Scientific Computing

Assignment 3

9/25/2013

1) What are the cache hit ratios c_h for

- stride s = 1
- stride s = 2
- stride s = 4
- stride s = 8

.

.

.

- stride s = L
- stride s = 2*L

Solution :

Consider a 8 line cache consisting a 4 word line that means the total size of the cache is 8*(4) = 32 bytes. Here we have considered that each line is considered as 4 Words.

Consider the function to be a vector update which would be ideally be as follows :

```
for(i=0,i=+s,i<n-1)
{
        X[i] = x[i] *2;
}
```

Stride s = 1

Initially the cache would be empty hence the first element would be element.
Hence initially it would be a miss but the next time it would load the four words. Since there are 3 hits out of 4 the cache hit ratio would be as below

$C_h = 3 / 4 = 0.75$, therefore the cache hit ratio is 75 %

Stride s = 2

Now the stride is 2 therefore now two entries would be missed therefore the cache hit ratio is Given as below

$C_h = 2 / 4 = 0.5$, therefore the cache hit ratio is 50 %

Stride s = L

For this since all would be misses the cache hit ratio would be 0.

Similarly we can derive the below generalized formula as

$c\_h = (L – s) / L$

---

2. Suppose that for stride 1, the scaling operation runs at G Gflops/second. A central principle in scientific computing is that performance is determined by data access, not computation. If that is the case, what computational performance (in, e.g., Gflops/second) should be expected for stride s=2? As the stride increases, for some value of s when doubling it to to 2*s the performance should not be expected to decrease. What value of s is that?

Solution :

We can establish the relation between stride (S) and computational performance. It is clear that as stride increases the computational performance decreases since the number of memory accesses increases. This can be represented mathematically as below :

Computational Performance (Gflops) is inversely proportional to Stride (s)

We are have been given the below formula

t_eff = c_h * t_cache + (1-c_h)*t_mem

t_eff = (L-s) / L * t_cache + (1 – ((L-s)/L))*t_mem

Further simplifying we get the following formula

t_eff = ((L-s)*t_cache + s * t_mem) / L

In order to find Gflops we need to inverse this equation

G = L / ((L-s)*t_cache + s * t_mem)

Let us find Gflops for Stride s = 1 , which would be denoted as G1

G1 = L / ((L-1) * t_cache + t_mem))

Similarly for Stride s = 2, which is G2

G2 = L / ((L-2) * t_cache + t_mem))

G1 / G2 = (((L-2)*t_cache + 2*t_mem) / (L-1)*t_cache + t_mem))

(L-2) * t_cache is much greater that t_mem

Hence we get the value as G1/G2 = 2

This proves that the computational performance for stride 2 is 50 % of stride 1.

Now coming back to the second question the value of s for which performance is determined by data access is L because at L c_h is 0 and for any other values greater than L it would be 0.

---

3. If a daxpy operation is used instead of the vector update one, how if at all would that change the cache hit ratios found above? Would it have any effect on the value of which stride that causes no further drop in performance when it is doubled?

Solution :

Since we are dealing with memory access here the type of transaction and the number of repetitions are immaterial and so the cache hit ratios remain the same even in the case of daxpy operation.

---

4. The vector scaling code can easily be written and timed. Timing it gives the effective memory access time teff, and finding the cache model parameters tmem and tcache correspond to cache hit ratios of ch = 0 and ch = 1. Making the stride large enough guarantees every access is a cache miss, i.e,. ch = 0. But even a stride of 1 does not give ch = 1. Given teff, tmem, and L, give a formula that gives tcache.

Solution :

Let assume stride s = 1 ,

t_eff = c_h * t_cache + (1 - c_h) * t_memory

c_h = (L-1) / L
1 - c_h = 1 / L

t_eff = (L-1) * t_cache / L + t_memory / L

L * t_eff = (L-1) * t_cache + t_memory

(L-1) * t_cache = L * t_eff - t_memory

$$t\_cache = (L * t\_eff - t\_memory) / (L-1)$$

Sources :

1) High Performance Computing , O'Reilly Publication