Assignment 3

Rene Anderson, Yiqi Liu, Francis Chau

In the previous assignment we initially set the RSA_NUMBER to a constant to improve the speed of all threads. We encountered a warp scheduling problem where the more blocks we added the slower the program performed. The following shows the increase in calculation time when the number of blocks is set to 655350 and the threads are 1024.

Fig 1 Warp causes increase in calculation time

This can also be verified by checking the occupancy calculator for the **"Active Thread Blocks per Multiprocessor"** field. It states that for the current device there are 4 ATBM. This means that at any time the maximum simultaneous threads that can be scheduled is (threads per warp) 32×4 (ATBM) = 128.

```
void BruteFactor::getPrimesCuda(){
    bool foundprime = false;
    unsigned __int64 i = 2;

    //set rsa number on device
    cudaError(cudaMemcpyToSymbol(d_RSA_NUMBER, &RSA_NUMBER, sizeof(__int64), 0, cudaMemcpyHostToDevice));

    //initialize temporary device variables
    unsigned __int64 *d_a, *d_b;

    cudaError(cudaMalloc((void**)&d_a, sizeof(__int64)));
    cudaError(cudaMalloc((void**)&d_b, sizeof(__int64)));

    //start timer
    start = chrono::high_resolution_clock::now();
    int blks = (RSA_NUMBER / NTPB) < 65535 ? (RSA_NUMBER / NTPB) : 65535;

    //loop until prime is found or max number is reached
    while (!foundprime && i < RSA_NUMBER){
        //starting number for each iteration
        unsigned __int64 h_a = i;
        //call cuda
        cudaGetPrimes << <blks, NTPB >> >(d_a, d_b, i);
        //get data from device
        cudaError(cudaMemcpy(&p1, d_a, sizeof(__int64), cudaMemcpyDeviceToHost));
        cudaError(cudaMemcpy(&p2, d_b, sizeof(__int64), cudaMemcpyDeviceToHost));

        if (p1 != 0 && p2 != 0)
            foundprime = true;

        i = i + blks * NTPB;
    }
    //free device variables
    cudaError(cudaFree(d_a));
    cudaError(cudaFree(d_b));

    //finish timer
    finish = chrono::high_resolution_clock::now();
    totalduration = chrono::duration_cast<chrono::milliseconds>(finish - start).count();
}
```

Fig Updated Kernel

Slight improvements were made to the way the program calculates the number of blocks. This increased performance by __ because it minimized the number of kernel calls needed to factorize the number. This only benefited numbers with 17 or more digits. Also the starting number constant from the previous assignment was removed as we later realised this gave no real benefit and actual caused some decreases in performance.

digits	A2 time	A3 time
5	0	1
7	0	21
8	0	123
10	10	151
11	10	150
12	10	150
13	10	162
14	10	151
15	20	150
16	190	299
17	610	600
18	1300	1350
19	3610	3477
20	6862	6452

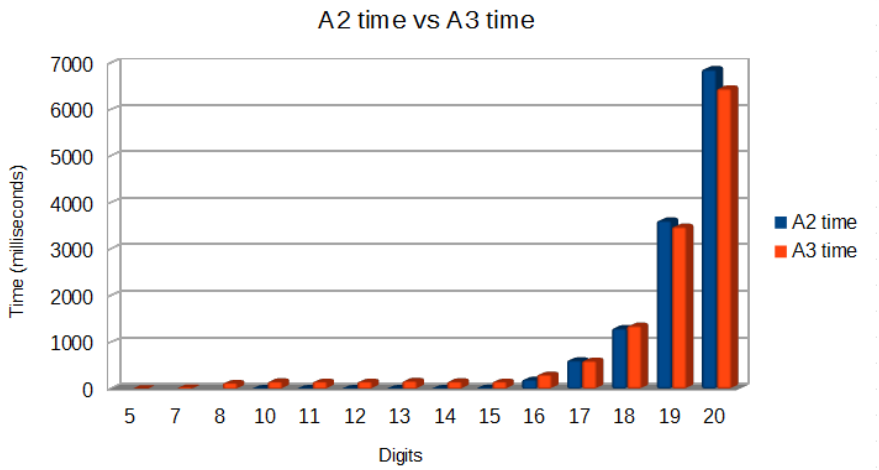

Fig 3 Slight improvements for lager digit numbers

For these results the block size was calculated using RSA_NUMBER / 1024 to get the blocks to be used. If the result was lager than 65535 the default value of 65535 would be used instead of the static block size of 1500 used in the previous assignment.