Il Modello GC-60

Sezione 1: Visione e Filosofia

Il progetto **GC-60** non nasce come un semplice algoritmo di calcolo, ma come una riconsiderazione logica della ricerca dei numeri primi. Mentre i metodi tradizionali si concentrano sulla verifica diretta dei numeri all'interno di un intervallo, il modello GC-60 adotta un approccio **divisore-centrico**.

1.1 Il Concetto di Contenitore Passivo

Nel modello GC-60, la finestra di ricerca non è l'obiettivo dell'analisi, ma viene trattata come un **contenitore passivo**. Questa finestra non viene "scansionata", ma "intercettata" dalle proiezioni informative generate dai soli divisori noti e significativi.

- **Irrilevanza Informativa**: Tutti i divisori che non contribuiscono alla determinazione dei primi nella finestra specifica sono considerati informativamente irrilevanti e vengono ignorati
- **Proiezione e Traslazione**: L'informazione utile emerge esclusivamente tramite la **traslazione dei soli divisori noti** Quello che "sopravvive" a questo processo di scrematura strutturale rappresenta i candidati primi della finestra.

1.2 La Struttura MicroPrime

Questa filosofia è implementata nel progetto **MicroPrime** attraverso due programmi cardine

- **microprime_crea**: Ha il compito di costruire e organizzare l'infrastruttura informativa, generando archivi strutturati di divisori utili (offset del modulo 60). Non cerca numeri primi come fine ultimo, ma li fa emergere come risultato della struttura stessa

microprime_studio: Utilizza tali archivi per esplorare finestre arbitrarie, applicando la traslazione dei divisori e identificando i superstiti nella finestra

In questo contesto ho voluto sperimentare la capacità del modello GC-60 attraverso la costruzione di un programma capace di confrontarsi direttamente con i moderni sieve ottimizzati, con le implementazioni avanzate del wheel sieve e con i motori ad alte prestazioni derivati dal crivello di Eratostene, verificando se l'approccio divisore-centrico potesse reggere il confronto non solo sul piano teorico, ma anche su quello computazionale e strutturale.

Il risultato di questa sperimentazione è lo **Sieve Wheel M60**, un motore di eliminazione strutturale progettato per operare su miliardi di numeri attraverso una compressione logica dello spazio di ricerca e una propagazione modulare delle informazioni divisorie.

Sieve Wheel M60

Sezione 2: Architettura Tecnica – Modulo 60 e Bitmasking

L'efficienza dello Sprinter Engine risiede nella sua capacità di mappare la realtà numerica in una struttura dati estremamente densa, ottimizzata per le operazioni logiche a basso livello.

2.1 Compressione Strutturale e Stato Traslato (16-bit Mapping)

Il modello GC-60 non introduce il modulo 60 come novità matematica, la wheel basata su 2×3×5 è nota da decenni, ma ne ridefinisce l'ancoraggio strutturale.

Lo spazio di ricerca viene organizzato in blocchi da 60 unità, ciascuno compresso in una parola a 16 bit (uint16_t). Tuttavia, la rappresentazione numerica non è centrata sull'origine naturale del modulo, bensì su una **traslazione costante di 10 unità**:

$n = L \cdot 60 + 10 + \text{residuo}$

Questa traslazione ha una funzione strutturale fondamentale:

- stabilizza la corrispondenza tra indice di sottolista e residuo;
- rende il ciclo dei multipli di ogni divisore perfettamente periodico;
- elimina la necessità di correzioni di stato tra segmenti;
- garantisce che l'intero sistema operi come una struttura ripetibile e chiusa.

I 16 bit rappresentano i residui coprimi con 60
{1,3,7,9,13,19,21,27,31,33,37,39,43,49,51,57}.

2.2 Logica di Eliminazione tramite Bitmasking

A differenza dei crivelli classici che calcolano la posizione di ogni multiplo, lo Sprinter Engine utilizza la **propagazione strutturale di maschere modulari**.

- **Pre-calcolo dei Pattern**: Per ogni divisore p, viene generato un pattern ciclico di maschere che riflette come i multipli di p "colpiscono" i 16 residui del modulo 60.
- **Operazioni Bitwise Atomiche**: L'eliminazione avviene tramite l'istruzione logica `AND` tra la cella di memoria e il NOT della maschera (ptr[L] &= ~mask). Questa operazione viene eseguita direttamente nei registri della CPU, permettendo di processare l'informazione alla velocità del clock del processore.

2.3 Ottimizzazione dei Salti (The Sprinter Jump)

Per massimizzare il throughput, l'algoritmo implementa una tecnica di salto calcolato:

- **Indici di Sottolista**: Il sistema calcola l'indice della sottolista (`L`) e il bit corrispondente per il punto di partenza p2.
- **Avanzamento Strutturale**: Invece di incrementare il valore numerico, il sistema incrementa direttamente l'indice della sottolista di un fattore p. Questo permette di saltare

intere porzioni della finestra di ricerca, intercettando solo i punti dove la proiezione del divisore è realmente significativa

Intuizione Strutturale

Questa architettura riconfigura la ricerca dei numeri primi come un processo di eliminazione strutturale basato sulla propagazione modulare dell'informazione divisoria. L'operazione non si fonda su verifiche aritmetiche ripetitive, ma su trasformazioni logiche applicate a rappresentazioni compresse dello spazio numerico.

Il **Modulo 60 in assetto switch +10**, ossia la wheel classica 2×3×5 ancorata tramite una traslazione strutturale costante di 10 unità, non introduce un nuovo sistema modulare, ma individua una posizione strutturalmente stabile nel contesto dei numeri interi. Questo ancoraggio rende perfettamente periodici i pattern dei divisori, elimina le correzioni di stato tra segmenti e consente una propagazione modulare coerente e ripetibile.

Combinato con il bitmasking strutturale, il sistema concentra il calcolo esclusivamente sui residui coprimi e sfrutta istruzioni logiche a basso costo, riducendo significativamente le operazioni ridondanti rispetto a un crivello tradizionale non ottimizzato, sia in termini di carico computazionale sia di pressione sulla memoria..

Sezione 3: Analisi Tecnica delle Prestazioni

L'eccezionale velocità del motore **GC-60 Sprinter** non è frutto di una semplice ottimizzazione del codice, ma deriva da una perfetta sinergia tra l'architettura logica del metodo e la struttura fisica dei processori moderni.

3.1 Benchmark e Throughput

I test eseguiti su una singola istanza (Single-Core) hanno mostrato i seguenti dati di elaborazione:

- **Finestra 1,4 Mld**: 0,38 secondi (~3,68 Mld numeri/sec).
- **Finestra 5,0 Mld**: 1,88 secondi (~2,65 Mld numeri/sec).
- **Finestra 10,0 Mld**: 4,39 secondi (~2,27 Mld numeri/sec).

Il **throughput** medio si attesta stabilmente sopra i 2 miliardi di numeri processati al secondo, un valore che pone il **Sieve Wheel M60** ai vertici dell'efficienza per algoritmi di questo tipo.

3.2 Il Fattore Cache L1 (Cache-Friendly Design)

Il collo di bottiglia principale nei crivelli tradizionali è la latenza della memoria RAM. Lo Sprinter Engine supera questo limite attraverso la **segmentazione ottimizzata**:

- **Dimensione del Blocco**: Il programma lavora su segmenti di **32 KB** (`SEG_SIZE`), dimensione scelta per risiedere interamente nella **Cache L1**, la memoria più veloce del processore.
- **Riduzione della Latenza**: Operando in Cache L1, il processore accede ai dati con una latenza di circa 1 nanosecondo, rispetto ai 60-100 nanosecondi necessari per la RAM. Questo permette alla logica GC-60 di correre alla massima frequenza di clock della CPU.

3.3 Efficienza del Bitmasking Modulo 60

La scelta del **Modulo 60** con compressione a **16 bit** permette una densità informativa ottimale:

- **Eliminazione delle Ridondanze**: Escludendo a priori i multipli di 2, 3 e 5, l'algoritmo non spreca cicli di clock su numeri che non possono essere primi.
- **Operazioni Bitwise**: La cancellazione dei composti avviene tramite operazioni `AND` logiche su maschere pre-calcolate. Questo tipo di istruzione è tra le più veloci eseguibili da un'unità logica aritmetica (ALU).

3.4 Scaling e Linearità

I dati mostrano che il tempo di esecuzione scala in modo quasi perfettamente lineare al raddoppiare della finestra (da 5 a 10 miliardi). Questa linearità dimostra che il metodo **GC-60** è immune al degrado delle prestazioni tipico degli algoritmi che soffrono di saturazione della memoria..

Sezione 4: Densità Strutturale e Sostenibilità in Memoria

Un aspetto fondamentale del modello GC-60 non riguarda esclusivamente la velocità di elaborazione, ma la densità con cui lo spazio numerico viene rappresentato in memoria.

Nel sistema:

- 60 numeri naturali sono compressi in 16 bit
- un intero blocco strutturale occupa 2 byte
- la densità risultante è:

$$\frac{16}{60} \approx 0{,}266\ bit\ per\ numero$$

Questa riduzione è resa possibile dalla rappresentazione esclusiva dei residui coprimi con 60, escludendo strutturalmente i multipli di 2, 3 e 5.

A confronto:

- un bit-sieve classico utilizza 1 bit per numero
- un sieve booleano utilizza 1 byte per numero

Ne consegue una riduzione della memoria pari a:

- ≈ 3,75× rispetto a un bit-sieve puro
- ≈ 30× rispetto a una rappresentazione booleana

Questa densità strutturale consente di mantenere in memoria intervalli numerici molto ampi senza ricorrere a segmentazione esterna o supporti secondari.

Verifica empirica su scala 10^{11}

Applicando il modello a un intervallo fino a 100 miliardi:

- i blocchi strutturali generati sono ≈ 1,67 miliardi
- l'occupazione complessiva di memoria è ~3,3 GB

L'elaborazione ha prodotto:

- Tempo totale: 111,88 secondi
- Candidati residui: 4.118.054.811
- Valore teorico $\pi(10^{11}) = 4.118.054.813$

Lo scarto di 2 unità è compatibile con gli estremi della finestra considerata e conferma la correttezza della proiezione strutturale.

Il dato rilevante non è solamente il risultato numerico, ma il fatto che l'intero intervallo sia stato gestito interamente in RAM, senza degrado dovuto a operazioni di I/O esterno.

Sezione 5: Ambiente Hardware di Test

Tutti i benchmark sopra riportati sono stati ottenuti utilizzando un hardware standard, a dimostrazione della validità logica del metodo a prescindere da infrastrutture di calcolo professionali..

- **CPU**: **AMD Ryzen 7 4800U** with Radeon Graphics.
- **Architettura**: 8 Cores fisici, 16 Threads.
- **Frequenza Base**: 1.80 GHz.
- **Memoria RAM**: 32 GB.
- **Configurazione Software**: Compilatore C++ ottimizzato per istruzioni **AVX2**.

L'utilizzo di un processore mobile (serie U) sottolinea come il modello **GC-60** non richieda potenze dissipate elevate per ottenere prestazioni di alto livello, grazie alla gestione intelligente della **Cache L1** e della linearità del salto strutturale.

Considerazione Finale su Sieve Wheel M60

Il fatto che un intervallo fino a 100 miliardi richieda poco più di 3 GB di RAM dimostra l'elevata densità strutturale del modello GC-60.

Questa caratteristica apre concretamente la possibilità di esplorare finestre dell'ordine di 1.000 miliardi (10^{12}) su una macchina dotata di 64 GB di RAM, mantenendo l'elaborazione interamente in memoria principale.

In questo senso, il **Sieve Wheel M60** non si distingue soltanto per il throughput elevato, ma per la sua capacità di coniugare velocità e sostenibilità della memoria su scale numeriche dell'ordine del trilione.