Quadratic Sieve implementation for factorization

backup, benchmark and network communication

Ayoub Ouarrak

26/04/2016

Universita' degli studi di Parma
Dipartimento di Matematica e Informatica

Introduction

RSA and factorization

RSA is a public key cryptosystem. Each user performs the following tasks:

- Choose p and q, large prime
- Calculate $N = pq$ and $\phi(N) = (p-1)(q-1)$
- Choose $e \in \mathbb{Z}^*_{\phi(N)}$ and $d \in \mathbb{Z}^*_{\phi(N)}$ such that $ed \equiv 1 \mod \phi(N)$
- (N, e) is the **public key**
- $(\phi(N), d)$ is the **private key**

To decrypt, is necessary to know e and $\phi(N)$, this means that we need to factorize N. The security of RSA relies on the difficulty of factoring N into it's prime factors. This problem is believed to be **NP**.

Quadratic Sieve

Integer factorization algorithm

- Invented by C.Pomerance in 1981
- Second fastest method known (after the general number field sieve)
- On April 1994, the factorization of RSA-129 was completed using QS.

Quadratic Sieve algorithm

Given the number n to factorize and an upper bound B.

Step 1 Create a parameter B and examine the numbers x^2 - n for B-smooth values, where x runs through the integers starting at $\lfloor n^{\frac{1}{2}} \rfloor$.

Step 2 Form the exponent vectors of B-smooth numbers, and use linear algebra to find subsequence x_1^2 - n, x_2^2 - n, ... , x_t^2 - n which has product a square, say A_2.

Step 3 From the exponent vectors of the numbers x_i^2 - n we can produce the prime factorization of A and find the least nonnegative residue of A mod n, say it a.

Quadratic Sieve algorithm

Step 4 Find the least nonnegative residue of the product $x_1 ... x_t \mod n$, say it b.

Step 5 We have $a^2 \equiv b^2 \mod n$. If $a \not\equiv \pm b \mod n$ then compute $gcd(a - b, n)$. Otherwise return to **Step 1**, find additional smooth values of x^2 - n, find a new linear dependency in **Step 2**, and repeat **Step 3-4**.

Quadratic Sieve parallel implementation

Step 1 Master initializes the variables and the sieving range in sub intervals.

Step 2 For each node, master sends the data needed to calculate the factor base and a sieving sub interval.

Step 3 If a node find a solution, it sends values back to the master.

Step 4 After gathering enough relations, master performs the Gaussian elimination and prints out the result and terminates nodes.

Quadratic Sieve parallel implementation

What happens when this process ends in the middle of computation?

Serialization

Backup

To prevent loss of data, we need a backup system.
A solution that can be used is **Serialization**.
Serialization is the process of translating data structures or objects state into a format that can be stored (for example, in a file or memory buffer, or transmitted across a network connection).

Programming language support

Several object-oriented programming languages directly support objects serialization.
Some of these are Ruby, Smalltalk, Python, PHP, Objective-C, Java, and the .NET family.

C++ has not a direct support, we need external libraries

- Boost
- Cereal
- Autoserial

Kairos

Kairos

C++ library for objects serialization

- Simple and clean Syntax
- Expandable library
- Usage of Archives
- Usage of Checkpoints to ensure serialization history

Example

```
class Fizz : public Serializable, public Serialization {
private:
    float v;
    int a;

public:
    Fizz() {
        .....
        registerObject(this, Serialization::TEXT);
    }

    void serialize(Archive& archive) {
        archive << v << a;
    }

    void deserialize(Archive& archive) {
        archive >> v >> a;
    }
};
```

Steps to serialize

- Extend *Serializable* and *Serialization*
- Register objects, choosing serialization format between *Serialization::TEXT* and *Serialization::BINARY*
- Implement *serialize* and *deserialize* methods

Extend Serializable and Serialization

```
class Object : public Serializable, public Serialization
```

Serializable is an abstract class that offers two pure virtual methods: *serialize* and *deserialize* "common" types.

Serialization is a class thats provide methods to register objects, create checkpoints and restore objects.

Objects registration

```
registerObject(this); // text serialization by default
registerObject(this, Serialization::TEXT);
registerObject(this, Serialization::BINARY);
```

Objects registration is necessary for the serialization process: if an object fail to register, a **SerializationException** is generated.

serialize and deserialize methods

```
void serialize(Archive& archive) {
  archive << data << ...;
}

void deserialize(Archive& archive) {
  archive >> data >> ...;
}
```

These two methods are virtual pure functions provided by the Serializable interface, so every class thats extend this interface needs to implement the serialization methods.

Create checkpoint

```
Fizz* fizz = new Fizz(2.3, 5);
try {
  Serialization::createCheckpoint(&fizz);
} catch(SerializationException* exp) {
  exp->what();
}
```

The method above get serialization format and calls the serialize method of Fizz, passing the correct archive.

Restore

```
try {
  auto objects = Serialization::restore<User>();
  object1 = objects.at("object1")->get();
} catch(SerializationException* exp) {
  exp->what();
}
```

The method above restores all objects of type *User* from the serialization index.

Kairos Serializations

Kairos supports 4 different serializations:

- Scalar
- Array
- Matrix
- Serializable Objects

Scalar

Serialization of scalar type is intuitive.

- Write values separated by space

Deserialization works in the same way.

- Reads values using $>>$ operator, in this way spaces are removed automatically.

For floating point types the serialization is different, to insure a portable serialization, double and float are encapsulated into a new type using IEE745 standard: **uint32** for float, and **uint64** for double

Array

Serialization:

- Write array size
- Iterate over the array and save values

Deserialization:

- Reads array size
- Iterate over the file and restore values

Matrix

Matrix used by the Quadratic Sieve algorithm are large and quite sparse. To prevent large serialization files and to improve deserialization time, we check if the percentage of zeros is higher than a certain threshold.

If the matrix is sparse, the serialization process save size of the matrix, non zero elements and their position.

The deserialization process reads size of matrix, creates a zero matrix and insert the elements from the file into the matrix.

Serializable Objects

```
class FactorBase : public Serializable, public Serialization
....

class QS : public Serializable, public Serialization {
  FactorBase factorBase;
   ....
   void serialize(Archive& archive) {
     archive <<  factorBase << ...;
   }
}
```

Serializable Objects

When we serialize QS, the serialize method of FactorBase is called first, in order to serialize all its data. After that the QS data are serialized.

Benchmark

Benchmark using cMark

We need a benchmark system to get resources usage information in order to improve the Quadratic Sieve performance. To achieve this goal a benchmark library called **cMark** has been developed. cMark work in two phases

- Collect and save data into a SQLite database.
- Read data from SQLite database and create charts.

Data collection

The data collection is made by a C++ program that offers a DeviceInfo interface and a distinct implementation for each supported platform (Windows, OSX, Linux).
When the program is executed, it enters a "infinity" loop, calling OS functions to get resource information each t minutes (configurable). All data is saved in a SQLite database.

Data representation

Data representation is made using web technologies

- The layout of charts is designed using HTML/CSS
- On the page load, a js script performs the following actions:
    - Localize the SQLite database
    - Reads all data and insert them in local vectors
    - Pass these vectors to the **Chart.js** library thats creates charts

cMark can be used as a normal web page or it can be packed with **electron** framework in order to build a native application.

Chart.js memory usage example

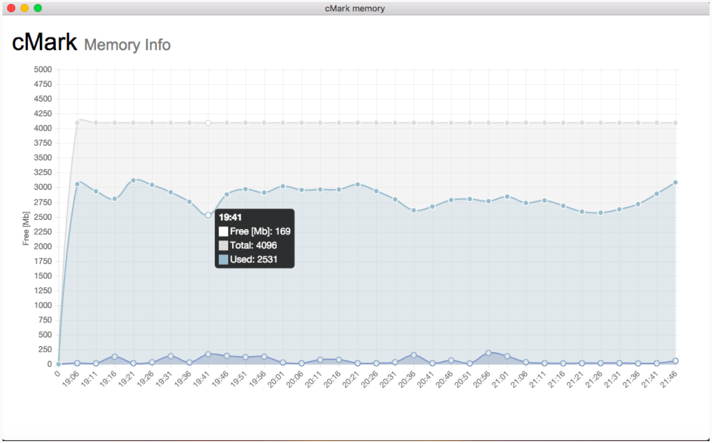

Figure 1: Memory usage example

Network communication

MPI

The parallelism in this version of the Quadratic Sieve is achieved using the open source OpenMPI library, an implementation of the Message Passing Interface.

- The first step is to initialize the connection between nodes and master using *MPI_init* that return a *rank* for each process of the environment.
- Master sends all is necessary to the nodes in order to construct the factor base.
- For each node, Master sends a sub interval for sieving process.
- Master remains listening for income solutions from the nodes when it have enough data it send to the nodes the command to stop.
- Last step is the Gaussian elimination made by the master.

Conclusion

Todo

Kairos and cMark are still in development.
Several potential improvements are possible:

- Serialization for other STL types.
- Check endianness in binary serializations.
- Change id generation during objects registration.
- Other types of Archives (e.g., JSON, XML).
- Monitoring objects state in order to make automatic checkpoints.
- Monitor usage of other resources, in particular network communication.
- Load SQLite database from the network.

Questions?