Module 12 Example set 4

1. What is the difference between strong scaling and weak scaling?
Strong scaling refers to using multiple processors to speedup the execution of a program for a fixed problem size. Weak scaling on the otherhand, increases the number of processors as the problem size increases.

2. Suppose that f is the fraction of the execution time for a program that is due to some operation that can be split into n independent parts. The remaining fraction of the execution time, (1-f), is due to the purely sequential part of the program. Based on Amdahl's law, what is the expresion that gives the speedup for the program provided by using n processors or cores compared to using a single processor or core.

The speedup would be $T1 / T_n$
Where T1 is the execution time using a single processor or core, and
T_n is the execution time using n processors or cores.
According to Amdahl's law $T_n = Ti*[(1-f) + f/n]$
Hence the speedup = $Ti / (T1*[(1-f) + f/n]) = 1 / (1-f) + f/n$

3. What is the upper limit on the speedup for the program, described in problem 2 above, as the parallel part of the program is split into more and more separate independent parts?
The expression f/n gets smaller and smaller with increasing values of n and approaches 0 as n approaches infinity. Hence the speedup approaches $1/(1-f)$.

4. Suppose that it is not possible to subdivide the parallel part of a program into more than 8 independent parallel parts. If the system contains 32 cores, then each of the 8 independent parts would be handled by a separate core. How can the remaining 24 cores be used to advantage?
One way to take advantage of the extra cores is to expand the problem so that after the sequential part of the program is executed, four groups of 8 cores can each be applied to four different datasets as opposed to the single dataset That is, weak scaling can be used.

5. A certain program consists of a sequential part that requires a time s to complete and N parallel parts each of which takes the same time p to complete.

a) Write down an expression for the total time required to complete the entire program using a single processor.

Ti = s + Np

(after completing the sequential part, the processor would execute the N parallel parts one after the other)

b) Write down an expression for the total time required to complete the entire program using N processors.

TN = s + p

since all parallel parts are executed at the same time and each takes time p, the total time is the sum of the sequential time s plus the time p for the parallel parts.

c) Write down an expression for F, the fraction of the total execution time that is accounted for by the parallel part.

The time for the parallel part is p and the total time is s + p.

So F = p / (s + p).

d) Write down an expression for the speedup provided by using N processors as a function of F (the fraction of the total execution time that is accounted for by the parallel part).

Speedup = Ti / TN = (s + Np) / (s+p) = s/(s+p) + N* p/(s+p)

Part c) above shows that F = p / (s+p)

Hence 1 - F = 1 - p/ (s + p) = [(s+p) - p] / (s+p) = s / (s+p).

Therefore speedup = 1 - F + N*F

This is known as Gustafson's law.

6. A uniprocessor system accesses memory over a 32-bit bus. The processor has a write-back (as opposed to write through) data cache employing a write-allocate policy. The data cache ooperates in look-through mode, uses true LRU replacement and contains 65536 lines each of which is 1024 bytes in size. Each data cache access takes 2 CPU clock cycles to either detect a cache miss, or to perform a cache read or to perform a cache write. Loading a memory block into cache or writing a cache line back to memory takes 400 CPU clock cycles. Recall that for a cache miss the data item that is needed is read in parallel with loading the memory block containing the data item into cache. The operating system flushes the contents of cache to memory after a program terminates.

The code shown below reads and updates each of the 134217728 four-byte elements in an integer array. The address of the array in memory i Ox1008A400

```
        lui    $8,0x1008     # point to first element to be processed
        ori    $8,0x,A400
        lui    $4,0x0800     # number of elements = 134217728 (=Ox08000000)
loop:   lw     $12,0($8)     # get next element
        addi   $4,$4,-1      #  decreme nt loop control variable
        sub    $12,$0,$12    # negate by subtracting from 0
        addi   $8,$8,4       # point to next element
        SW     $12,-4($8)    # update element that was read
        bgez   $4,loop       # repeat if more eleme nts remain
        nop
```

a) What would be the data cache hit ratio for this code running on the uniprocessor system? Assume that the data cache is initially empty.

Since the line size is 1024 bytes, each line holds 256 elements. The data cache can hold 65536*256 = 16777216 array elements (1/ 8th of the 134217728 total). The code steps sequentially through the array elements and causes a data cache miss on the initial reference to the first element in each line. The 256 elements in a line are referenced twice (a read followed by a write). Hence for each line there will be 2*256 = 512 references and one miss (i.e., 511 hits and 1 miss). The entire array corresponds to 134217728/ 256 = 524288 lines.

So the data cache hit ratio = (524288*511) / (524288*512) = 511/512 = 0.998 or 99.8%.

b) A separate copy of this code is executed on each of the cores in an 8-core system. Each core has a separate data cache that is identical to that of the uniprocessor system and the code on each core uses the appropriate starting address and loop limit that would correspond to 1/8th of the array. What would be the data cache hit ratio for the code running on each of the 8 cores? Assume that all data caches are initially empty.

The code on each core accesses its 1/8th of the array (524288/8 = 65536 lines) sequentially. So the data cache hit ratio for each core = (65536*511)/(65536*512) = 511/512 = 0.998 or 99.8%. This is the same as the data cache hit ratio seen by the uniprocessor.