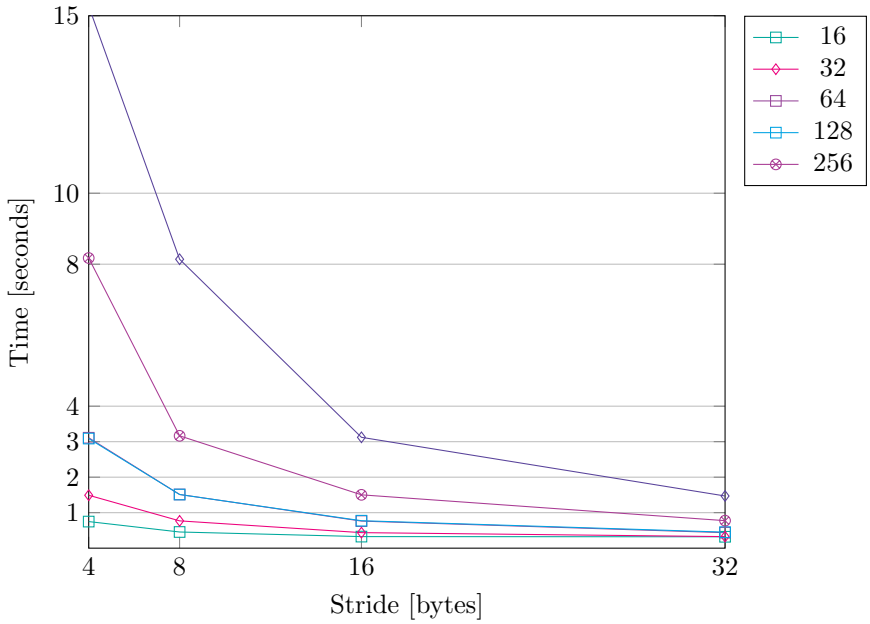
0.1 Exercise 2

0.1.1 C Code

Calculations

```c
//
//  bench.c
//
//
# include <stdio.h>
# include <string.h>
# include <stdlib.h>
# include <time.h>

#define MAX   100

int t; // temporary variable

void bench(int size, int stride, int A[]){
    for (int times = 0; times < MAX; times++){
        for (int i=0; i < size; i = i + stride){
            t = A[i];
        }
    }

}

int main (){
    int size;
    int stride;
    printf("Choose an array size: ");
    scanf("%d", &size);

    int arr[size];
    for (int i = 0; i < size; i += 1){
        arr[i] = i;
    }

    printf("Choose a stride: ");
    scanf("%d", &stride);

    printf("Timing ... ");

    // begin time
    clock_t begin = clock();

    bench(size, stride, arr);

    // end time
    clock_t end = clock();
    double time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
    printf("Seconds: %f s\n", time_spent);

    return 0;
}
```

Table 1. Time (s) vs. Stride



0.2 Time *vs.* Stride Initial Analysis

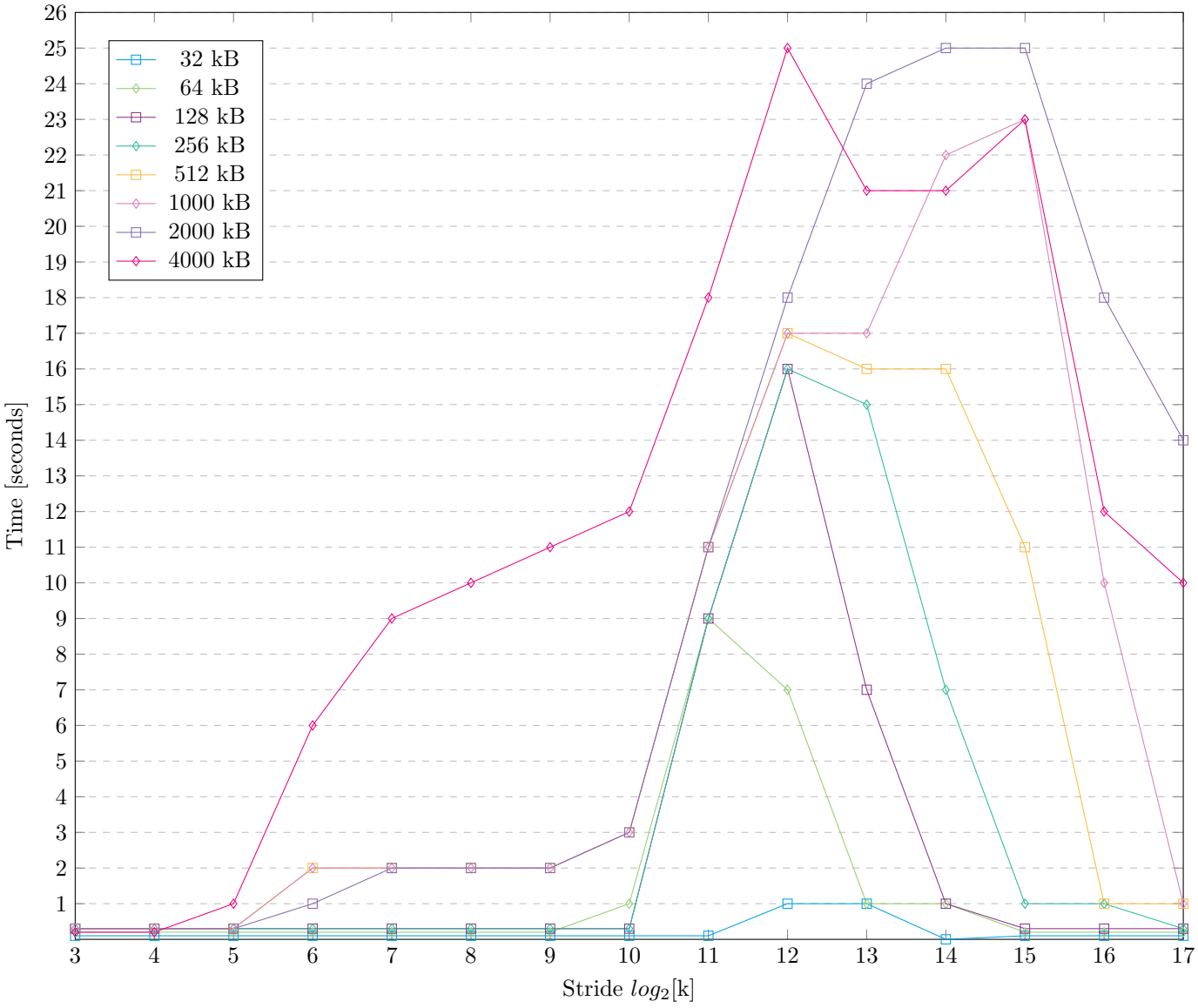
This graph shows the run time in seconds for the executed program given above, with the following array sizes: 16, 32, 64, 128, and 256. The graph with the smallest time value shows the values for array size 16, and so on. The first row of vertical square marks on each of the graphs is the main memory. Finally, the second row, where $x = stride$ at x = 8, is the L1 cache, which has a cache size of 32 bytes. We verify this with the following code in a Unix shell:

```
sysctl -a

// once we scroll down the list far enough
hw.l1icachesize: 32768
hw.l1dcachesize: 32768
```

where *l1i* stands for the Level 1 Instruction Cache and *l1d* is the Level 1 Data Cache.

0.3 Additional Results

Table 2. Time (s) vs. Stride



We can estimate the block size of the L1, L2 caches by noting that, for each array size, eventually the plot reaches a maximum point when the stride no longer increases the access time. This is the point at which all data brought from the next level in the memory hierarchy is accessed for a data point.

$$t \geq t_{max} = t \neg \propto \text{ stride}$$

$$Block = \frac{\text{Time}}{\text{Stride}} \tag{1}$$

Using equation (1), we find $k \overset{\text{estimate}}{\approx}$ 64 kB. (131,072 bytes / 2048 stride or 65,536 bytes / 1024 stride)

To estimate the cache sizes, we observe that plot significantly jumps for an array of size 64 kB, at which point for stride 10 it increases at a constant rate as stride increases. This suggests that the L1 Cache has 64 kB size. For arrays larger, we see 128 kB and 256 kB have the same slope as the 64 kB plot—indicating that after that level in the hierarchy, a new access time constant is reached. The 256 kb value seems to the second upper bound, as arrays larger do not share the same slope and have increased to a longer access time, as we see for 512 kB. Finally, comparing the plots for 2000 kB and 4000 kB, we estimate that the size of the L3 cache is somewhere on the order of 2 MB to 4 MB.

The code for generating this graph comes from Computer Architecture: A Quantitative Approach by David A. Patterson. Similarly to the C code in 0.1.1, it increases the stride and accesses the array. However, it loops for array sizes ranging from 1024 to 2048*2048 and has stride increases of 2^k.

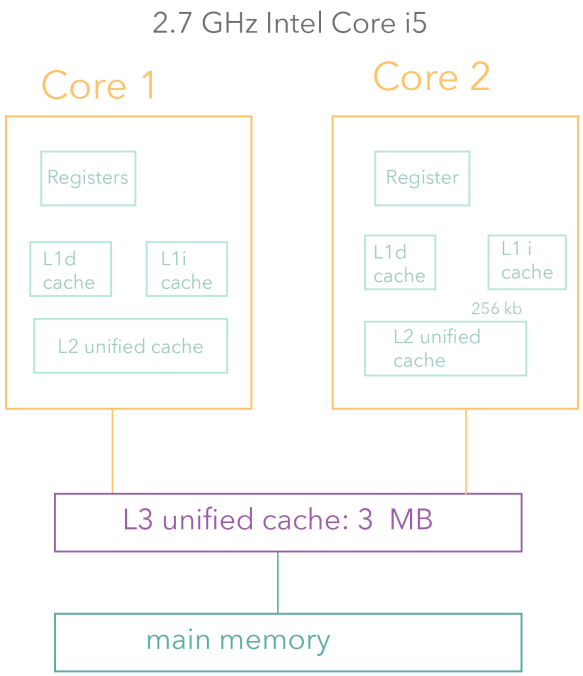


Figure 1: A figure showing the basic configuration of our computer's processor.