



VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS

Ilgiausios Collatz'o iteracijos paieška duotajame skaičių intervale

Lygiagretieji skaičiavimai

Atliko: Tomas Kozakas

Vertino: Prof., Dr. Rimantas Vaicekuskas

Vilnius
2024

Turinys

1. Problemos formulavimas	3
2. Lygiagretusis algoritmas	4
3. Vykdymo aplinka	5
4. Eksperimentinio tyrimo rezultatai	7
5. Išvados apie pateiktojo algoritmo spartinimą bei plečiamumą	9
6. Programos kodas	10

1. Problemos formulavimas

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

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

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

- $A(0)$ - 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 kiekvienas Collatz sekos skaičius priklauso tik nuo savo pirmtako, skaičiavimą galima lengvai padalinti į dalis. Pvz., jei turime intervalą nuo 1 iki 1 milijono, galime jį suskirstyti į mažesnes dalis ir priskirti kiekvieną dalį atskirai gijai (žr. 2.1 lentelę).

Intervalas	Ilgiausias Collatz ilgis
1 iki 62500	340
62501 iki 125000	354
125001 iki 187500	383
187501 iki 250000	443
250001 iki 312500	407
312501 iki 375000	441
375001 iki 437500	449
437501 iki 500000	444
500001 iki 562500	470
562501 iki 625000	447
625001 iki 687500	509
687501 iki 750000	504
750001 iki 812500	468
812501 iki 875000	525
875001 iki 937500	476
937501 iki 1000000	507

2.1 lentelė. Ilgiausias Collatz ilgis kiekviename intervale

3. Vykdymo aplinka

Šiam projektui buvo naudojama mif superkompiuteris su šiais techniniais parametrais:

Pavadinimas	Mazgai	CPU	GPU	RAM	HDD	Tinklas
Main	35/36	48	0	384GiB	0	1Gbit/s, 2x10Gbit/s, 4xEDR(100Gbit/s)

3.1 lentelė. Pagrindinio Klasterio Techninės Specifikacijos

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. 3,4, 5 kodą).

Komandinis failas viduj MIF klasterio buvo sukonfigūruotas atitinkamai:

```
1 #!/bin/bash
2
3 START=$1
4 END=$2
5 MODE=$3
6 THREADS=$4
7
8 javac -d build src/*.java
9 java -Xss5m -Xmx1g -cp build Main $START $END $MODE $THREADS
```

Kodas 1: run.sh failas su programos paleidimu

```
1 #!/bin/bash
2
3 if [ "$#" -ne 4 ]; then
4     echo "Usage: ./run_collatz_experiment.sh <cores> <mode> <start> <end>"
5     exit 1
6 fi
7
8 CORES=$1
9 MODE=$2
10 START=$3
11 END=$4
12
13 mkdir -p output
14
15 for THREADS in 1 2 4 8 16 32; do
16     OUTPUT_FILE="output/collatz_c${CORES}_t${THREADS}.out"
17     echo "Running with $CORES core(s) from $START to $END with $THREADS thread(s) in $MODE mode."
18     sbatch --exclusive --ntasks=1 --nodes=1 --cpus-per-task $CORES -o $OUTPUT_FILE run.sh $START
19     echo "Find output in ${OUTPUT_FILE}"
20 done
```

Kodas 2: run_collatz_experiment.sh failas su experimentais

Šis skriptas (žr. 1, 2) yra skirtas automatiniam eksperimentų, vykdomų su SLURM. Kiekviena eksperimento užduotis yra vykdoma nustatant konkretų skaičių CPU branduolių ir keičiant gijų skaičių. Rezultatai kiekvienai konfigūracijai yra saugomi atskiruose failuose. Trumpas **sbatch**

komandas paaiškinimas:

1. **–exclusive**

Užtikrina, kad darbas naudos tik jam skirtus mazgus, neleidžiant kitiems darbams juos naudoti.

2. **–ntasks=1**

Nustato, kad bus vykdoma viena užduotis.

3. **–nodes=1**

Nurodo, kad užduotis bus vykdoma tik ant vieno mazgo.

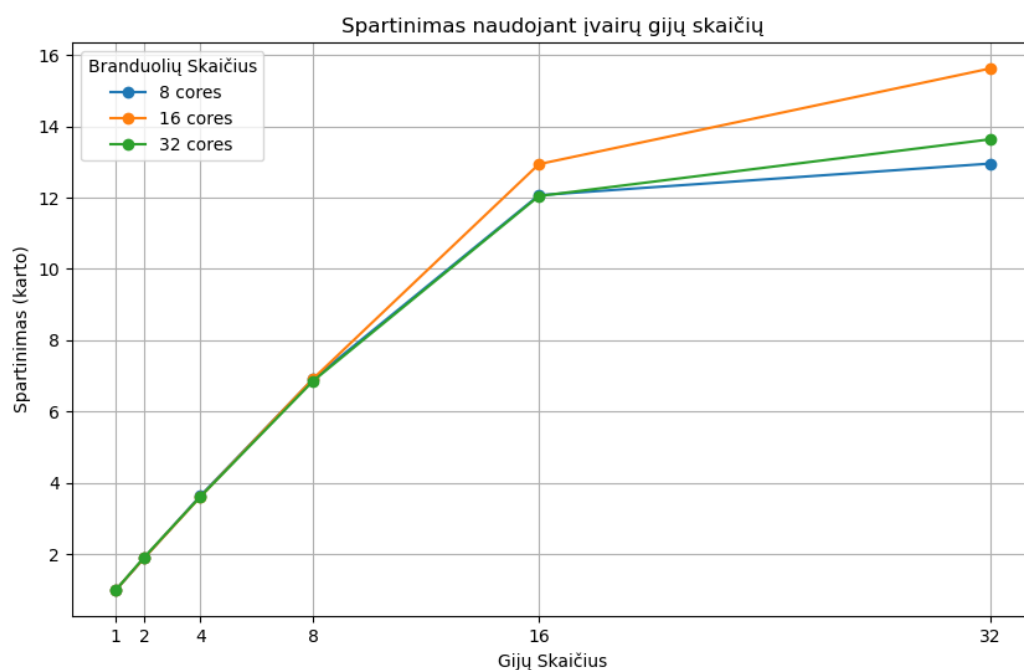
4. **–cpus-per-task=\$CORES**

Skiria nustatytą skaičių procesoriaus branduolių kiekvienai užduočiai.

4. Eksperimentinio tyrimo rezultatai

4.1 lentelė. Skaiciavimo laikai ms (milisekundės) nuo 1 iki 10 mln., priklausomai nuo branduolių ir gijų skaičiaus

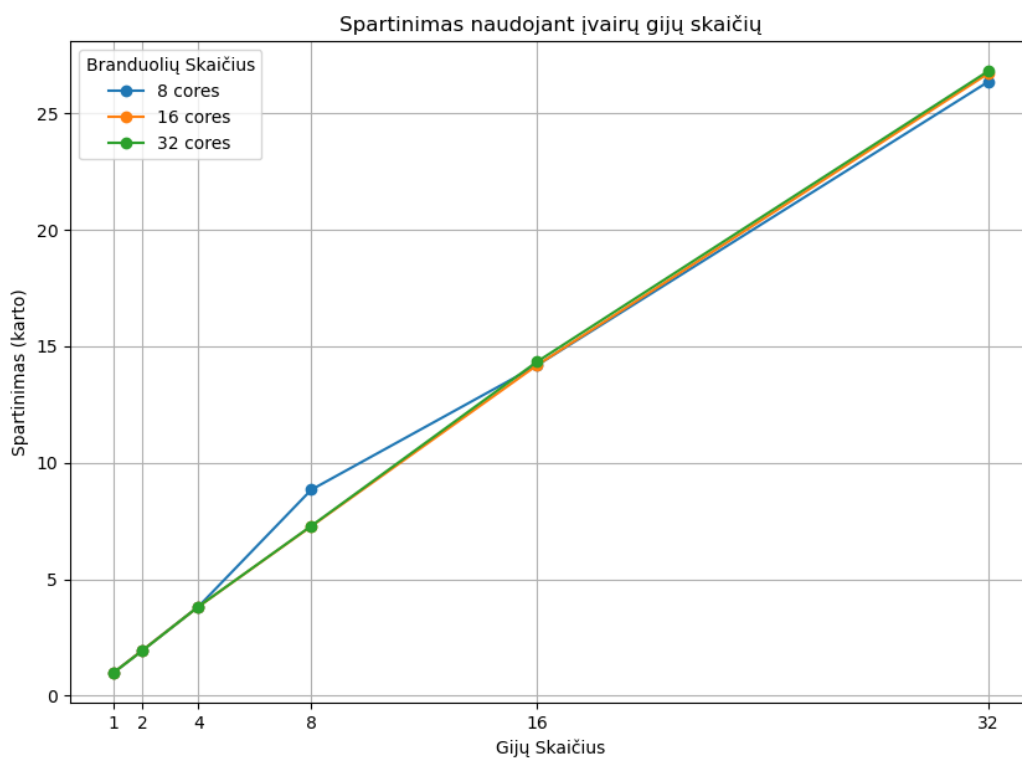
Branduolių Sk. / Gijų Sk.	1 gija	2 gijos	4 gijos	8 gijos	16 gijos	32 gijos
8 branduoliai	7229	3823	1985	1045	599	558
16 branduoliai	7233	3825	2001	1044	559	463
32 branduoliai	7227	3780	1998	1053	600	530



4.1 pav. Spartinimas nuo 1 iki 10 mln. naudojant įvairų gijų skaičių

4.2 lentelė. Skaiciavimo laikai ms (milisekundės) nuo 1 iki 1 mlrd., priklausomai nuo branduolių ir gijų skaičiaus

Branduolių Sk. / Gijų Sk.	1 gija	2 gijos	4 gijos	8 gijos	16 gijos	32 gijos
8 branduoliai	923846	477311	241465	104500	65167	35060
16 branduoliai	923935	477432	241620	127065	65151	34600
32 branduoliai	923851	479926	241427	126761	64455	34463



4.2 pav. Spartinimas nuo 1 iki 1 mlrd. naudojant įvairų gijų skaičių

Taigi, spartinimas vyksta tiek su mažais intervalais nuo 1 iki 10 mln. (žr. 4.1), tiek su dideliais nuo 1 iki 1 mlrd. (žr. 4.2) Tačiau matosi, kad skirtumas tarp intervalų yra ryškesnis didesniuose intervaluose.

5. Išvados apie pateiktojo algoritmo spartinimą bei plečiamumą

Iš pateiktų duomenų ir grafikų (žr. 4.1, 4.2), aiškiai matyti, kad skaičiavimų spartinimas didinant gijų skaičių yra ryškus visose branduolių konfigūracijose, tačiau spartinimas yra beveik tiesiškas tik dideliuose skaičiavimų intervaluose, pavyzdžiui, nuo 1 iki 1 milijardo. Mažesnius intervalus, kaip nurodyta lentelėje, kur skaičiavimo intervalas yra nuo 1 iki 10 milijonų. Pereinant nuo vienos iki dviejų gijų skaičiavimo laikas beveik perpus sumažėja - nuo 7229 ms iki 3823 ms. Tačiau tolesnis gijų skaičiaus didinimas jau nebeužtikrina tokio spartinimo. Spartinimą su 8 branduoliais galima pamatyti lentelėje: Spartinimas apskaičiuotas naudojant šią formulę:

5.1 lentelė. Spartinimas nuo 1 iki 10 mln. ir nuo 1 iki 1 mlrd.

Intervalas	Spartinimas (1 iki 10 mln.)	Spartinimas (1 iki 1 mlrd.)
1 gija	1.89	1.94
2 gijos	3.64	3.83
4 gijos	6.92	8.84
8 gijos	12.07	14.18
16 gijų	12.96	26.35
32 gijos	12.96	26.35

$$\text{Spartinimas} = \frac{\text{Pirmosios bazinės būsenos greitis}}{\text{Kiekvienos kito greitis}}$$

6. Programos kodas

```
1  public class Main {
2  public static void main(String[] args) {
3      if (args.length < 4) {
4          System.out.println("Usage: java CollatzCalculator <startRange> <endRange> <mode> <
5          threadsCount>");
6          System.out.println("<mode> should be 'debug' or 'fast'");
7          return;
8      }
9
10     long startRange = Long.parseLong(args[0]);
11     long endRange = Long.parseLong(args[1]);
12     String mode = args[2];
13     int threadsCount = Integer.parseInt(args[3]);
14
15     CollatzCalculator calculator = new CollatzCalculator(startRange, endRange, mode,
16     threadsCount);
17     calculator.run();
18 }
```

Kodas 3: Main Java class

```

1 import java.util.concurrent.ExecutorService;
2 import java.util.concurrent.Executors;
3 import java.util.concurrent.TimeUnit;
4
5 public class CollatzCalculator implements Runnable {
6     private final long startRange;
7     private final long endRange;
8     private final String mode;
9     private final int threadsCount;
10
11     public CollatzCalculator(long startRange, long endRange, String mode, int threadsCount) {
12         this.startRange = startRange;
13         this.endRange = endRange;
14         this.mode = mode;
15         this.threadsCount = threadsCount;
16     }
17
18     @Override
19     public void run() {
20         System.out.println("Operating in " + mode.toUpperCase() + " mode.");
21         System.out.println("Testing with " + threadsCount + " threads.");
22
23         long startTime = System.nanoTime();
24
25         runCalculations(startRange, endRange, threadsCount, mode);
26
27         long endTime = System.nanoTime();
28         long duration = TimeUnit.NANOSECONDS.toMillis(endTime - startTime);
29
30         System.out.println("Calculation with " + threadsCount + " threads took: " + duration + "
31 ms\n");
32     }
33
34     public static void runCalculations(long startRange, long endRange, int threadsCount, String
35 mode) {
36         ExecutorService executor = Executors.newFixedThreadPool(threadsCount);
37
38         long intervalLength = (endRange - startRange + 1) / threadsCount;
39         long currentStart = startRange;
40
41         for (int i = 0; i < threadsCount; i++) {
42             long currentEnd = (i == threadsCount - 1) ? endRange : currentStart + intervalLength
43 - 1;
44             executor.execute(new CollatzTask(currentStart, currentEnd, mode));
45             currentStart = currentEnd + 1;
46         }
47
48         executor.shutdown();
49         try {
50             executor.awaitTermination(Long.MAX_VALUE, TimeUnit.NANOSECONDS);
51         } catch (InterruptedException e) {
52             System.out.println("Execution was interrupted.");
53         }
54     }
55 }

```

Kodas 4: CollatzCalculator Java class

```

1 public class CollatzTask implements Runnable {
2     private final long startRange;
3     private final long endRange;
4     private final String mode;
5
6     public CollatzTask(long startRange, long endRange, String mode) {
7         this.startRange = startRange;
8         this.endRange = endRange;
9         this.mode = mode;
10    }
11
12    @Override
13    public void run() {
14        long startTime = System.currentTimeMillis();
15        if (mode.equals("debug")) {
16            System.out.println("Starting task in debug mode for range: " + startRange + " to " +
17                endRange);
18            try {
19                Thread.sleep(1000); // Sleep for 1 second
20            } catch (InterruptedException e) {
21                Thread.currentThread().interrupt();
22            }
23
24            long maxLengthNumber = 0;
25            int maxLength = 0;
26
27            for (long i = startRange; i <= endRange; i++) {
28                int length = calculateCollatzLength(i);
29                if (length > maxLength) {
30                    maxLength = length;
31                    maxLengthNumber = i;
32                }
33            }
34
35            long endTime = System.currentTimeMillis(); // End timing
36            long taskDuration = endTime - startTime; // Calculate duration
37
38            if ("debug".equals(mode)) {
39                System.out.println("Task for range: " + startRange + " to " + endRange + " finished.
40                    Max length: " +
41                        maxLength + " for number: " + maxLengthNumber + ". Time taken: " +
42                        taskDuration + " ms");
43            }
44
45            private int calculateCollatzLength(long n) {
46                if (n == 1) return 1;
47                long next = n % 2 == 0 ? n / 2 : 3 * n + 1;
48                return 1 + calculateCollatzLength(next);
49            }
50        }
51    }

```

Kodas 5: CollatzTask Java class