SOLUTION NOTES

Comparative Architectures 2003 Paper 8 Question 1 (IAP)

The data sheet of a new workstation CPU tells you simply that it has a 8KB L1 data cache and 512KB L2 cache. As an inquisitive computer scientist, you wish to learn more about the system's memory hierarchy.

(*a*) Devise a method for determining the cache line size. You may assume that the L1 and L2 caches use the same line size, and that the operating system provides a microsecond accurate time function. Provide pseudo code of any test programs you would use, and describe how you would interpret the results. [6 marks]

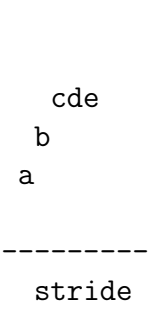
The cache line size will be a power of 2 somewhere between 16 and 256 bytes. Our strategy will be to measure the time taken to perform N read accesses to an array of size 'size', striding through it 'stride' bytes at a time.

*We pick 'size' to be significantly larger than the L2 cache size e.g. 2*1024*1024. We pick N such that we expect the test to run for at least 100ms to mitigate timing errors: e.g. if we assume a miss is 50ns, we'd need 2M to run for 100ms. Worst case is the line size is 256 bytes and we're using stride 16. Hence N = 32million. We execute read_test with stride sizes of 16,32,64,128,256 and plot the time taken.*

```
int read_test(N, size, stride)
{
  volatile char arr[size]

  flush_cache();  // e.g. memset of very large array

  start = get_time_in_microsecs()

  for(i=0,j=0;i<N;i++)
  {
    (void)arr[j];  // do the access
    j+=stride;
    if(j>=sizeof(arr)) j=0;
  }

  stop = get_time_in_microsecs();
  return stop-start;
```

```
}
```

We would expect to see a graph that looked something like:



In this case, the cache line size would be identified by point 'c', 64 bytes. This is the first stride where each access is causing a new cache line to be fetched from main memory, whereas previously the spatial locality was resulting in fewer accesses and hence reduced run time.

(*b*) Describe how you produce an accurate estimate for the load latency incurred by accesses to the two caches and main memory. [6 marks]

Load latency can be measured by preparing a circular chain of interdependent memory access (rather like a linked-list). Having established the line size, we can ensure that each access is to a different cache line. To measure L1 cache latency, we need to ensure the whole list can fit in the L1 cache at the same time. To measure L2 latency we need a list that is big enough to not fit in the L1, but will fit in the L2. To measure main memory latency, we use a list too big to fit in the L2.

```
long arr[2*1024*1024];

prepare_list( size, stride )
{
  n = stride/sizeof(long);

  for (i=0; i<(size/sizeof(long))-n; i+=n)
  {
  arr[i] = &arr[i+n]; // address of location arr[i+n]
  }
```

```
  arr[i] = &arr[0];
}


chase_list(N)
{
long *p;

// could do a few iterations to prime the caches, but probably OK anyway

start = get_time_in_microsecs()

p=&arr[0];
for(i=0;i<N;i+=N)
  {
  p= (long*) *p; // unroll a bit
  p= (long*) *p;
  p= (long*) *p;
  p= (long*) *p;
  p= (long*) *p;
  p= (long*) *p;
  p= (long*) *p;
  p= (long*) *p;
  }

  stop = get_time_in_microsecs();

  latency_in_nanosecs = (1000*(stop-start))/N;
}
```

To calculate the latency for L1, L2 and L3 respectively:

```
prepare_list(8*1024, <stride> ); chase_list(2000000);
prepare_list(256*1024, <stride> ); chase_list(2000000);
prepare_list(2*1024*1024, <stride> ); chase_list(2000000);
```

(*c*) Outline a method could you use to determine the associativity of the caches.
[8 marks]

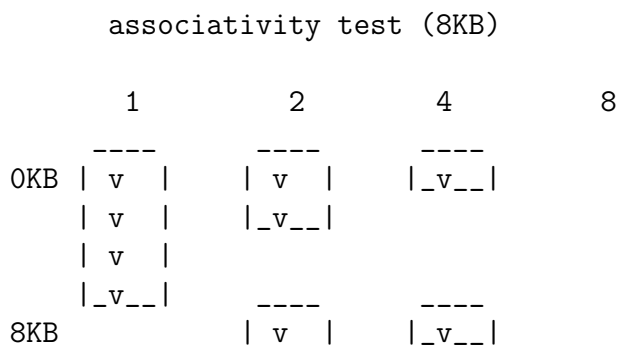
The associativity will be a power of 2, probably somewhere between 1 and 8. We can detect it by deliberately creating a situation where aliasing and hence conflict misses occur.

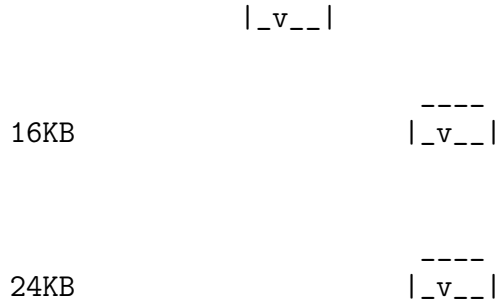
For the L1 cache case, we will still use a list of 8KB/cache line size entries, but for different associativity tests we divide the array up into N parts and place them on addresses boundaries that may map to the same line in the cache.

If list elements do map to the same line, the latency reported will be much higher as the fetch will have to happen from the next level in the memory hierarchy.

```
assoc_prepare_list( total_size, assoc, stride )
{
  n = stride/sizeof(long);
  size = total_size / assoc;

  for (j=0; j<assoc; j++)
  {
    base=(total_size/sizeof(long)) * j;
    for (i=0; i<(size/sizeof(long))-n; i+=n)
    {
      arr[base+i] = &arr[base+i+n];
    }
    arr[base+i] = &arr[(total_size/sizeof(long)) * (j+1)];
  }
  arr[(total_size/sizeof(long)) * (assoc-1) + (size/sizeof(long))-n ] = &arr[0];
}
```

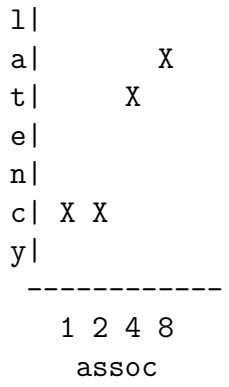
For the L1 case:






```
assoc_prepare_list( 8*1024, 1, <stride> ); chase_list(2000000);
assoc_prepare_list( 8*1024, 2, <stride> ); chase_list(2000000);
assoc_prepare_list( 8*1024, 4, <stride> ); chase_list(2000000);
assoc_prepare_list( 8*1024, 8, <stride> ); chase_list(2000000);
```

We would expect to get a graph something like the following:



This indicates that the associativity of this level of this cache is equal to 2 – if more than 2 items alias to the same location the measured latency increases as the lines must be brought from the next level of the memory hierarchy.