CS6560: Assignment2

Abhishek Yadav, CS12B032

February 16, 2016

Q1

My system is running on 4GB RAM. The size of the matrices as specified in the assignment statement is 10^5x10^5, which amounts to a total of $3 * 10^5 * 10^5 * 4 \approx 2^{36}$ Bytes = 64GB of memory. In the best case, we could store the sum of two cells in one, say A[i][j] = A[i][j]+ B[i][j] but that would also require $\approx$ 32GB of memory. And moreover when trying to allocate these chunks the compiler throws the error message- **"error: size of array 'X' is too large"**, for all allocation attempts. Hence, I have assumed, the size to be $10^4 * 10^4$.

(a) Compile the program using: gcc -o exec_name prog_name.c -lpthread

```c
#include <stdio.h>
#include <pthread.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#define N 10000
int A[N][N];
int B[N][N];
int C[N][N];

void *calc_sum(void * p){
    int x=(int)p;
    int i,j;
    for(i=x-1000; i< x; i++)
        for(j=0;j<N;j++)
            C[i][j]=A[i][j]+B[i][j];
}

int main(){
    pthread_t threads[10];
    int return_ids[10];
    int i,j;
    srand(time(NULL));
    for(i=0;i<N;i++){
        for(j=0;j<N;j++){
            A[i][j]= rand()%100;
            B[i][j]= rand()%100;
        }
    }
    j=1000;
    for(i=0;i<10;i++){
        return_ids[i]= pthread_create(&threads[i], NULL,calc_sum, (void*)j);
        j+=1000;
    }

    for(i=0;i<10;i++)
        pthread_join(threads[i],NULL);
    exit(EXIT_SUCCESS);
}
```

(b) compile using: gcc -o exec_name prog_name.c -fopenmp

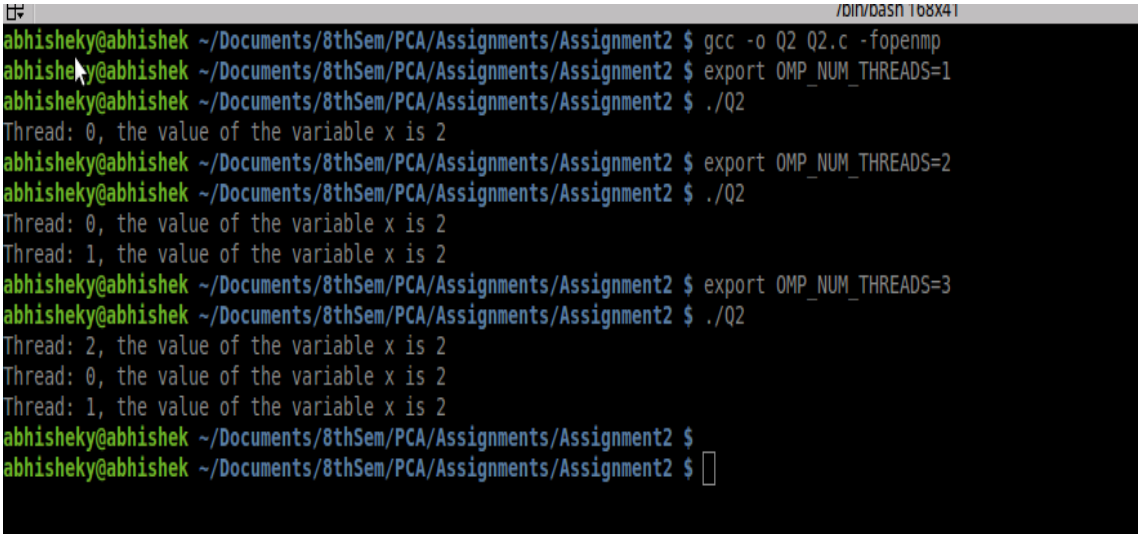
```c
#include <omp.h>
#include <stdio.h>

#define N 10000
int A[N][N];
int B[N][N];
int C[N][N];

int main(){
    int i,j;
    srand(time(NULL));
    for(i = 0; i < N; i++)
        for(j = 0; j< N; j++){
            A[i][j]= rand()%100;
            B[i][j]= rand()%100;
        }

    #pragma omp parallel num_threads(10)
    {
        int i,j;
        for(i = 0; i < N; i++)
            for(j = 0; j < N; j++)
                C[i][j] = A[i][j]+ B[i][j];
    }

    return 0;
}
```

(c) To reduce the cache-misses we must use only 1 thread, which will obviously make the program slower but as compared to using more than 1. Since, cache can accommodate 4096 integers at a time, given no block size and cache type, we can assume that when first miss is incurred for respective location of A,B and C, a block of ≈4096/3 int-size will be brought into the cache and this process repeats till the process completes the execution.

(d) Even if hardware has 8 threads I won't change my implementation if the concern is to reduce the cache misses and use the policy as suggested in part c. But, if execution speed/time is a concern then I'll spawn 8 threads one for each hardware thread. Because, spawning more than 8 is useless as the parallelism that can be achieved using more than 8 threads can be achieved using 8 as well in the above program because at a time only 8 threads would be running and rest will remain idle. So, using more than 8 threads will be same as alternating the threads to execute the same task. And if spawned less than 8 threads, resource utilization will be less and hence will result in slow computation as compared to using 8 threads.

Q2

Output of the successive runs of the programs for different values of the environment variable OMP_NUM_THREADS:



The statement **X=0**; can either be removed or put before print statement to reflect the changes in the value of X due to assignment. Since threads run in parallel, they get the same value of X as 2 as defined in the outer scope.

Q3

Transcoding: It involves encoding already compressed video streams to better match the storage and decoding abilities of the target device.

Intel's Quick Sync Layer is Intel's hardware video encoding and decoding technology. It is used for faster transcoding of videos from one format, say **DVD** or **MP4** to a format appropriate to a target device. Unlike video encoding on GPUs, Quick Sync is a dedicated hardware core on the processor chip which allows for a faster and more power efficient video processing. There is a trade-off between the speed and quality of results obtained while using hardware accelerated video encoding technologies than with the CPU only encoders and hence quality of results obtained using Quick Sync is lower. Some of the features of Quick Sync layer are as follows:

- Quick Sync comes with Multi-Format Codec(MFX) which is a dedicated parallel engine and supports MPEG2, VC1 and AVC standards. It also supports Multi View Coding(MVC) for 3D.
- Video decoding takes place fully on MFX and which no longer uses EU arrays.
- High performance Videos decoder supports multi-stream with frame based context-switch, contains well balanced streaming data pipe and takes frequency advantage on 32nm process technology.
- Video processing accelerators are equipped with high-quality video-scaling, denoise-filtering, film-mode-detection, detail enhancement filtering and achieves high throughput maintaining programmable flexibility.
- High performance Video Encoders contain programmable EU arrays which is assisted by high-throughput VME(Video Motion Estimator) in Media Sampler. Parallel MFX engine supports AVC formats, shares decode logic through reconstruction, provides high-throughput.

Software tools using the Quick Sync technology:

1. CyberLink PowerDirector
2. CyberLink MediaEspressor
3. Core Digital Studio
4. Roxio Creator
5. Arcsoft MediaImpression
6. Arcsoft MediaConverter
7. MainConcept