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. **??** lentelė).

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:

```
#!/bin/bash

START=$1
END=$2
MODE=$3
THREADS=$4

javac -d build src/*.java
java -Xss5m -Xmx1g -cp build Main $START $END $MODE $THREADS
```

Kodas 1: run.sh failas su programos paleidimu

```
#!/bin/bash

if [ "$#" -ne 4 ]; then
    echo "Usage: ./run_collatz_experiment.sh <cores> <mode> <start> <end>"
    exit 1
fi

CORES=$1
MODE=$2
START=$3
END=$4

mkdir -p output

for THREADS in 1 2 4 8 16 32; do
  OUTPUT_FILE="output/collatz_c${CORES}_t${THREADS}.out"
  echo "Running with $CORES core(s) from $START to $END with $THREADS thread(s) in $MODE mode."
  sbatch --exclusive --ntasks=1 --nodes=1 --cpus-per-task $CORES -o $OUTPUT_FILE run.sh $START
    $END $MODE $THREADS
  echo "Find output in ${OUTPUT_FILE}"
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.

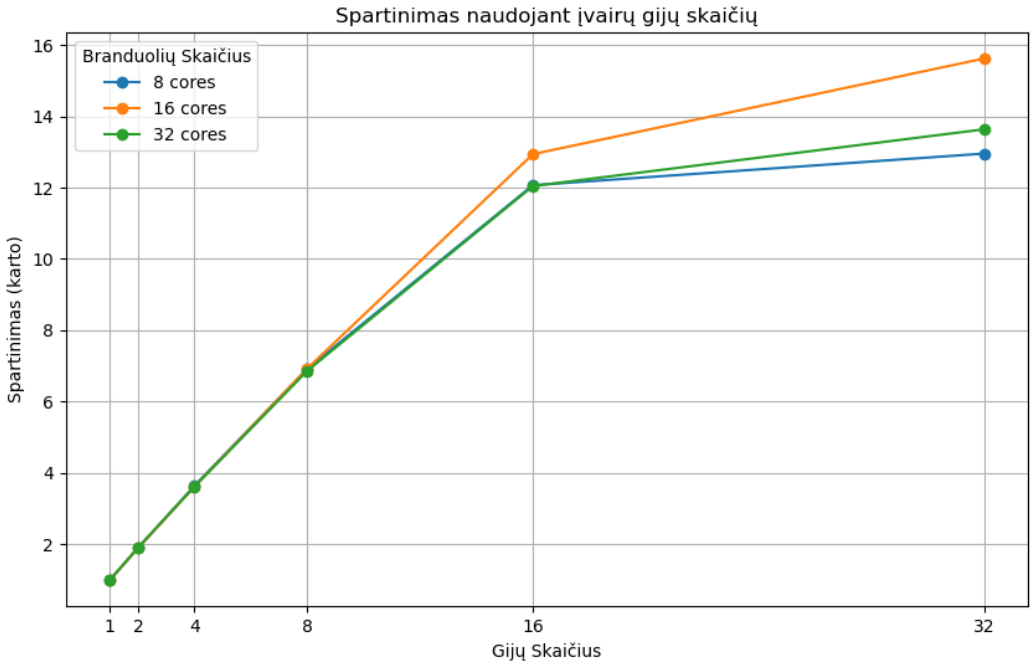
3. **–nodes=1**
   Nurodo, kad užduotis bus vykdoma tik ant vieno mazgo.

4. **–cpus-per-task=$CORES**
   Skiria nustatytą skaičių procesoriaus branduolių kiekvienai užduočiai.

4. Eksperimentinio tyrimo rezultatai

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

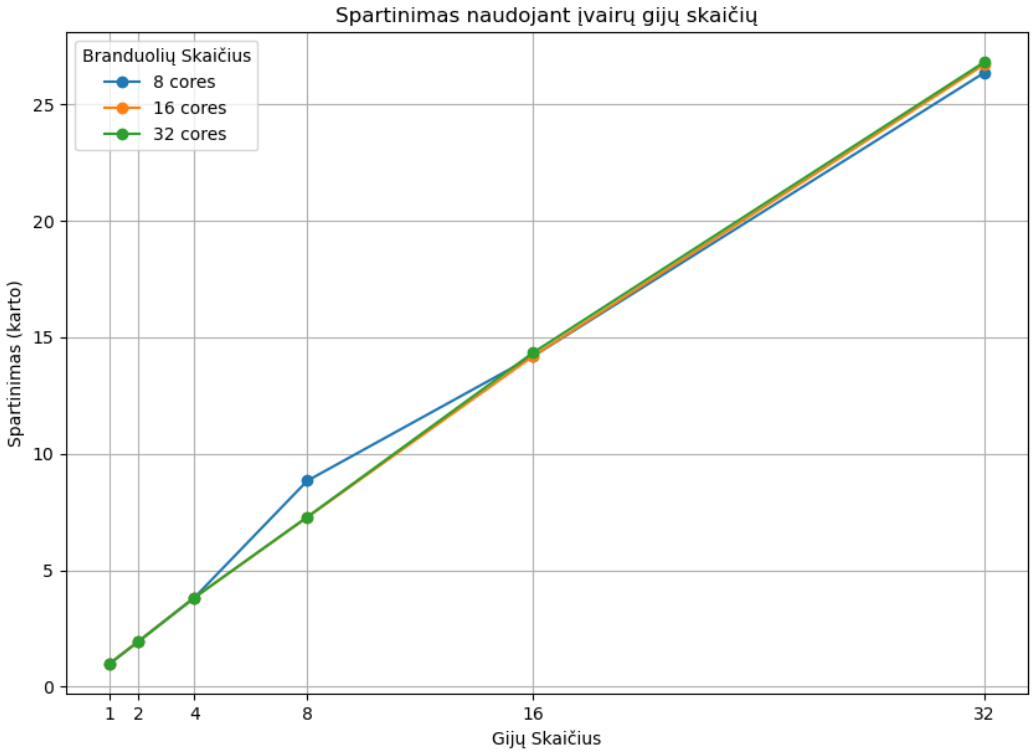
Branduolių Sk. / Gijų Sk.	1 gija	2 gijos	4 gijos	8 gijos	16 gijos	32 gijos
8 branduoliai	7229	3823	1985	1045	599	558
16 branduoliai	7233	3825	2001	1044	559	463
32 branduoliai	7227	3780	1998	1053	600	530



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

4.2 lentelė. Skaičiavimo 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

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 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 (4.1), matome, kad 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 greito spartinimo. Pavyzdžiui, pereinant nuo 16 iki 32 gijų, spartinimas yra mažiau akivaizdus. Kita vertus, kai apsvarstome didesnius intervalus, pavyzdžiui nuo 1 iki 1 milijardo, spartinimo efektas išlieka beveik tiesiškas, laikas mažėja beveik dvigubai, atsižvelgiant į didesnį gijų skaičių.

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 3: Main 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 4: CollatzCalculator Java class

```java
public class CollatzTask implements Runnable {
    private final long startRange;
    private final long endRange;
    private final String mode;

    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;
        long next = n % 2 == 0 ? n / 2 : 3 * n + 1;
        return 1 + calculateCollatzLength(next);
    }
}
```

Kodas 5: CollatzTask Java class