Ontwerp

Bij het maken van een multi-threaded versie van het algoritme zorgen we ervoor dat de workload verdeeld wordt over meerdere threads. Dit doen we door een deel van de zeef te calculeren per thread.

We doen dit door een start positie en eind positie te maken voor elke thread. De 1e thread begint bij 0 en eindigt waar de volgende thread begint. Zo krijgt thread 1 bijvoorbeeld bij een N van 100 de 1e 50, en de 2e thread krijgt 50 tot 100, als er maar 2 threads zou zijn.

Thread 2 begint dan met k=2 net als wanneer er geen multi-threading zou zijn. Maar hierna wordt de index berekent om zo alleen vanaf zijn deel te starten.

Dat doen we door te kijken welk getal vanaf 50 het eerst geen restant heeft wanneer we het delen door k, in dit geval is dat dan 50, maar als in thread 2 zijn start positie 51 zou zijn dan zou de index 52 worden.

Dan vanaf 50 gaan we met een for-loop alle posities in de zeef dat deelbaar is door 2 omzetten naar false. Dus 50 wordt false, dan 52 dan 54 etc. Dit doen we met de volgende speudo-code:

index = get_index(start_position, end_position, k)

For i in range(index + k, eindpositie, k) # vanaf index, tot eind positie, met stappen van k groot.

Zeef[i] = False

We zien dat de index zelf hiermee overgeslagen wordt. Dit is voor als de index hetzelfde is als k. Dit checken we, en als dat het geval niet is maken we hem alsnog false:

If index != k:

    Zeef[index] = false

Bij K=7 zal de index 56 worden, omdat dat het eerste getal is dat deelbaar is door 7. Dan wordt 56 false, dan 63, dan 70 etc.

Als de index al door een vorige k false is gemaakt wordt deze k overgeslagen, dat is met k=4 zo waar de index 50 zou zijn, maar door k=2 is 50 al false gemaakt en hoeft k=4 niet berekent te worden.

We stoppen zodra k*2 groter is dan de eindpositie (100), en daarbij hebben we alle priemgetallen tussen 50 en 100. We tellen het aantal priemgetallen tussen 50 en 100 op. Bij thread 1 gebeurt ditzelfde process waar het aantal priemgetallen tussen 1 en 50 wordt geteld.

Daarna wordt het aantal priemgetallen van elke thread bij elkaar opgeteld.

amount_primes = comm.reduce(amount_primes, MPI.SUM, 0)

Voorbeeld

We gaan met 2 threads en een N van 100, zoals hiervoor al als voorbeeld gebruikt, het gehele programma door.

Thread 1 krijgt start positie 1 en eindpositie 49. Een zeef van 100 booleans worden aangemaakt, allemaal True behalve de eerste.

K begint met 2 en word met 1 verhoogt tot k**2 groter is dan de eind positie (49)

Zeef[k] wordt gekeken of dat niet al false is, zo niet gaan we met een for-loop door alle andere posities die deelbaar zijn door k en worden dan false gemaakt.

K wordt met 1 verhoogt en de loop begint opnieuw. Na de loop wordt het aantal priemgetallen berekent tussen 1 en 49

In thread 2 doen we hetzelfde als thread 1 maar beginnen we met start positie 50 en eindpositie 100.

K begint met 2 en word met 1 verhoogt tot k**2 groter is dan de eind positie (100)

De index wordt berekend, dat is nu 50

Zeef[index] wordt gekeken of dat niet al false is, zo niet gaan we met een for-loop door alle andere posities die deelbaar zijn door k+index en worden dan false gemaakt.

K wordt met 1 verhoogt en de loop begint opnieuw. Na de loop wordt het aantal priemgetallen berekent tussen 1 en 49

Nadat alle threads klaar zijn tellen we het aantal priemgetallen van elke thread bijelkaar op.

Reflectie

Bij het maken van deze opdracht gingen er vaak wat dingen minder soepel dan gehoopt. Voornamelijk bij het compleet begrijpen hoe mpi4py werkt en de documentatie ervan begrijpen. Maar het ontwerpt van multi-threading ging nog wel goed, zo had ik voor ik eraan begon al een goed idee hoe ik het wou aanpakken, en het met een klein beetje hoofdrekenen kunnen bewijzen dat het ontwerp werkt.

Nadat alle benodigdheden voor het werken met mpi4py goed gecodeerd was en het tijd was om het ontwerpt zelf te implementeren ging het al veel sneller en kon ik goed progressie maken. Ik vind dat het ontwerp en concept het sterkste kant is van mijn opdracht. Maar de volgende keer zal ik meer test programma’s maken om een beter begrip te verkrijgen over het werken met deze library.