VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS

Ilgiausios Collatz’o iteracijos paieška duotajame skaičių intervale

Lygiagretieji skaičiavimai

Atliko: Tomas Kozakas

Vertino: Prof., Dr. Rimantas Vaicekauskas

Vilnius
2024

Turinys

1. Problemos formulavimas

Problema: Ilgiausios Collatz'o iteracijos paieška duotajame skaičių intervale.

Užduoties tikslas yra suprojektuoti pateiktajai problemai efektyvų lygiagretaus vykdymo algoritmą ir jį realizuoti.

Kiekvienam $i = 1, 2, 3, \ldots$

- $A(0)$ - pradinis iteracijos elementas, natūralusis.
- $A(i+1) = 3 \cdot A(i) + 1$, kai $A(i)$ - nelyginis
- $A(i+1) = \frac{A(i)}{2}$, kai $A(i)$ - lyginis.

Pagal Collatz'o hipotezę, kiekvienam natūraliajam n, po baigtinio skaičiaus žingsnių galima gauti 1.

Pavyzdžiui: $7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1$.

2. Lygiagretusis algoritmas

Kadangi kiekviena Collatz sekos reikšmė priklauso tik nuo savo pirmtako, sekos skaičiavimas gali būti lengvai suskirstytas. Pavyzdžiui, jeigu turime skaičių intervalą nuo 1 iki 1 milijono, šį intervalą galima padalinti į mažesnius sub-intervalus, kiekvieną iš jų priskiriant atskirai gijai (žr. 2.1 lentelė).

2.1 lentelė. Užduočių rezultatai su 16 gijų

Intervlas	Maks. Sekos Ilgis	Skaičius	Užtruko Laikas (ms)
1 iki 62500	340	52527	1457
62501 iki 125000	354	106239	1508
125001 iki 187500	383	156159	1579
187501 iki 250000	443	230631	1571
250001 iki 312500	407	263103	1663
312501 iki 375000	441	345947	1671
375001 iki 437500	449	410011	1717
437501 iki 500000	444	461262	1707
500001 iki 562500	470	511935	1790
562501 iki 625000	447	615017	1782
625001 iki 687500	509	626331	1668
687501 iki 750000	504	704623	1773
750001 iki 812500	468	767903	1709
812501 iki 875000	525	837799	1773
875001 iki 937500	476	910107	1771
937501 iki 1000000	507	939497	1786

Pastaba: Bendras skaičiavimo laikas su 16 gijų buvo 2126 ms su įjungtų DEBUG mode.

2.2 lentelė. Užduočių rezultatai su 16 gijų

Intervlas	Maks. Sekos Ilgis	Užtruko Laikas (ms)
1 iki 62500	340	1457
62501 iki 125000	354	1508
125001 iki 187500	383	1579
187501 iki 250000	443	1571
250001 iki 312500	407	1663
312501 iki 375000	441	1671
375001 iki 437500	449	1717
437501 iki 500000	444	1707
500001 iki 562500	470	1790
562501 iki 625000	447	1782
625001 iki 687500	509	1668
687501 iki 750000	504	1773
750001 iki 812500	468	1709
812501 iki 875000	525	1773
875001 iki 937500	476	1771
937501 iki 1000000	507	1786
Bendras laikas		**1811**

3. Vykdymo aplinka

Šiam projektui buvo naudojama mif superkompiuteris su šiais techniniais parametrais:

- Pavadinimas: beta
- Mazgai: 42/56
- Procesoriai: 2 x Intel Xeon X5650 2.66GHz = 12 cores
- RAM: 24GB
- HDD: 160GB
- Tinklas: 1Gbit/s,4xDDR(20Gbit/s) infiniband
- Ypatybės: beta,ib,ib20,ups

Testavimo programa buvo parašyta ant Java, pasinaudojant Java gijomis, kurios yra sukurtos ir valdomos per ‘ExecutorService‘ klasę, leidžiančią valdyti gijų grupę ir jų vykdymą. Programos kodą galima rasti šio dokumento pabaigoje (žr. 1, 2, 3 kodą).

Komandinis failas viduj MIF klasterio buvo sukonfigūruotas atitinkamai:

```
#!/bin/sh
javac Main.java CollatzTask.java CollatzCalculator.java
java Main 1 1000000 debug 16

sbatch -N1 -c8 -o run.out run.sh
```

- -c8 kiek branduolių rezervuoti
- -o išvisties failas

4. Eksperimentinio tyrimo rezultatai

Iš žemiau pateiktų lentelių galima matyti, kad skaičiavimo laikas, kuris reikalingas maksimalaus sekos ilgio radimui naudojant lygiagretųjį programavimą su skirtingu skaičiumi gijų, gali skirtis priklausomai nuo intervalo, su kuriuo dirba kiekviena gija, ir pačių apdorojamų skaičių dydžio. Pavyzdžiui, didesnių intervalų apdorojimas gali užtrukti ilgiau nei mažesnių intervalų, kaip matome iš lentelių 4.1 ir 4.2. Intervalas nuo 1 iki 1000000 užtrunka ilgiau nei intervalas nuo 1 iki 500000 lygiagrečiai su 500001 iki 1000000. Taigi, pirmas optimizavimas su 2 gijomis duoda rezultatą, kuris pagerina skaičiavimo laiką. Skaičiavimo laikas sumažėja, nes skaičiavimai atliekami lygiagrečiai, todėl intervalų apdorojimo trukmė sumažėja.

Svarbu pažymėti, kad yra riba, kada didesnis gijų skaičius nebepadeda pagreitinti skaičiavimų ar net gali juos sulėtinti.

Kaip matome iš lentelių 4.3 ir 4.4, padidinus gijų skaičių nuo 4 iki 8, skaičiavimo laikas sumažėja. Tai rodo, kad daugiau gijų leidžia efektyviau panaudoti resursus ir pagreitinti skaičiavimus lygiagrečiai (žr.4.1, 4.2) **Pastaba:** Nurodytas laikas yra su įjungtų DEBUG mode.

4.1 lentelė. Programos testavimo rezultatai su nuoseklia programa

Intervlas	Maks. Sekos Ilgis	Užtruko Laikas (ms)
1 iki 1000000	525	1700
Bendras laikas		**1720**

4.2 lentelė. Programos testavimo rezultatai su 2 gijomis

Intervlas	Maks. Sekos Ilgis	Užtruko Laikas (ms)
1 iki 500000	449	1310
500001 iki 1000000	525	1610
Bendras laikas		**1613**

4.3 lentelė. Programos testavimo rezultatai su 4 gijomis

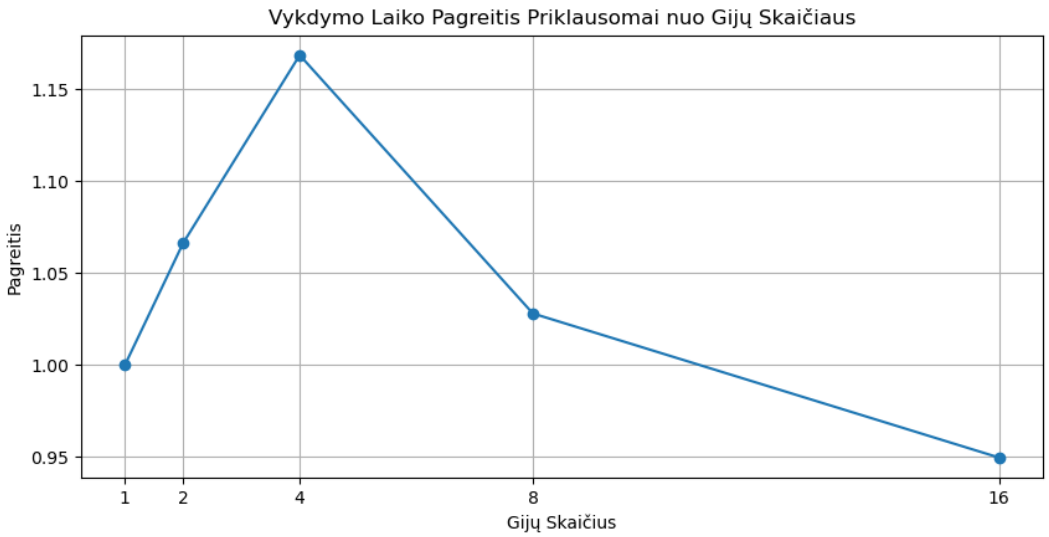
Intervlas	Maks. Sekos Ilgis	Užtruko Laikas (ms)
1 iki 250000	443	1261
250001 iki 500000	449	1427
500001 iki 750000	509	1445
750001 iki 1000000	525	1468
Bendras laikas		**1472**

4.4 lentelė. Programos testavimo rezultatai su 8 gijomis

Intervlas	Maks. Sekos Ilgis	Užtruko Laikas (ms)
1 iki 125000	354	1451
125001 iki 250000	443	1495
250001 iki 375000	441	1495
375001 iki 500000	449	1538
500001 iki 625000	470	1630
625001 iki 750000	509	1655
750001 iki 875000	525	1662
875001 iki 1000000	507	1667
Bendras laikas		**1673**

4.5 lentelė. Užduočių rezultatai su 16 gijų

Intervlas	Maks. Sekos Ilgis	Užtruko Laikas (ms)
1 iki 62500	340	1457
62501 iki 125000	354	1508
125001 iki 187500	383	1579
187501 iki 250000	443	1571
250001 iki 312500	407	1663
312501 iki 375000	441	1671
375001 iki 437500	449	1717
437501 iki 500000	444	1707
500001 iki 562500	470	1790
562501 iki 625000	447	1782
625001 iki 687500	509	1668
687501 iki 750000	504	1773
750001 iki 812500	468	1709
812501 iki 875000	525	1773
875001 iki 937500	476	1771
937501 iki 1000000	507	1786
Bendras laikas		**1811**



4.2 pav. Programos veikimo pagreitis priklausomybė nuo gijų skaičiaus

5. Išvados apie pateiktojo algoritmo spartinimą bei plečiamumą

Iš analizuotų rezultatų, galima matyti, kad skaičiavimo laikas gali skirtis priklausomai nuo intervalo, su kuriuo dirba kiekviena gija ir nuo gijų skaičiaus. Norint maksimaliai išnaudoti lygiagretaus programavimo privalumus, svarbu tinkamai pasiskirstyti skaičiavimus tarp gijų ir atsižvelgti į intervalų dydį.

Padidinus gijų skaičių, skaičiavimo laikas sumažėja. Tai rodo, kad didesnis gijų skaičius leidžia efektyviau panaudoti resursus ir pagreitinti skaičiavimus. Tačiau yra riba, kada didesnis gijų skaičius nebedaro teigiamo poveikio ar net gali sulėtinti skaičiavimus dėl perteklinio sinchronizavimo ir konkurencijos už resursus.

6. Programos kodas

```
    public class Main {
    public static void main(String[] args) {
        if (args.length < 4) {
            System.out.println("Usage: java CollatzCalculator <startRange> <endRange> <mode> <
    threadsCount>");
            System.out.println("<mode> should be 'debug' or 'fast'");
            return;
        }

        long startRange = Long.parseLong(args[0]);
        long endRange = Long.parseLong(args[1]);
        String mode = args[2];
        int threadsCount = Integer.parseInt(args[3]);

        CollatzCalculator calculator = new CollatzCalculator(startRange, endRange, mode,
    threadsCount);
        calculator.run();
    }
}
```

Kodas 1: Main Java class

```java
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

public class CollatzTask implements Runnable {
    private final long startRange;
    private final long endRange;
    private final String mode;
    private static final Map<Long, Integer> cache = new ConcurrentHashMap<>();

    public CollatzTask(long startRange, long endRange, String mode) {
        this.startRange = startRange;
        this.endRange = endRange;
        this.mode = mode;
    }

    @Override
    public void run() {
        long startTime = System.currentTimeMillis();
        if (mode.equals("debug")) {
            System.out.println("Starting task in debug mode for range: " + startRange + " to " + endRange);
            try {
                Thread.sleep(1000); // Sleep for 1 second
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }

        long maxLengthNumber = 0;
        int maxLength = 0;

        for (long i = startRange; i <= endRange; i++) {
            int length = calculateCollatzLength(i);
            if (length > maxLength) {
                maxLength = length;
                maxLengthNumber = i;
            }
        }

        long endTime = System.currentTimeMillis(); // End timing
        long taskDuration = endTime - startTime; // Calculate duration

        if ("debug".equals(mode)) {
            System.out.println("Task for range: " + startRange + " to " + endRange + " finished. Max length: " +
                    maxLength + " for number: " + maxLengthNumber + ". Time taken: " + taskDuration + " ms");
        }
    }

    private int calculateCollatzLength(long n) {
        if (n == 1) return 1;
        if (cache.containsKey(n)) return cache.get(n);

        long next = n % 2 == 0 ? n / 2 : 3 * n + 1;
        int length = 1 + calculateCollatzLength(next);

        cache.put(n, length);
        return length;
    }
}
```

Kodas 2: CollatzTask Java class

```java
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class CollatzCalculator implements Runnable {
    private final long startRange;
    private final long endRange;
    private final String mode;
    private final int threadsCount;

    public CollatzCalculator(long startRange, long endRange, String mode, int threadsCount) {
        this.startRange = startRange;
        this.endRange = endRange;
        this.mode = mode;
        this.threadsCount = threadsCount;
    }

    @Override
    public void run() {
        System.out.println("Operating in " + mode.toUpperCase() + " mode.");
        System.out.println("Testing with " + threadsCount + " threads.");

        long startTime = System.nanoTime();

        runCalculations(startRange, endRange, threadsCount, mode);

        long endTime = System.nanoTime();
        long duration = TimeUnit.NANOSECONDS.toMillis(endTime - startTime);

        System.out.println("Calculation with " + threadsCount + " threads took: " + duration + "
    ms\n");
    }

    public static void runCalculations(long startRange, long endRange, int threadsCount, String
    mode) {
        ExecutorService executor = Executors.newFixedThreadPool(threadsCount);

        long intervalLength = (endRange - startRange + 1) / threadsCount;
        long currentStart = startRange;

        for (int i = 0; i < threadsCount; i++) {
            long currentEnd = (i == threadsCount - 1) ? endRange : currentStart + intervalLength
    - 1;
            executor.execute(new CollatzTask(currentStart, currentEnd, mode));
            currentStart = currentEnd + 1;
        }

        executor.shutdown();
        try {
            executor.awaitTermination(Long.MAX_VALUE, TimeUnit.NANOSECONDS);
        } catch (InterruptedException e) {
            System.out.println("Execution was interrupted.");
        }
    }
}
```

Kodas 3: CollatzCalculator Java class

7. Programos išvestis

```
Operating in DEBUG mode.
Testing with 1 threads.
Starting task in debug mode for range: 1 to 1000000
Task for range: 1 to 1000000 finished. Max length: 525 for number: 837799. Time taken: 1700 ms
Calculation with 1 threads took: 1720 ms
```

Kodas 4: 1 gija

```
Operating in DEBUG mode.
Testing with 2 threads.
Starting task in debug mode for range: 1 to 500000
Starting task in debug mode for range: 500001 to 1000000
Task for range: 1 to 500000 finished. Max length: 449 for number: 410011. Time taken: 1310 ms
Task for range: 500001 to 1000000 finished. Max length: 525 for number: 837799. Time taken: 1610
    ms
Calculation with 2 threads took: 1613 ms
```

Kodas 5: 2 gijų

```
Operating in DEBUG mode.
Testing with 4 threads.
Starting task in debug mode for range: 1 to 250000
Starting task in debug mode for range: 500001 to 750000
Starting task in debug mode for range: 750001 to 1000000
Starting task in debug mode for range: 250001 to 500000
Task for range: 1 to 250000 finished. Max length: 443 for number: 230631. Time taken: 1261 ms
Task for range: 250001 to 500000 finished. Max length: 449 for number: 410011. Time taken: 1427
    ms
Task for range: 500001 to 750000 finished. Max length: 509 for number: 626331. Time taken: 1445
    ms
Task for range: 750001 to 1000000 finished. Max length: 525 for number: 837799. Time taken: 1468
    ms
Calculation with 4 threads took: 1472 ms
```

Kodas 6: 4 gijų

```
Operating in DEBUG mode.
Testing with 8 threads.
Starting task in debug mode for range: 1 to 125000
Starting task in debug mode for range: 125001 to 250000
Starting task in debug mode for range: 500001 to 625000
Starting task in debug mode for range: 750001 to 875000
Starting task in debug mode for range: 625001 to 750000
Starting task in debug mode for range: 375001 to 500000
Starting task in debug mode for range: 875001 to 1000000
Starting task in debug mode for range: 250001 to 375000
Task for range: 1 to 125000 finished. Max length: 354 for number: 106239. Time taken: 1451 ms
Task for range: 125001 to 250000 finished. Max length: 443 for number: 230631. Time taken: 1495
    ms
Task for range: 250001 to 375000 finished. Max length: 441 for number: 345947. Time taken: 1495
    ms
Task for range: 375001 to 500000 finished. Max length: 449 for number: 410011. Time taken: 1538
    ms
Task for range: 500001 to 625000 finished. Max length: 470 for number: 511935. Time taken: 1630
    ms
Task for range: 625001 to 750000 finished. Max length: 509 for number: 626331. Time taken: 1655
    ms
Task for range: 750001 to 875000 finished. Max length: 525 for number: 837799. Time taken: 1662
    ms
Task for range: 875001 to 1000000 finished. Max length: 507 for number: 939497. Time taken: 1667
    ms
Calculation with 8 threads took: 1673 ms
```

Kodas 7: 8 gijų

```
Operating in DEBUG mode.
Testing with 16 threads.
Starting task in debug mode for range: 62501 to 125000
Starting task in debug mode for range: 250001 to 312500
Starting task in debug mode for range: 312501 to 375000
Starting task in debug mode for range: 937501 to 1000000
Starting task in debug mode for range: 500001 to 562500
Starting task in debug mode for range: 812501 to 875000
Starting task in debug mode for range: 875001 to 937500
Starting task in debug mode for range: 125001 to 187500
Starting task in debug mode for range: 375001 to 437500
Starting task in debug mode for range: 437501 to 500000
Starting task in debug mode for range: 687501 to 750000
Starting task in debug mode for range: 625001 to 687500
Starting task in debug mode for range: 187501 to 250000
Starting task in debug mode for range: 1 to 62500
Starting task in debug mode for range: 562501 to 625000
Starting task in debug mode for range: 750001 to 812500
Task for range: 1 to 62500 finished. Max length: 340 for number: 52527. Time taken: 1457 ms
Task for range: 125001 to 187500 finished. Max length: 383 for number: 156159. Time taken: 1579
    ms
Task for range: 187501 to 250000 finished. Max length: 443 for number: 230631. Time taken: 1571
    ms
Task for range: 62501 to 125000 finished. Max length: 354 for number: 106239. Time taken: 1508 ms
Task for range: 250001 to 312500 finished. Max length: 407 for number: 263103. Time taken: 1663
    ms
Task for range: 312501 to 375000 finished. Max length: 441 for number: 345947. Time taken: 1671
    ms
Task for range: 625001 to 687500 finished. Max length: 509 for number: 626331. Time taken: 1668
    ms
Task for range: 437501 to 500000 finished. Max length: 444 for number: 461262. Time taken: 1707
    ms
Task for range: 375001 to 437500 finished. Max length: 449 for number: 410011. Time taken: 1717
    ms
Task for range: 750001 to 812500 finished. Max length: 468 for number: 767903. Time taken: 1709
    ms
Task for range: 687501 to 750000 finished. Max length: 504 for number: 704623. Time taken: 1773
    ms
Task for range: 562501 to 625000 finished. Max length: 447 for number: 615017. Time taken: 1782
    ms
Task for range: 500001 to 562500 finished. Max length: 470 for number: 511935. Time taken: 1790
    ms
Task for range: 812501 to 875000 finished. Max length: 525 for number: 837799. Time taken: 1773
    ms
Task for range: 875001 to 937500 finished. Max length: 476 for number: 910107. Time taken: 1771
    ms
Task for range: 937501 to 1000000 finished. Max length: 507 for number: 939497. Time taken: 1786
    ms
Calculation with 16 threads took: 1811 ms
```

Kodas 8: 16 gijų