Implementazione crivello quadratico con SIQS e large prime variation

A cura di Cristian Milia

Caratteristiche del programma

- Scritto in linguaggio C
- Uso della libreria gmp per le operazioni su numeri “grandi”
- Uso della libreria mpfr per le operazioni su numeri “grandi” floating point
- Uso di thread dove possibile per cercare di minimizzare il tempo necessario per svolgere i task
- Codice parametrico:i parametri di esecuzione possono essere tranquillamente modificati per cercare di migliorare il programma
- Liberazione della memoria appena possibile
- Non richiede molta memoria ram per essere eseguito

Algoritmo SIQS

```
INPUT: N to factorize
While not found factorization  of N:
        FB=Calculate_factor_base()
        While not found enough square relations:
                a=calculate_coefficient_a()
                bi=calculate_coefficients_bi()
                sieving()
                relations=create_square_relation_and_semi_B_smooth_relation()
                union_square_relations()
                union_semi_B_smooth_relations()
                large_prime_variation()
        Remove_square_relation_useless()
        lin_syst=Create_linear_system()
        Reduce_echelon_form_linear_system(lin_syst)
        base=calculate_base_linear_system(lin_syst)
        while enough_solution available:
                solution=calculate solution_linear_system(base)
                calculate_c_d(solution)                 /* such that c^2==d^2 mod n */
                try_to_factorize_n(c,d,n)
                go to exit_if_one_factorization_is_found
OUTPUT:N=P*Q
```

Input del programma:N

- N viene preso da un file di input
- viene verificato essere composto con un certo numero di test di Miller Rabin
- viene memorizzato in una variabile mpz_t

Calcolo Factor Base

- Il calcolo della factor base viene fatto suddividendo l'intervallo [0,B] in t sottointervalli della stessa lunghezza
- Ogni thread ti ha il compito di calcolare la factor base sul sottointervallo
- Il main thread ha il compito di unire tutte le factor base così ottenute

Il calcolo della factor base su un sottointervallo si effettua trovando iterativamente tutti i primi nel sottointervallo con la funzione nextprime() e considerare solo quei primi tali che $n^{(p-1)/2}==1 \bmod p$

Il calcolo della factor base risulta essere molto veloce per la divisione del task in più thread.

Calcolo del coefficiente a

Per il SIQS si ha la necessità di generare polinomi del tipo aj^2+2bj+c dove a è il prodotto di s primi della factor base, b viene generato con la formula del Grey_code e c dipende da b e da N.

Per ogni a si generano 2^(s-1) coefficienti b diversi,di conseguenza 2^(s-1) polinomi diversi.

Per calcolare il coefficiente a occorre calcolare il valore ideale di a=rad(2*n)/M e scegliere randomicamente il prodotto di s primi della factor base che minimizzano la distanza dal valore di a ideale.

La scelta randomica dei fattori di a ci permette di generare a (molto probabilmente) diversi tra di loro,quindi anche polinomi diversi ogni volta che ne abbiamo bisogno.

Fase di sieving

La fase di sieving consiste nell'assegnare ad L thread un sottoinsieme dei polinomi generati a partire da un singolo coefficiente a.

Ogni thread per un singolo polinomio calcola l'array a(j)=aj^2+2bj+c j=-M,..M e scansiona tutta la lista della factor base memorizzando per ogni posizione dell'array la somma dei logaritmi dei primi che dividono a(j).

Per capire quali posizioni dell'array sono divisibili per pi della factor base basta trovare le radici del polinomio mod p,quindi risolvere aj^2+2bj+c==0 mod p

Creazione relazioni quadratiche e semi B-smooth

Dopo la fase di sieving si scansiona nuovamente l'array per cercare i numeri a(j) che hanno il valore della variabile sum_log_pi oltre una certa soglia. Tali numeri probabilmente sono B_smooth rispetto alla factor base,quindi vengono fattorizzati con l'algoritmo trial division per vedere se possono formare una relazione quadratica del tipo (aj+b)^2==num*a mod n.[dove num è esprimibile come prodotto di primi della factor base]

Nel caso gli a(j) siano B_smooth si forma una relazione quadratica,altrimenti si verifica che il resto P della divisione per tutti i primi della factor base sia compreso tra B e B^2 e si crea così una relazione semi_B_smooth del tipo (aj+b)^2==(num/P)*P*a mod n. [dove num/P è esprimibile come prodotto di primi della factor base].Le relazioni semi_B_smooth combinate tra loro sono utili per creare nuove relazioni quadratiche.

Union Square_relations e Union semi_B_smooth_relations

- Le relazioni quadratiche e semi_B_smooth trovate da ogni thread vengono unite tra loro dal main thread
- Vengono aggiornati i valori del numero di relazioni quadratiche e semi_B_smooth trovate
- Se le relazioni quadratiche superano un valore soglia viene chiamata la funzione large_prime_variation()

Large Prime Variation

- Le relazioni semi_B_smooth vengono ordinate per P crescente,dove P è il resto della divisione di a(j) per i primi della factor base.
- Se si trovano delle relazioni con lo stesso P è possibile creare nuove relazioni quadratiche come segue:
  - Siano k le relazioni con lo stesso P
  - Sia (1,i) l'insieme delle coppie con lo stesso P tenendo fissata la prima relazione,dove i=1..k
  - Per ogni coppia (1,i) è possibile creare una nuova relazione quadratica
    - prima relazione semi_B_smooth $q^2=P*FB1 \bmod n$
    - seconda relazione semi_B_smooth $r^2=P*FBi \bmod n$
    - relazione quadratica risultante $(q*r*P^{(-1)})^2==FB1*FBi \bmod n$ (dove FBi è il prodotto di alcuni primi della factor base con molteplicità)

Creazione sistema lineare

- Dalla lista delle relazioni quadratiche si costruisce il sistema lineare di dimensione cardinality_factor_base x num_B_smooth .
- Nella colonna iesima vi sono gli esponenti (modulo 2)dei primi della factor base della relazione quadratica iesima .
- Gli elementi della matrice sono perciò binari (0 oppure 1)
- Possiamo memorizzare la matrice come una matrice binaria risparmiando così memoria :
    - Ogni riga della matrice è un array di unsigned long int e ogni unsigned long int può memorizzare fino a 8*sizeof(unsigned long int) numeri binari

Reduce echelon form matrice binaria

- Per ridurre a scala la matrice occorre cercare i pivot e ridurre la sottomatrice in modo tale che tutti gli elementi sotto i pivot siano 0.
- Ma la matrice è binaria
  - i pivot sono per forza 1
  - La sottrazione di una riga per un altra moltiplicata per un certo fattore consiste in una sottrazione di righe binarie,che può essere fatta con lo XOR bit a bit,ciò permette di velocizzare la funzione di riduzione a scala di un fattore 64 rispetto ad una implementazione senza lo XOR binario

Calcolo della base del sistema lineare

- Una volta ridotta la matrice a scala occorre calcolare la dimensione della soluzione del sistema lineare e creare così un opportuno numero di vettori di base.
- Ogni vettore di base è una soluzione del sistema lineare
- I vettori di base sono linearmente indipendenti
- Una qualsiasi combinazione lineare dei vettori di base è ancora una soluzione del sistema lineare
- Siamo interessati a scansionare tutte le soluzioni fino a quando non troviamo quella che ci aiuta a fattorizzare N

Calcolo di C e D tali che $C^2 == D^2 \bmod n$

- Presa una singola soluzione del sistema lineare è possibile calcolare C e D tali che $C^2 == D^2 \bmod n$
- Ogni soluzione è composta da tanti 1 e tanti 0,
- Moltiplichiamo le relazioni quadratiche che corrispondono agli elementi con un 1 nel vettore soluzione considerato
- Moltiplicando tra loro queste relazioni quadratiche si ottengono proprio le variabili C e D

Try_to_factorize_N()

- Per ogni soluzione abbiamo trovato i valori di C e D tali che che $C^2 == D^2 \bmod n$
- Calcolando il gcd(C-D,N) e gcd(C+D,N) è possibile trovare dei fattori non banali di N!
- Se i gcd non portano a nessuna fattorizzazione di N allora si calcola una nuova soluzione,quindi 2 nuovi valori per C e D e a quel punto si riprova a fattorizzare N

Colli di bottiglia

L’unico collo di bottiglia riscontrato risiede negli step di algebra lineare,ossia dalla creazione del sistema lineare in poi.

Nonostante il sistema lineare sia stato memorizzato in modo binario e sia provvisto di una implementazione efficiente per la riduzione a scala,esso rimane il punto debole del programma:

- il tempo per la riduzione a scala potrebbe essere non trascurabile per N da 80 cifre in poi
- il tempo per il calcolo della base del sistema lineare potrebbe essere dominante per N da 80 cifre in poi(il tempo impiegato per calcolare la base del sistema lineare potrebbe eccedere il tempo impiegato per fare tutte le altre operazioni)

Possibili miglioramenti

- Implementazione più efficiente riduzione a scala per sistemi lineari binari(Parallelizzazione a livello di word e parallelizzazione a livello di flussi di programma,metodo di lanczos,ecc)
- Implementazione più efficiente calcolo della base del sistema lineare
- Velocizzare fase di sieving:per esempio cambiando algoritmo di sieving

FINE