**Conceptueel ontwerp**

In de vorige twee opdrachten heb ik leren werken met OpenMP en Open MPI. In deze opdracht ga ik een beschrijven hoe je deze twee libaries kan combineren. Open MPI kan verschillende processen op verschillende nodes parallel laten uitvoeren. En OpenMP kan alleen verschillende processen parallel laten lopen op dezelfde node. Dus om de kracht van deze twee libaries te combineren beschrijf ik hieronder hoe ik dat wil gaan doen.

Omdat Open MPI, processen op verschillende nodes parallel kan laten uitvoeren. Hebben we eerst meerdere nodes nodig. Elke node krijgt dan een MPI-proces en voert deze vervolgens uit. Als een node zijn proces berekend heeft moet deze node de uitkomst hiervan verzenden naar de Master node. De Master node is een van tevoren gekozen node die alle antwoorden van de overige nodes ontvangt en samenvoegt tot een eind antwoord.

Om de nodes ook hun MPI-proces parallel te laten uitvoeren maken we gebruiken van OpenMP. OpenMP laat het MPI-proces wat de node heeft gekregen parallel uitvoeren over de beschikbare cores in de node. En ook hier is dan een Master core die de antwoorden van verzamelt van de overige cores en die samenvoegt tot een eindantwoord, die dan weer wordt terug gegeven aan het MPI-proces.

Tot slot, word het proces op deze methode vele male versneller doordat het op een dubbele manier geparallelliseerd wordt, zowel op de server/node als op de processor/core.

A picture containing drawing

Description automatically generatedTe verduidelijking heb ik hier onder een ontwerp getekend.

**Hoe dit toe te passen voor de** **Zeef van Eratosthenes**

Dit ga ik als volgt toepassen om de ZeefvanEratosthenes parallel te laten lopen met mpi4py. Ik zal hier ook beschrijven hoe ik pymp hiervoor zal gebruiken maar ik pas dit niet toe omdat pymp geen fijne libarie is om mee te werken.

Eerst zal ik de master node de zeef laten aanmaken op met een gegeven maximum. Daarna zal ik met behulp van mpi4py de aangemaakt zeef opsplitsen over de beschikbare nodes, hierbij geef ik ook gelijk aan wat het eerste getal is van het begin van een sub-zeef voor elke node. Zodat niet elke node bij 0 begint met checken of een getal een priemgetal is.

Ver volgens zal dan met behulp van pymp het proces parallel laten uitvoeren op de node. Als dit gedaan is en de node berekend heeft van de sub-zeef wat priemgetallen zijn en wat niet, dan wordt de sub-zeef weer samengevoegd op de master node. Deze eindzeef daarvan wordt dan gesommeerd en de uitkomst daarvan zijn dan het aantal priemgetallen dat voorkomt tot en met het gegeven maximum.

**Analyse**

Na het schijven van de parallelle versie van de Zeef van Eratosthenes heb ik deze getest met een verschillend aantal, en de limiet van de zeef van 1000000. Hieronder zie je een overzicht van de runtime tijden. De blauwe lijn geeft aan hoeveel tijd er nodig is per aantal cores voor het uitvoeren van de code. De rode lijn geeft aan hoelang de non-parallelle versie deed over het uitvoeren van de code. In deze grafiek vallen een aantal dingen op. Tot en met ongeveer 8 cores heb je binnen de parallelle versie heel veel tijdwinst. Maar daarna is er nauwelijks meer tijdwinst. Dit komt omdat bij dit aantal cores de communicatie tussen de cores meer performance van de computer vraagt dan dat het oplevert. Wat mij nog meer opvalt is dat de sequentiële versie in alle gevallen sneller is dan de parallelle versie. Dit is denk ik het resultaat van alle extra stappen die in neem om de parallelle versie parallel te maken. De stappen zoals het her distribueren van de zeef over de verschillende cores kan een intensieve taak zijn voor de computer, omdat er in die stap veel communicatie tussen de cores plaats vindt.

**A screenshot of a social media post

Description automatically generated**

**Zelfreflectie**

Om te beginnen zal ik eerst beschrijven wat ik lastig vond aan deze opdracht. Een van de dingen die ik lastig vond aan deze opdracht was het algoritme daadwerkelijk sneller laten uitvoeren met meer cores. Bij het bedenken van het ontwerp voor deze opdracht kwam ik steeds tot de conclusie dat de verschillende cores op elkaar moesten wachten zodat er geen tijdwinst mogelijk is. Het is mij uiteindelijk wel gelukt om de parallelle versie sneller te maken door de data van de zeef te verspreiden over de verschillende cores.

Verder vind ik dat ik deze opdracht voldoende gemaakt heb en ik ben blij dat het mij gelukt is de parallelle versie sneller te laten draaien bij meer cores. Wat bij deze opdracht minder ging was het op tijd inleveren, ik ben ergens halverwege de lesweken een week gaan achterlopen. Dit resulteerde erin dat ik deze opdracht verder moest afmaken in de projectweken en periode D. Hierdoor had ik die weken een extra werkdruk. Ik zou de volgende keer er dus voor moeten zorgen dat ik mijn opdrachten op tijd inlever.