

# INFOF404: Operating Systems II

## Project 2: The Ulam Spiral

Delhayé Quentin

December 2, 2013

### 1 Introduction

The aim of the project is to generate an Ulam spiral using the MPI C++ library on Hydra cluster.

When running the Makefile, please make sure that the required modules are loaded:

### 2 Algorithm choice

The basic algorithm is the Eratosthenes sieve: beginning from two, we eliminate every multiple of the integer up to the limit. Then we do the same for the next integer that has not been eliminated. Every integer that has not been crossed are prime numbers.

The main flaw of this algorithm is that we will eliminate the same numbers several times. One of the improvements made to this algorithm is the wheel sieve [?]. This will go through several passes, generating the next prime numbers from the previously generated ones, avoiding the already discarded number.

Although the theoretical time complexity of Eratosthenes' algorithm ( $O(n \log \log n)$ ) is higher than wheel sieves' ( $O(n / \log \log n)$ ), [?] brought out that in practice, the distributed implementation of the wheel sieve (as described in [?]) was more than 200 times slower than the distributed Eratosthenes sieve on the same hardware, and for a limit of  $N = 1000000$ . The reason stated in the paper is that the Eratosthenes sieve computes much easier operation, when the Wheel sieve is more complex.

There exists another improvement of Eratosthenes sieve: Atkin Sieve [?]. Instead of eliminating all the multiples of previously computed prime number, it lays on a more complex mathematical reasoning. The main idea is to select some numbers with specific properties up to the square root of the limit,  $\sqrt{N}$ , then eliminate all the multiple of their square.

The wikipedia page devoted to this sieve [?] proposes a sequential pseudocode implementation of this algorithm. It served as a basis for this project implementation.

Beside the parallelization of the algorithm, the main modification is the adjustment of the different passes bounds. The original implementation proposed to go from  $y = 1$  to  $y = \sqrt{n}$ , but since we have  $4x^2 + y^2 \leq n$ , we can limit  $y$  to:  $y \leq \sqrt{n - 4x^2}$ . The same type of reasoning can be applied to the second pass with  $3x^2 + y^2 \leq n$  becomes  $y \leq \sqrt{n - 3x^2}$ . The last thing to check then is that the argument of the square root is positive or not; if it is, we set  $y$  to that bound, otherwise  $y = 1$ .

### 3 Implementation

First, the sequential algorithm needs to be split and distributed among the processes. We choose here a star layout: one process is the master and the others are its slaves.

The interval is the following:  $[5; N]$ , where  $N$  is the number of primes the client wants to generate, squared. This ensures to have both a square and the number of prime numbers required.

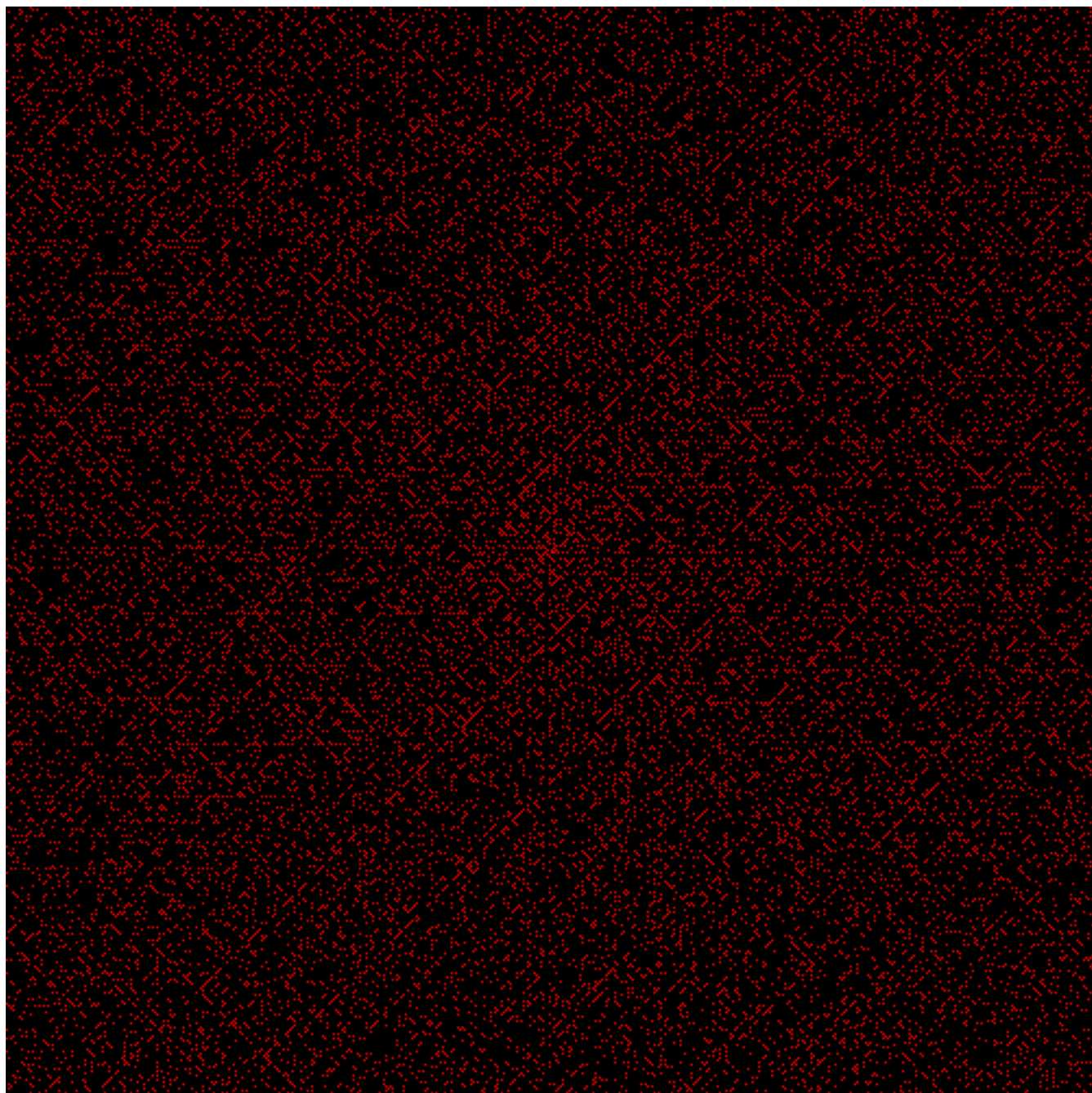


Figure 1: Ulam spiral for the 22261 first prime numbers.

The master first splits the study interval this way: he first removes the remainder of the division of the study interval by the amount of slaves and keep it as his local study interval. This way, the remaining interval is congruent to 0 modulo number of slave.

The master then sends the upper and lower interval bounds to each slave. Once it's done, everyone does his work in his own study interval and sends it back to the master. The master receives all the lists from the slaves and gather them in one big array. He then removes all the multiples of the squared marked prime numbers.

The last step is then taken by the master: generating the image from the list. Basically, he begins with the lower right corner of the spiral and paint it from the end up to the beginning.

The time calculation is done in two different ways:

- Computation time of each process using `clock()`;  
It is launched right after the MPI initializations and stopped at the very end of the program, just before the `MPI_Finalize`.
- Wall clock time using `gettimeofday()`;  
It is launched and stopped at the same time that the computation time.

## 4 Message passing

Almost all the messages are sent and received using blocking methods.

Hereunder is an sample of code representing the sending and receiving of an array:

```
int id, nbInstance;
MPI_Comm_rank(MPI_COMM_WORLD, &id);
MPI_Comm_size(MPI_COMM_WORLD, &nbInstance);
int BOUNDS_ARRAY = 0; //tag
MPI_Status status;

/**Master ends the bounds to the slaves*/
if(id == 0) {
for(int i=1; i<nbInstance; i++) {
int slaveLimit[2];
MPI_Send(slaveLimit, 2, MPI_INT, i, BOUNDS_ARRAY, MPI_COMM_WORLD);
}
}
/**Slaves receive the bounds from the master*/
else {
int slaveLimit[2];
MPI_Recv(slaveLimit, 2, MPI_INT, 0, BOUNDS_ARRAY, MPI_COMM_WORLD, &status);
}
```

There is only the last sending by the slaves, when they send their computation time, that is nonblocking, and this hoping that they will not wait the master to be ready to receive their message before initiating the sending.

## 5 Analysis

For some reason, it seems that when a lot of processes are used, some primes are not recognized, leading to an incomplete prime set.

	np = 2	np = 4	np = 8	np = 12
4900	0 0.010	0 0.010	0.02 0.057	0.02 0.037
10000	0.01 0.138	0.010.017	0.03 0.042	0.01 0.056
49729	0.1 0.117	0.110.113	0.12 0.238	0.07 0.237
99225	0.3 0.347	0.30.316	0.24 0.500	0.28 0.684
250000	1.19 1.202	1.181.190	0.88 1.198	1.03 2.06
499849	3.31 3.326	3.223.229	1.96 3.678	2.88 5.816
749956	5.9 5.930	5.895.901	5.65 7.698	5.31 9.920
1000000	9.089.0877	9.28 9.29	12.0113.893	6.8513.7742

Table 1: Vertical: limit N, horizontal: number of processes involed.

## 6 Improvements

**Study interval** The study interval may be narrowed to a lower upper bound. Indeed, the current implementation computes way more prime numbers than asked by the user.

**Messages** Instead of using `MPI_Send` and `MPI_Recv`, we may experiment the impact of using `MPI_Gather` instead.

**PNG generation** The file generation can not be done for very large set of prime numbers. The user should at least be informed that the image will not be generated if he asks too many prime numbers.

## References

- [1] D. J. Bernstein A. O. L. Atkin. Prime sieves using binary quadratic forms. *Mathematics of Computation*, 73(246):1023 – 1030, april 2003.
- [2] Gabriel Paillard. Le crible de la roue en distribué. *MAJECSTIC 2003 (MANifestation des JEunes Chercheurs en STIC)*, 2003.
- [3] Gabriel Paillard. A fully distributed prime numbers generation using the wheel sieve. *Parallel and Distributed Computing and networks*, 2005.
- [4] Wikipedia. Sieve of atkin — wikipedia, the free encyclopedia. [http://en.wikipedia.org/wiki/Sieve\\_of\\_Atkin](http://en.wikipedia.org/wiki/Sieve_of_Atkin). Last revision: 25 November 2013 22:29 UTC.