Parallelism and Distribution

Opdracht parallellisme: Analyse van COVID-19-infecties

Professor: Elisa Gonzalez Boix
Assistent: Jens Van der Plas
E-mail: jevdplas@vub.be

1 Projectoverzicht

In dit project ga je data van COVID-19-patiënten analyseren. De data die je gaat onderzoeken is gebaseerd op de data die door het Amerikaanse *Centers for Disease Control and Prevention* (CDC) vergaard werd. Het project bestaat uit drie delen: een parallelle implementatie d.m.v. Java Fork/Join, een evaluatie van de performantie, en een verslag dat de implementatie, je evaluatie, en de interpretatie van je resultaten beschrijft.

Deadline 27 mei 2022, 23:59 CEST. Laattijdige inzendingen zullen tot een puntenaftrek leiden. Inzendingen na 30 mei zullen automatisch met een 0 gequoteerd worden. Indien je een probleem ondervindt waardoor je niet tijdig kan inzenden, gelieve ons op voorhand te contacteren.

Deliverable Maak een zip-bestand van je code en je verslag als PDF en dien deze in via de respectievelijke opdracht voor dit project in de cursusruimte op Canvas. Let op het volgende:

- Indien je project afhankelijkheden heeft die niet standaard in Java zitten, voeg deze dan toe.
- Je verslag heeft de naam `<netid>-verslag-fj.pdf`[1]. Geef de zip de naam`<netid>-fj.zip`.
- Zorg ervoor dat ieder bestand met code bovenaan je netid (in comments) bevat.

Evaluatie Je zal beoordeeld worden op basis van de *correctheid van de implementatie* volgens de specificatie, het *correct gebruik van Java Fork/Join*, je *benchmarks*, de *kwaliteit van je code* (voldoende commentaar, duidelijke namen voor variabelen, een goede structuur,...), de *kwaliteit van het verslag* (zowel inhoudelijk, bv. de interpretatie van de resultaten, als qua vorm), en de *verdediging* tijdens de examenperiode.

Let op: Deze opdracht bepaalt 25% van je eindscore, en is slechts één onderdeel van de twee opdrachten die je moet maken (de andere is de opdracht voor distributie). Je zal alleen geëvalueerd worden in deze zittijd indien je beide opdrachten indient.

Lees eerst de projectopgave volledig door vooraleer je begint met de uitvoering. Om je te helpen met je project vind je op Canvas skeleton code (zie Sectie 3) en richtlijnen voor het benchmarken (zie Sectie 4). Als je vragen over dit project hebt, aarzel dan niet om ons tijdig te contacteren via pnd@dinf.vub.ac.be.

Belangrijk!

Deze opdracht moet alleen gemaakt worden. Plagiaat is een serieuze overtreding en wordt bestraft! Elektronische tools worden gebruikt om alle oplossingen met elkaar te vergelijken, zelfs over verschillende academiejaren heen. Elk vermoeden van uitwisseling van code zal onverwijld aan de decaan van

[1] Je *NetID* (netwerkidentiteit) is een verkorte versie van je naam, zoals bv. *jevdplas*. Deze wordt aan iedere student op VUB toegekend en is dus *niet* gelijk aan je studentennummer. Je gebruikt je netid o.a. om in te loggen op CaLi.

de faculteit worden gerapporteerd. Zowel gebruiker als verlener van zulke code zullen gesanctioneerd worden conform de regels van het examenreglement.

Bij deze projectopgave voorzien we code die je kan gebruiken om je op weg te helpen. De enige andere code die, mits expliciete vermelding in het verslag, eventueel overgenomen en aangepast mag worden is (exhaustief) de code uit de werkcolleges (opgaven en oplossingen), de code uit de slides van de hoorcolleges, en de code die op Canvas ter beschikking gesteld wordt in het bestand `pd-code-chapter2.zip`. Het kopiëren, overnemen en/of aanpassen van code uit om het even welke andere bron (inclusief websites die vermeld werden tijdens de les) wordt dus expliciet verboden, zal als plagiaat beschouwd worden en zal dus ook als dusdanig bestraft worden. Samenwerken of je code delen met medestudenten is dus verboden: je moet al je code zelf geschreven hebben. **Ook het overnemen van (delen van) code of algoritmes van externe bronnen zoals het Internet is dus expliciet niet toegelaten.** Op de verdediging zal je in staat moeten zijn om (detail-)vragen in verband met de werking van al je code te beantwoorden. Contacteer ons als je twijfelt of iets al dan niet als plagiaat zou beschouwd worden.

Veel succes!

2 Analyse van COVID-19-infecties

De laatste twee jaar werd de wereld in de ban gehouden door COVID-19, een erg besmettelijke ziekte. Gedurende deze pandemie hebben allerlei gezondheidsinstanties, zoals het Amerikaanse *Centers for Disease Control and Prevention* (CDC), dan ook allerlei data betreffende COVID-19-infecties vergaard en bijgehouden. Bovendien stelt het CDC online datasets beschikbaar op haar website[2].

Om onderzoek naar het coronavirus te doen, willen wetenschappers van het CDC enkele queries op de dataset kunnen uitvoeren. Echter, het sequentieel verwerken van de steeds groter wordende dataset begint meer en meer tijd in beslag te nemen. Daarom moet het CDC het verwerken van de data nu parallelliseren, zodat de wetenschappers snel antwoord kunnen krijgen op hun prangende vragen.

Helaas! Na een uit de hand gelopen lockdownfeestje hebben alle programmeurs van het CDC die iets van parallellisme kennen een infectie met het virus opgelopen. Zij zijn dus tijdelijk buiten strijd. Nochtans heeft het CDC van de President een strakke deadline gekregen, niets doen is dus geen optie! Daarom heeft het CDC nu jou expertise nodig om reeds enkele van hun analyses te kunnen parallelliseren.

In dit project ga je twee eenvoudige queries van COVID-19-infecties parallelliseren. De eerste query berekent eenvoudige metrieken, in casu enkele totalen die een globaal overzicht van de data geven. Hoeveel mannen en vrouwen zijn er bijvoorbeeld in totaal besmet, en hoeveel ouderen? De tweede query is complexer en peilt naar het aantal dagen vanaf het moment waarom een gegeven aantal vrouwen reeds besmet werd tot en met het moment vanaf waarop er nog een gegeven aantal personen in de dataset besmet zouden worden die comorbiditeiten (andere aandoeningen zoals bv. diabetes) hebben en die op een afdeling intensieve zorgen opgenomen worden (of omgekeerd, indien het tweede moment eerst komt).

Om de queries zo snel mogelijk te kunnen uitvoeren, ga je het process parallelliseren met behulp van Java Fork/Join. Concreet ga je in twee fases telkens één query parallelliseren. Het is natuurlijk de bedoeling dat je dit zo efficiënt mogelijk doet. Nadien ga je je implementatie evalueren en beschrijf je alles (je implementatie, evaluatie, en de analyse van je resultaten a.d.h.v. gegeven vragen) in een kort rapport.

[2]Dit project is gebaseerd op deze dataset (update van 4 maart 2022).

3 Implementatie

Op Canvas vind je bij deze opgave een Java project dat reeds de volgende code bevat en waarop je je project dient uit te werken:

- `CovidAnalyser`, een interface die COVID-19-analyses moeten implementeren.
- `SequentialAnalyser`, een sequentiële implementatie van de gevraagde functionaliteit.
- `Reader`, een klasse die methoden aanbiedt om de dataset in te lezen, alsook om nieuwe data te genereren m.b.v. de `generateData`-methode.
- Enkele korte unittests die je kunnen helpen om te verifiëren of je parallelle analyse correct werkt.

Het doel van dit project is om de klasse `ParallelAnalyser`, een parallelle versie van `SequentialAnalyser`, met behulp van Fork/Join te implementeren (een skelet van deze klasse is reeds gegeven). De implementatie verloopt in twee fases, waarvan je de implementatie ook moet evalueren.

3.1 Fase 1: Totalen berekenen

In fase 1 implementeer je een query die enkele totalen berekent, om zo een overzicht van de data te krijgen. Concreet dien je na te gaan (1) hoeveel gevallen er "lab-confirmed" zijn (dus waarvan de infectie met een positieve test bevestigd werd), (2) hoeveel patiënten er man zijn, (3) hoeveel patiënten er vrouw zijn, (4) hoeveel ouderen (van 60 jaar of ouder) er zijn, (5) hoeveel patiënten er gehospitaliseerd werden, (5) hoeveel patiënten op de afdeling intensieve zorgen opgenomen werden, (6) hoeveel personen overleden aan het virus, en (7) hoeveel personen comorbiditeiten (andere aandoeningen) hadden. Het is hierbij dus de bedoeling dat je deze waarden voor de patiënten in de dataset in parallel berekent met behulp van Java Fork/Join. Je gaat dus de `phaseone`-methode van `ParallelAnalyser` moeten implementeren zodat deze een instantie van de `Metrics`-klasse teruggeeft die de correcte totalen bevat.

3.2 Fase 2: Een tijdsinterval berekenen

In fase 2 implementeer je een meer complexe query. Concreet gezien wil je de volgende vraag kunnen beantwoorden: "Hoeveel dagen tellen we, vanaf er `numFemales` vrouwen besmet zijn, tot en met het moment vanaf waarop er nog `numICU` personen besmet geraken die comorbiditeiten hebben en op de afdeling intensieve zorgen opgenomen worden (of omgekeerd, indien het tweede moment eerst komt)?" Voor het vaststellen van de tweede datum moet je natuurlijk enkel rekening houden met de personen in de dataset. Bovendien, om het je makkelijker te maken, is de data reeds oplopend gesorteerd op datum. Voor deze fase is het ook de bedoeling dat je een implementatie van deze query maakt met behulp van Java Fork/Join. Je gaat dus de `phaseTwo`-methode van `ParallelAnalyser` moeten implementeren, zodat deze het juiste aantal dagen teruggeeft.

3.3 Specifieke implementatievereisten

We sommen hier nog even enkele specifieke implementatievereisten op, waaraan je implementatie eveneens dient te voldoen.

- Logischerwijs dient je implementatie correct te zijn en steeds hetzelfde resultaat terug te geven als de sequentiële code. De bijgeleverde unittests kunnen je helpen om dit na te gaan, deze moeten dus minstens slagen.
- Je gebruikt zo weinig mogelijk verschillende Fork/Join-taken (subklassen van `RecursiveTask` en `RecursiveAction`). Probeer dus zo veel mogelijk werk per parallellisatie uit te voeren.

- De implementatie streeft naar efficiëntie en past Fork/Join correct toe. Gebruik Fork/Join zoals gezien in de les (en beschreven in de cursustekst) en vermijd dure operaties (dynamische geheugenallocaties, frequente toegang tot geheugen dat niet in de cache zit,...). Langs de andere kant, vermijd ook micro-optimalisaties die de leesbaarheid van de code verminderen. Zorg er in ieder geval voor dat je voldoende commentaar aan je code toevoegt, zodat dit het lezen en begrijpen van de code bevordert.
- Voor het benchmarken (zie Sectie 4 van dit document) zal je implementatie voor beide fases een sequentiële cut-off moeten bevatten. De gewenste cut-off geef je mee aan de constructor van `ParallelAnalyser`.
- Zoals vermeld in het werkcollege wordt het afgeraden om in je implementatie Java lambda's te gebruiken gezien deze (blijkbaar) een invloed kunnen hebben op de performantie van je implementatie. Hou ook rekening met de Java-versie beschikbaar op Firefly (zie Sectie 4.4.3).

4 Benchmarks

Het hoofddoel van je experimenten is om de performantie te evalueren van je parallelle implementatie. Allereerst ga je de performantie van je oplossing benchmarken op je eigen laptop, en daarna ga je de benchmarks herhalen op Firefly (een krachtige server die op het SOFT-lab staat). Deze sectie beschrijft alleen hoe je je benchmarks moet uitvoeren. In Sectie 5 vind je de vragen die je zal moeten beantwoorden over de benchmarks.

4.1 Dataset

Om te kunnen benchmarken, heb je natuurlijk data nodig. Omdat het inlezen van de eigenlijke dataset van het CDC te veel tijd in beslag neemt (omwille van o.a. file IO), voorzien we code die voor jou een dataset van patiënten genereert. Hiermee is het eveneens mogelijk om het gewenste aantal patiënten te genereren, wat kan helpen om te benchmarken. Om een dataset te genereren, dien je de `generateData`-methode van de `Reader`-klasse aan te roepen. Deze methode neemt een `int` als argument die aangeeft hoeveel patiënten er gegenereerd moeten worden. De dataset die door deze methode gegenereerd wordt, is reeds oplopend gesorteerd op datum. Dit kan je uitbuiten in de implementatie van fase 2. We hebben ervoor gezorgd dat de gegenereerde data ongeveer dezelfde karakteristieken heeft dan de eigenlijke dataset van het CDC (bv. hetzelfde percentage vrouwen en personen met comorbiditeiten).

4.2 Optimale threshold

Benchmark je implementaties voor fases 1 en 2 met een dataset die minstens 100 miljoen patiënten bevat om voor iedere implementatie de optimale threshold te vinden.

- Bereken de overhead van je implementaties voor verschillende waarden van de threshold. Je bepaalt zelf welke voor welke waarden van de threshold je je benchmarks gaat runnen. Kies deze zorgvuldig!
- Bereken voor `P=<#cores van je computer>` de application speed-up van je implementatie voor verschillende waarden van de threshold.

4.3 Speed-up

Voor de implementaties van fase 1 en fase 2 ga je nu benchmarks uitvoeren om uit te zoeken hoe goed deze schalen met de beschikbare hoeveelheid cores. Benchmark de implementaties met een dataset die minstens 100 miljoen patiënten bevat. Kies bij de benchmarks een sequentiële cut-off

waarvan jij denkt dat die optimaal is voor de betreffende implementatie, gegeven je resultaten voor de benchmarks in Sectie 4.2. Je verantwoordt deze keuze in het verslag.

- Bereken de computational en application speed-up van je implementaties voor verschillende waarden van `P` (bv. 1, 2, 3, 4, 5, 6, 7 en 8 cores). Varieer `P` zodat je zeker alle cores van je computer gebruikt.

4.4 Benchmarks op Firefly

Firefly is een krachtige server met 128 logische cores en 128GB geheugen. De server is daarom uitermate geschikt om experimenten uit te voeren waar veel parallellisme aan te pas komt.

4.4.1 Experimenten en dataset

Herhaal de benchmarks voor fases 1 en 2 en maak, waar van toepassing, optimaal gebruik van alle cores en geheugen van Firefly. Je hoeft hier niet je benchmarks te herhalen voor *elk* aantal threads (1, 2, 3, . . . , 127, 128), maar zorg ervoor dat je een beeld krijgt over hoe je implementaties schalen naar meerdere cores.

Om te benchmarken op Firefly kan je van een grotere dataset gebruikmaken, je kan bijvoorbeeld 1 miljard patiënten genereren. Hiervoor zal je de JVM natuurlijk genoeg geheugen moeten geven.

4.4.2 Reserveren van Firefly

Om je benchmarks te runnen op Firefly moet je een *tijdslot* reserveren. Een tijdslot is een periode van 12 uur waar je de enige gebruiker bent op Firefly. Vanaf 4 april tot en met 20 mei zullen er 2 slots per dag zijn, van 05u00 tot 17u00 en omgekeerd. **Je krijgt slechts *één* slot om je benchmarks te runnen.** Hoewel je tijdens de bovenstaande periode continu toegang tot Firefly hebt, dien je ervoor te zorgen dat je je slot strikt respecteert. Immers, indien je buiten de grenzen van je slot op Firefly werkt of iets op de server hebt runnen, breng je mogelijk de uitvoering van de experimenten van je medestudenten in gevaar!

Hou dus de volgende punten in gedachten bij het gebruik van Firefly:

- Reken op voorhand (aan het begin van je slot) uit dat je benchmarks kunnen voltooien binnen de grenzen van je slot (op basis van het aantal benchmarks, het aantal herhalingen, de gemiddelde runtime,...). Op Firefly zal je immers een grotere dataset gebruiken dan lokaal op je computer. Schrijf resultaten ook zo snel mogelijk weg naar een bestand. Zo verlies je geen data moest er iets mislopen in de rest van het programma.
- Controleer aan het begin van je slot of er geen experimenten van de vorige student meer draaien. Indien dit het geval is, probeer dan eerst even je medestudent te contacteren vooraleer ons een email te sturen.
- Indien je merkt dat je experimenten aan het einde van je slot nog niet klaar zijn, dien je deze zelf te beëindigen. Indien je tijdens het benchmarken reeds merkt dat je experimenten te lang zullen duren, kan je natuurlijk reeds ingrijpen. Aan het einde van je slot moet je al je processen (incl. al je `tmux`-sessies[3]) beëindigd hebben. Met het commando `tmux ls` kan je controleren of je al je tmux-sessies gesloten hebt. **Het niet respecteren van de grenzen van je tijdslot zal mogelijkerwijs een impact op je score hebben.**
- Je mag de server **enkel** binnen het kader van dit project gebruiken.
- **Indien je technische problemen ondervindt tijdens je slot op de server (bv. je hebt geen toegang tot de server (meer), een andere gebruiker is nog bezig,...), aarzel dan niet om**

[3] Zie het WPO van week 28.

een email te sturen naar pnd@dinf.vub.ac.be. We zullen je dan zo snel mogelijk proberen te helpen.

Om je te helpen vertrouwd te worden met Firefly, vond er in week 28 een WPO plaats waarin je je toegang tot de server kon testen en via enkele oefeningen vertrouwd kon worden met het werken op een server. In de opgave van dit WPO vind je enkele nuttige aanwijzingen om op de server te benchmarken.

Hou er rekening mee dat slots volzet kunnen raken. Begin dus zo snel mogelijk aan dit project en reserveer tijdig je slot. Het reserveren van een slot kan je hier doen.

4.4.3 Firefly's specificaties

Tot slot vind je hieronder enkele van Firefly's specificaties:

- **Hostname:** firefly.vub.ac.be
- **Hardware:**
  - **Model:** Custom built
  - **Year:** 2021
  - **CPU:** AMD Ryzen™ Threadripper™ 3990X Processor (64 cores @ 2.9Ghz base, 4.3Ghz boost, 128 threads)
  - **RAM:** 128 GB DDR4-3200
  - **SSD:** 2x2TB nvme
- **Software:**
  - **OS:** Ubuntu 20.04.2 LTS
  - **Java:** openjdk version 14.0.2 (Java 14)

4.5 Richtlijnen voor benchmarken

Hoewel je veel vrijheid hebt in de manier waarop je je experimenten opstelt, vragen we wel dat je dit doet op een wetenschappelijke manier. Daarom dien je de richtlijnen in het document "Guidelines for Performance Evaluation" te volgen en toe te passen (je wordt hier ook op geëvalueerd). Dit bestand wordt samen met deze opgave op Canvas gepubliceerd. Hou zeker ook rekening met de cursusspecifieke richtlijnen die in Sectie 4 van het bestand beschreven staan.

Enkele opmerkingen:

- Voor je project mag je benchmarken zoals we in het WPO gezien hebben, met een (vooraf) bepaalde w (het aantal warm-up iteraties, bv. 5). Je hoeft dus niet zelf experimenteel de steady-state-performantie te bepalen.
- Merk op dat het bestand gewag maakt van het "Java Microbenchmark Harness (JMH)" voor het benchmarken van Java programma's. Dit framework wordt vaak gebruikt voor het uitvoeren van microbenchmarks op de JVM, gezien het helpt om vaak voorkomende moeilijkheden te vermijden. Voor dit project hoef je dit framework *niet* te gebruiken. Indien je dit framework wel correct (!) gebruikt, en de resultaten hiervan correct interpreteert, zal je hiervoor een bonus krijgen.
- In bovenstaande secties gaven we reeds enkele richtlijnen m.b.t. hoeveel patiënten je dient te genereren voor ieder experiment. Kies het aantal patiënten zo dat je experimenten niet te kort runnen (zodat je makkelijk speed-ups kan zien), maar kies dit ook niet te groot (zodat je nog

voldoende geheugen hebt). **Omdat je hierin redelijk vrij bent, dien je voor ieder experiment duidelijk in je verslag aan te geven hoe groot de dataset die je gebruikt hebt was.**

Met minder patiënten zullen je experimenten vanzelfsprekend sneller uitgevoerd worden. Echter, bij korte runtimes wordt de invloed van externe factoren groter, en zal er dus meer variatie in je gemeten runtimes zijn. Je zal je experimenten dan ook voldoende moeten herhalen. Wanneer je experimenten bijvoorbeeld slechts 10-100ms duren, kan je je experimenten bijvoorbeeld meer dan 1000 keer herhalen.

Indien je veel patiënten genereert (veel meer dan 1 miljard), kan je mogelijk te maken krijgen met integer overflows, waarbij Java's integers niet groot genoeg meer zijn om berekende getallen te bevatten. In dit geval dien je ofwel minder data te genereren, of kan je je code aanpassen. Vermeld dit dat zeker in je verslag.

5 Verslag

Het verslag is opgedeeld in 2 delen: de beschrijving van je implementatie, en de evaluatie van beide fases op je eigen machine en op Firefly. Hou in je verslag rekening met de richtlijnen in het bestand "Guidelines for Performance Evaluation" betreffende het rapporteren van je experimenten en resultaten. Je verslag, figuren niet meegerekend, is niet langer dan 6 bladzijden. Het heeft de volgende structuur:

1. **Implementatie fase 1 en 2**

   1.1. **Fase 1: Totalen berekenen**
   Welke klassen implementeren fase 1? Heb je hier iets speciaal op aan te merken? (max. 400 woorden)

   1.2. **Fase 2: Een tijdsinterval berekenen**
   Welke klassen zijn verantwoordelijk voor het berekenen van de complexe query? Heb je hier iets speciaal op aan te merken? (max. 400 woorden)

   1.3. **Patronen**
   Heb je in je implementatie een Fork/Join-patroon uit de les (map, reductie, prefix, pack, parallel sorteren) kunnen gebruiken? Leg voor iedere fase uit waarom je wel/geen patro(o)n(en) kon gebruiken. (max. 400 woorden)

2. **Evaluatie**
   In deze sectie bespreek je de resultaten van je benchmarks. Het is belangrijk dat je hier een kritische en analytische blik werpt op je resultaten en dus **uitlegt** wat de resultaten betekenen voor je programma, alsook **waarom** je deze resultaten ziet. Rapporteer over deze resultaten met wetenschappelijk verantwoorde grafische voorstellingen (bv. met box plots, histogrammen, enz.), conform de richtlijnen die je gekregen hebt in het reeds vermelde document. Verwijs in je tekst duidelijk naar corresponderende figuren. Indien van toepassing, geef duidelijk aan wat je verwachtingen waren.

   Bespreek in volgende secties voor iedere vraag telkens eerst de resultaten van de benchmarks op je eigen computer, en daarna de resultaten van de benchmarks op Firefly in vergelijking met de benchmarks op je eigen computer. Doe dit voor je implementatie van beide fases. **Beantwoord iedere vraag afzonderlijk, in een aparte sectie die aangeduid is met het corresponderende vraagnummer.**

   2.1. **Uitgevoerde experimenten en experimentele set-up**
   Bespreek in deze sectie alle informatie die relevant is met betrekking tot de uitvoering van je benchmarks, conform de richtlijnen die je gekregen hebt in het reeds vermelde document. Zorg ervoor dat het duidelijk is hoe je jouw experimenten uitgevoerd hebt, hoe je experimentele set-up eruit ziet (bv. informatie over je computer), en wat je gemeten hebt.

Vergeet zeker ook niet te vermelden hoe groot de dataset is die je voor ieder experiment gebruikt hebt.

2.2. Optimale threshold

In deze sectie bespreek je, aan de hand van je resultaten voor de experimenten uit Sectie 4.2, de sequentiële cut-offs van je implementaties voor beide fases en hun invloed op de overhead en speed-up. Je bekijkt eveneens welke threshold het beste is.

1. Wat is de overhead van je implementaties? Is deze overhead acceptabel? Waardoor wordt deze overhead veroorzaakt?
2. Op welke manier heeft de threshold invloed op de overhead en speed-up van je implementaties? Wat is de invloed van het vergroten of verkleinen van de threshold? Leg uit.
3. Indien je de threshold naïef zou moeten bepalen (zonder te benchmarken), hoe zou je dit dan aanpakken? Welke afweging maak je?
4. Op basis van je metingen, welke threshold is de beste voor iedere implementatie? Waarom?
5. Is de threshold uit het vorige puntje voor beide fases hetzelfde? Indien wel, hoe komt dit? Indien niet, wat kan/kunnen hier de reden(en) voor zijn?

2.3. Speed-up

In deze sectie bespreek je de speed-ups van je implementaties, gegeven de optimale threshold(s) die je in de vorige sectie bepaald hebt. Gebruik hiervoor je resultaten voor de experimenten uit Sectie 4.3.

1. Wat zijn de speed-ups van je implementaties? Zijn deze speed-ups hoger of lager dan je verwacht? Waarom?
2. Welke speed-up is groter: de computational of de application speed-up? Is dit wat je verwacht? Waarom?
3. Hoe goed schalen je implementaties? Waarom schalen deze (niet) goed?

2.4. Implementatiekarakteristieken

In deze sectie ga je enkele karakteristieken van je implementatie onderzoeken. Je dient onderstaande vragen enkel voor de implementatie van fase 2 te beantwoorden.

1. Wat zijn `work` en `span` van de computatie? Leg duidelijk je berekening uit. (We verwachten hier een asymptotische complexiteit.)
2. Hoeveel taken worden er door je implementatie aangemaakt? (We verwachten hier eveneens een asymptotische complexiteit.)
3. Wat is het gemiddelde parallellisme van de computatie? Wat zegt dit over je gemeten resultaten?
4. Is de behaalde speed-up op je eigen computer en op Firefly asymptotisch optimaal (gegeven de beste threshold)?