Programming Assignment № 1

Meet Pragnesh Shah | 13D070003 | ME 766 23/01/2016

1 Problem 1

1.1 (a)

We are given that the serial portion S of the algorithm constitutes of 20 % of the total computations. Let's assume the number of processors to be P and the parallelizable part of the code to be C. Thus we can say that the

$$\begin{aligned} Speedup(\gamma) &= \frac{T_1}{T_p} \\ &= \frac{1}{S + \frac{C}{P}} \end{aligned}$$

Now we need to find the maximum possible speedup i.e the speedup when number of processes are infinte. It is hence equal to

$$\begin{aligned} Max.Speedup &= \lim_{P\to\infty} \gamma \\ &\approx \frac{1}{S} \\ &\approx \mathbf{5} \end{aligned}$$

1.2 (b)

We need to find the serial part of the computation in the algorithm given the desirable maximum speedup γ as **50**.

Hence using the above formula of maximum achieveable speedup we get :

$$\begin{aligned} S &= \gamma^{-1} \\ &\approx \frac{1}{50} \\ &\approx 0.02 \\ &\approx \mathbf{2\ \%} \end{aligned}$$

2 Problem 2

2.1 (a)

We are given the following specifications for the hypothetical Von Neumann Machine :

$$\begin{aligned}
CacheSize &= 1024bytes \\
MemorySize &= 10 * 1024 * 1024bytes \\
CachetoMemoryBlockSize &= 1024bytes \\
FetchTime(fromMemory) &= 150CPUcycles \\
FetchTime(fromcache) &= 1CPUcycles
\end{aligned}$$

We are given a $2D$ array **A[i][j]** of size $1024 * 1024$ occupied by **ints** each of which needs 4 bytes assuming standard **int_32**.

Thus total memory in bytes we need to access from the main memory is :

$$Size(A) = 1024 * 1024 * 4bytes$$

Since we have a cache that can have a block of size 1024 bytes at any instance , we can assume that when the loop executes from the beginning and tries to access the element **A[0][0]** a block of memory containing elements **A[0][0]** to **A[0][256]** is copied to the cache from the main memory as the block size of the above data is $256 * 4bytes$ = $CacheSize$ = $CachetoMemoryBlockSize$. This assumption only holds true since **A** is an array and by the nature of it the data is stored in continuous memory registers unlike other complex data structures.

Thus the cost to transfer the entire array **A** is :

$$\begin{aligned}
TimeCost &= \frac{Size(A)}{CachetoMemoryBlockSize} * [150 + \frac{CacheSize}{Size(int)} * 1]CPUcycles \\
&= 1662976CPUcycles
\end{aligned}$$

2.2 (b)

The difference between **FORTRAN** and **C** is that the former stores data in a **column major** manner and the latter uses **row major** system to store arrays in memory. Since the cache retrieves a block of data from the memory and the computation in the given code is done row-wise, similar code in **FORTRAN** will perform poorly as the number of cache misses will be very high and the blocks will have to be copied from memory to cache for every element in the 2D array. For each element $A_{i,j}$ the cache contains elements in the same column upto $A_{i+256,j}$, but unfortunately they are of no use to us as the computation is row wise. Thus the cost to transfer the entire array **A** in **FORTRAN** will be :

$$\begin{aligned} TimeCost &= \frac{Size(A)}{Size(int)} * [FetchTime(fromMemory) + FetchTime(fromCache)] \\ &= \frac{1024 * 1024 * 4}{4} * [150 + 1] \\ &= 158334976 CPU cycles \end{aligned}$$