

## Task 1: Using multiple threads for speed-up

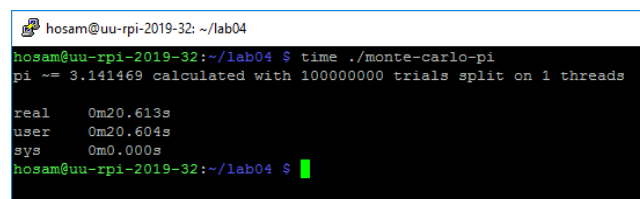
**Question 1.1:** *Does the above Monte Carlo algorithm for estimating the value of  $\pi$  seem like a good candidate for speed-up with parallelization using several threads? Clearly explain and motivate your answer*

**Answer to Question 1.1.** Yes, if the number of threads is equal to the number of cores, the Monte Carlo algorithm produces more random points at the same time. Therefore, the Monte Carlo should run faster or by increasing the total amount of the random points, there is a possibility to get more precise  $\pi$  value. However, if the number of the threads is greater than the number of cores, the performance remains the same as running with number of threads equal to number of the cores.

**Question 1.2:** *How long does it take for the process to finish in wall clock time with one thread? How much CPU time has the process consumed in total? What could you observe about the load on the CPU cores from htop?*

**Answer to Question 1.2.**

In wall clock, the Monte Carlo process took around 0m20s to finish and consumed 0m20s of CPU time using one thread. As shown in Figure1, it was clearly observed that using multi-threads in Monte Carlo process speeds up the processing e.g. when using 1, 2 and 4 threads it will utilizes 1, 2 and 4 cores in the RPI3 respectively and the CPU time consumed by the process is 20s divided by the number of threads has been used. Furthermore when using number of threads more than the number of available cores in the RPI3 (e.g. 4 threads), the result remains the same as using 4 threads, for instance when using 8 threads, the consumed CPU time is also around 5s.



```
hosam@uu-rpi-2019-32: ~/lab04
hosam@uu-rpi-2019-32:~/lab04 $ time ./monte-carlo-pi
pi ~= 3.141469 calculated with 100000000 trials split on 1 threads

real    0m20.613s
user    0m20.604s
sys      0m0.000s
hosam@uu-rpi-2019-32:~/lab04 $
```

Figure 1: CPU Time consumed by Monte Carlo process using one thread.

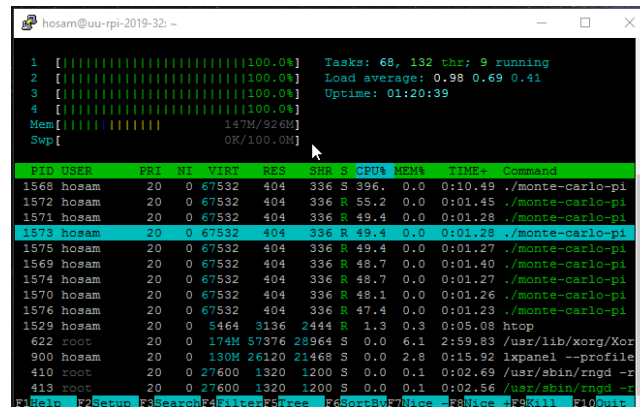


Figure 2: CPU Utilization When running Monte Carlo process with 4 threads.

**Question 1.3:** *How long does it take for the process to finish in wall clock time with 2, 4, or 8 threads? How much CPU time has the process consumed in total in each case? What could you observe about the load on the CPU cores from htop?*

**Answer to Question 1.3.** For 1, 2, 4 and 8 threads the approximate consumed CPU time is 20s, 10s, 5s and 5s respectively.

**Question 1.4:** *Explain all the above observations as clearly as possible, taking into account the number of processor cores on the Raspberry Pi.*

**Answer to Question 1.4.** In multicore CPUs, when using single thread it will run on one core, and this affect the performance greatly, but when using multi-threads the performance is significantly increased since the number of threads is less than or equal to the number of cores, also when using more threads than available cores, it does not enhance the performance.

**Listing 1.3** - *.C file for the program.*

## Task 2: Multiple threads for handling I/O

**Listing 2.1:** *.c file for the program.*

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
```

```

#include "joystick.h"
#include "led_matrix.h"

//struct to pass the variables to thread
struct snek_properties
{
    uint16_t snek_color;
    int snek_len;
    int terminate_flag;
};

uint16_t snek_color = RGB565_CYAN;
int snek_len = 3;

void *handle_input(void *arg)
{
    struct snek_properties *snek = arg;

    struct js_event ev;
    open_joystick_device();
    while(1) {
        ev = read_joystick_input();
        if(ev.type == JOYSTICK_PRESS)
        {
            if(ev.direction == DIRECTION_NORTH)
            {
                if(snek->snek_len < 7)
                {
                    snek->snek_len ++;
                }
            }
            else if(ev.direction == DIRECTION_SOUTH)
            {
                if(snek->snek_len > 1)
                {
                    snek->snek_len --;
                }
            }
            else if(ev.direction == DIRECTION_EAST)
            {
                snek->snek_color = RGB565_YELLOW;
            }
            else if(ev.direction == DIRECTION_WEST)
            {
                snek->snek_color = RGB565_MAGENTA;
            }
            else if(ev.direction == DIRECTION_DOWN)
            {
                clear_leds();
                close_joystick_device();
                close_led_matrix();
                snek->terminate_flag = 1;
                return NULL;
            }
        }
    }
}

void sleep_ms(int ms) {
    usleep(1000 * ms);
}

int main() {
    int snek_pos = 0;
    open_led_matrix();

    pthread_t input_handle;
    struct snek_properties *snek = malloc(sizeof(struct snek_properties));
    snek->snek_len = snek_len;
    snek->snek_color = snek_color;
    snek->terminate_flag = 0;

    if(pthread_create(&input_handle, NULL, handle_input, snek)){

```

```

    fprintf(stderr, "Error creating new thread. \n");
    return -1;
}

/* Move the snek. */
while (snek->terminate_flag == 0) {
    clear_leds();
    snek_pos = (snek_pos + 1) % NUM_LEDS;
    for (int i = 0; i < snek->snek_len; i++) {
        int led_num = (snek_pos + i) % NUM_LEDS;
        set_led_num(led_num, snek->snek_color);
    }
    sleep_ms(100);
}

pthread_join(input_handle, NULL);

close_led_matrix();
}

```

## Task 3: Cache behavior

**Question 3.1.** - *What timings did you get from the program? Try to explain the reported timings with respect to the memory hierarchy and the program's behavior.*

**Answer to Question 3.1.** To access 1 to 32 KB of data, it takes around 1.5s because it uses L1 cache which is very fast. And to access data of size 64 Kb upto 512 KB, it takes 2 to seconds because it uses L2 cache. when it try to reach more bigger amount of data e.g more than L2 cache size then it takes longer time because it access the main memory directly which is slower than cache. For instance it takes 16 seconds to access 8 MB of data.

**Question 3.2.** - *Based on the timings and your knowledge of the Raspberry Pi's memory hierarchy structure, guess the sizes of the L1 data and L2 caches. Motivate your guesses carefully.*

**Answer to Question 3.2.** According to the observation of the output of the two functions with random and sequential access to array elements, the L1 cache size is 32KB and L2 cache size could be 256 KB or 512KB, and according to RPI website it confirms our estimations [1]

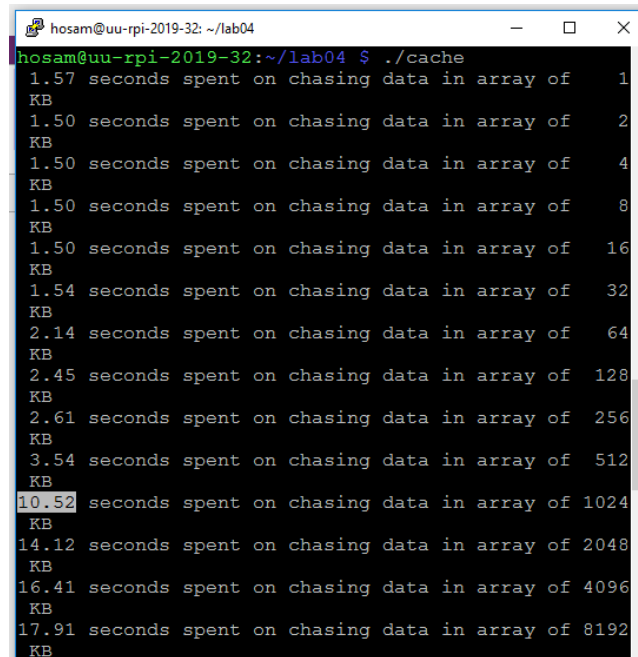


Figure 3: Random array access response time.

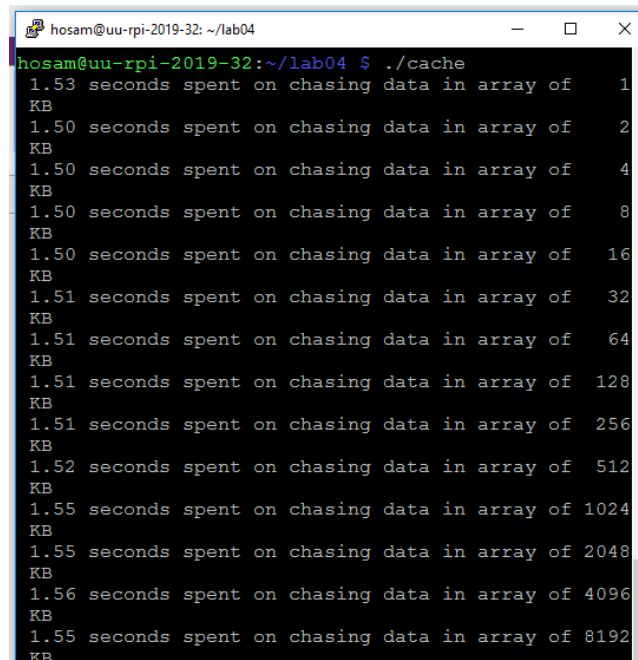


Figure 4: Sequential array access response time.

**Listing 3.3:** *.c file for the sequential array access program.*

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

#define MIN_ELEMENTS      (1 << 8)  /* 2^8 */
#define MAX_ELEMENTS      (1 << 21) /* 2^21 */
#define ITERATIONS_PER_TEST 100000000

int *make_ordered_array(size_t size)
{
    int *tmp_array = malloc(size * sizeof(int));
    int *array = malloc(size * sizeof(int));
    for (int i = 0; i < size; i++) {
        tmp_array[i] = i;
    }

    int j;
    for (int i = 0; i < size; i++) {
        j = tmp_array[i];
        array[j] = tmp_array[(i+1) % size];
    }

    free(tmp_array);
    return array;
}

/*
 * Chase data in an array by selecting the index of the next element
 * as the value of the current element.
 */
void loop_scan_array(int *array, size_t size, int iterations) {
    int index = 0;
    for (int i = 0; i < iterations; i++) {
        index = array[index];
    }
    return;
}

int main() {
    /* Time chasing of data in random matrices of different sizes. */
    for (size_t size = MIN_ELEMENTS; size <= MAX_ELEMENTS; size *= 2) {

        int *array = make_ordered_array(size);

        clock_t start = clock();
        loop_scan_array(array, size, ITERATIONS_PER_TEST);
        clock_t end = clock();

        free(array);

        printf("%5.2f seconds spent on chasing data in array of %4zu KB\n",
              (double)(end - start) / CLOCKS_PER_SEC,
              size * sizeof(int) / 1024);
    }

    return 0;
}
```

**Question 3.4.** - *What timings did you get from the edited program? Explain why these differ from the original program.*

**Answer to Question 3.4.** The time taken to access 1KB to 8MB of data was fixed in around 1.5s. Because the the behaviour of the program is changed

into sequential access to the memory, so the cache will succeed to implement spatial locality and prefetching mechanism to predict the required data before even it is requested by the CPU.

## References

- [1] Raspberry Pi 3 model B specificaion, *[Online] Available:.*  
<https://magpi.raspberrypi.org/articles/raspberry-pi-3-specs-benchmarks>  
[Accessed October 30, 2019].