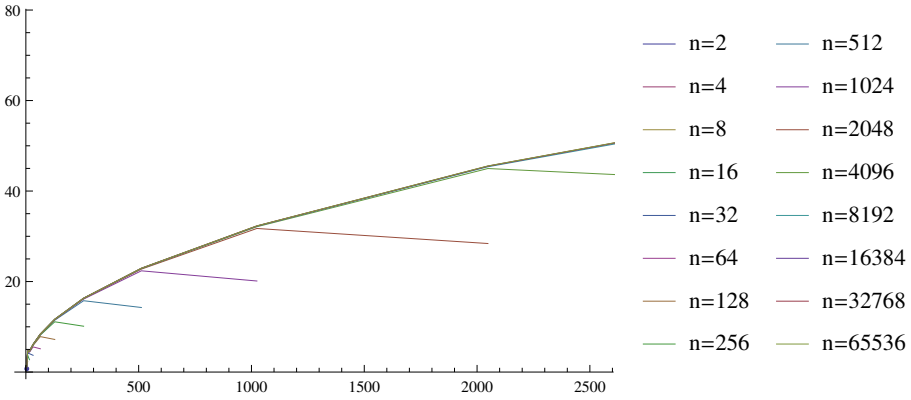

Figure 1: This plot shows how the speedup behaves as we increase r for different n values.

- The optimal r for $n = 2$ is 2, since for $r = 1$ we get a speedup of 0.689 while $r = 2$ we get a speedup of 0.744
- The optimal r for $n = 4$ is 2, since for $r = 2$ we get a speedup of 0.993 while $r = 4$ we get a speedup of 0.9

- The optimal r for $n = 16$ is 8

In general, as we increase n the sequential part of the code dominates, so it makes more sense to increase the number of BCs allocated for serial execution.

Question 2

- (a) *Coherence* — In a multi-processor system, modifications to the cache done in one processor are reflected in all other processors. Note that unlike consistency, the coherence does not state when the writes will become visible.
- (b) *Memory consistency model* — a contract between the programmer and the system that states that if the programmer follows specific rules, then writes to different memory locations will appear in an order that makes sense.
- (c) The relation between coherence and a memory consistency model is that consistency addresses when a processor sees writes to different locations while coherence addresses when processor writes are seen when modifications happen to the same memory location.
- (d) One way they come up in uni-processor system is consistency/coherence in a cluster configured with shared memory space, since a cluster configured with shared memory space essentially represents system RAM as the node's cache which must be consistent with other node memory states.
- (e) Without low level access to the cache, which is not possible, it is not possible to know whether your memory is cache coherent with just loads.
- (f) It is not possible to tell if the hardware is SC, but it is possible to introduce a race condition and deduce that the hardware is TSO.

Question 3

MESI

- 1: P1 (Exclusive), P2 (Invalid), P3 (Invalid) – 30 cycles for memory load
- 2: P1 (Modified), P2 (Invalid), P3 (Invalid) – no cycles
- 3: P1 (Invalid), P2 (Modified), P3 (Invalid) – 20 cycles for read
- 4: P1 (Invalid), P2 (Modified), P3 (Invalid) – no cyles
- 5: P1 (Shared), P2 (Shared), P3 (Invalid) – 20 cycles for read
- 6: P1 (Shared), P2 (Shared), P3 (Shared) – 20 cycles for read

- 7: P1 (Invalid), P2 (Write), P3 (Invalid) – 4 cycles for invalidation bcast
- 8: P1 (Shared), P2 (Shared), P3 (Invalid) – 20 cycles for read
- 9: P1 (Shared), P2 (Shared), P3 (Shared) – 20 cycles for read
- 10: P1 (Invalid), P2 (Invalid), P3 (Shared) – 4 cycles for invalidation bcast
- 11: P1 (Invalid), P2 (Shared), P3 (Shared) – 20 cycles for read

In this MESI scheme, 8 cycles are for invalidation broadcasts, 120 cycles are for reads satisfied by another cache, and 30 cycles are for reads from main memory

DEC Firefly

- 1: P1 (Exclusive), P2 (Invalid), P3 (Invalid) – 30 cycles for memory load
- 2: P1 (Dirty), P2 (Invalid), P3 (Invalid) – no cycles
- 3: P1 (Shared), P2 (Shared), P3 (Invalid) – 20 cycles for read
- 4: P1 (Shared), P2 (Shared), P3 (Invalid) – no cyles
- 5: P1 (Shared), P2 (Shared), P3 (Invalid) – 22 cycles for read
- 6: P1 (Shared), P2 (Shared), P3 (Shared) – 22 cycles for read
- 7: P1 (Shared), P2 (Shared), P3 (Shared) – 22 cycles for update
- 8: P1 (Shared), P2 (Shared), P3 (Invalid) – no cyles
- 9: P1 (Shared), P2 (Shared), P3 (Shared) – no cyles
- 10: P1 (Shared), P2 (Shared), P3 (Shared) – 22 cycles for update
- 11: P1 (Shared), P2 (Shared), P3 (Shared) – no cyles

In this DEC scheme, 20 cycles are for reads satisfied by another cache, 88 cycles are for updates, and 30 cycles are for reads from main memory.

Question 4

- (a) Intel Core i7 — uses a directory to maintain a list of L2 caches containing a block and and uses a directory to maintain all blocks in the L3 cache.
- (a) DASH Multiprocessor — used a heirartical coherence model where processors in the node used a snooping protocol, but across nodes a directory based protocol was used.
- (b) QPI is a hybrid design. It is a point-to-point protocol similar to a directory based method, but snooping is still used on the home network.
- (c): Number of blocks in memory is $2^3 1/2^7 = 2^2 4$
- Dir_N – $256 = 32 * 8$, $Dir_N = 32/128 = 25\%$
- Dir_4CV – $256/4 = 64$, $Dir_4CV = 8/128 = 6.25\%$

- Dir_N – $8n/128 = 1 => n = 128/8 = 16$
- Dir_4CV — $8n/4/128 = 1 => n = 128/2 = 64$

Question 5

- No Sharing — Both are bad, since both would cause unnecessary data to be sent on the bus.
- Pipe — Invalidation based protocols is best, since we will only invalidate one other node (one producer one consumer)
- Multiple write Pipe — Invalidation based protocol is best, since in the update based protocol would send the data on each write (even though only the last write is needed).
- Irregular — It really does not matter which method is used. Invalidation based protocol is slightly better, since the access pattern is not known and update based protocols are only good when there is a single producer multiple consumer configuration.
- The exclusive state cuts down on bus traffic when memory is only being read and then written. For example, if a processor reads data and then writes, then the processor need not invalidate the cache block of the other processors and goes to modified state silently.

Question 6

- No, the memory consistency model is a contarct between the hardware and the programmer. This means that the programmer or compiler is limited from reordering certain instructions.
- The two things memory consistency determines are:
- Whether a processor can read its own writes before any other processor sees them
- Whether the interconnect may reorder messages
- The $W \rightarrow R$ relaxation provides the most benifit, since one can hide the latency of write operations.

Question 7

- execute the code in the following order:

```
1 flag[0] = true                    flag[1] = true
   <*>                               <*>
2 while (flag[1] == true) {         while (flag[0] == true) {
     ...                                 ...
3 enter critical section            enter critical section
```

- The way to fix this is to place a memory fence instruction at `<*>` the reason this is needed is because when one reads `flag[1]` and writes `flag[0]` then these are two memory addresses which the system is free to reorganize.

Question 8 One interleaving which produces `{1, 0, 1, 0}` is the following. It is illegal because we read the value of `a` before invalidating `a` and this can be caused by network lag when one processor gets the messages out of order with other processors.

```
P0:               P1:                          P2:
  a = 1             r1 = ld b                    r3 = ld a
  b = 1             MB (recieve invalidate b)    MB (recieve invalidate a)
                    r2 = ld a                    r4 = ld b
                       (recieve invalidate a)         (recieve invalidate b)
```