Sarah Goossens & Axel Willekens

Abstract

In deze labosessie wordt een cluster opgesteld.   
Vervolgens wordt de uitvoersnelheid van een operatie met en zonder cluster beschouwd.

Clusters

Computerarchitectuur

Inhoud

[Inleiding 2](#_Toc9449223)

[Probleemstelling 2](#_Toc9449224)

[Oplossing 2](#_Toc9449225)

[Opstellen cluster 2](#_Toc9449226)

[Programma 2](#_Toc9449227)

[Besluit 3](#_Toc9449228)

[Bijlage A – code parallel programma 5](#_Toc9449229)

[Bijlage B – code serieel programma 6](#_Toc9449230)

# Inleiding

Om de rekentijd te verkorten wordt vandaag de dag de rekenkracht van verschillende processoren dikwijls gebundeld. Hierdoor kunnen threads of processen parallel worden uitgevoerd wat een positieve impact heeft op de Million Instructions Per Seconds (MIPS) rate. Om dit te verwezenlijken kunnen meerdere processoren op één chip worden geplaatst, een andere manier is om verschillende computers te laten samenwerken over een netwerk.

Dit labo werd verwacht dat we een cluster opstelden met behulp van verschillende Raspberry Pi’s. Een cluster is een groep van verbonden computersystemen die tezamen werken als een rekeneenheid. Op deze manier wordt de illusie gewekt dat men maar op één machine bezig is.

We baseerden ons op de tutorial terug te vinden op de onderstaande URL:

# Probleemstelling

Hoe kunnen zware rekenkundige taken toch in een aanvaardbare tijdsspanne worden uitgevoerd?

# Oplossing

Om zware berekeningen sneller uit te voeren kan gebruik worden gemaakt van clusters. Door het werk de verdelen over meerdere processoren en het proces parallel uit te voeren kan een taak sneller worden afgerond. In deze labosessie baseerden we ons op de tutorial terug te vinden in de volgende link: *https://www.instructables.com/id/How-to-Make-a-Raspberry-Pi-SuperComputer/.*

## Opstellen cluster

Om een cluster op te bouwen uit Raspberry Pi (RPI)’s dienen eerst de RPI’s correct geconfigureerd te worden. Als besturingssysteem werd gekozen voor Raspbian. Hierop moeten de juiste instellingen worden gekozen. Voor het gebruiksgemak kiest men best een unieke hostname op het netwerk, dit vereenvoudigt het gebruik van *ssh*. De cluster zal als background netwerk gebruik maken van het ethernetnetwerk in het lokaal.

Na het installeren en configureren van het OS dient een Message Passing Interface (MPI) te worden geïnstalleerd. MPI is een standaard voor een softwarebibliotheek die communicatie tussen processen vereenvoudigt en zo helpt bij het programmeren van parallelle computers. Ook de python module mpi4pi wordt geïnstalleerd, het parallelle programma zal worden geschreven in python.

Wanneer één RPI volledig is geconfigureerd wordt de *image file* ervan gebrand op andere SD-kaartjes voor de andere RPI’s in de cluster.

Om de MPI tussen de verschillende RPI’s op te stellen dient gewerkt te worden *rsa-keys.* Door de *rsa-keys* van **PI02** en **PI03**aan **PI01** toe te voegen aan de *authorized\_keys file* en de *rsa-key* van **PI01** toe te voegen aan de *authorized\_keys file* van **PI02** en **PI03** kan *ssh* naar de RPI’s onderling worden verwezenlijkt zonder dat hier authenticatie met behulp van een wachtwoord vereist is.

Na deze stap is de MPI tussen de verschillende RPI’s opgesteld en kan een parallel programma worden geschreven.

## Programma

**Parallel programma**

Nadat de cluster was opgesteld, kon het parallelle programma worden geschreven. In het programma zal het gemiddelde worden genomen van een reeks getalwaarden. De code is terug te vinden in Bijlage A en staat op elke RPI die onderdeel uitmaakt van de cluster.

Nadat de nodige modules zijn geïmporteerd worden enkele variabelen gedeclareerd voor de MPI. Hieronder worden de belangrijkste hiervan besproken:

* Het object *comm* stelt de communicatielink voor.
* De variabele *size* houdt bij hoeveel computersystemen (hier RPI’s) er in de cluster zitten.
* De variabele *rank* onderscheidt de RPI’s onderling. Zo heeft de master steeds een *rank* die nul is.
* De variabele *aantal* wordt verder gebruikt in het programma en bepaalt de lengte van de array waarvan het gemiddelde zal moeten worden genomen.

Is de huidige processor van de RPI de master (rank = 0), dan wordt de timer nodig voor de benchmark gestart en in de variabele a wordt een array van aantal elementen aangemaakt. Deze elementen zijn random gegenereerd binnen een interval . Merk op dat de methode np.random.random\_sample() een waarde teruggeeft binnen een Gausscurve met het gemiddelde op 0,5 binnen een interval . Om deze reden verwacht men hier op een groot aantal waarden een gemiddelde dat rond de 500 ligt.

Om deze berekening uit te voeren, verdelen we de array a (die enkel op de master terug te vinden is) in drie gelijke delen. Aan elke processing unit geven we een deel. Deze dient hiervan het gemiddelde te bereken. Vervolgens moeten deze resultaten terug bij de master belanden en wordt het gemiddelde van deze tussenresultaten bepaalt.

De functie comm.Scatter(a,local\_a,root=0) zorgt ervoor dat aan elke RPI een deel van de array a (die enkel op de master terug te vinden is) wordt uitgedeeld. Hiervoor is er eerst op elke RPI een lege array local\_a aangemaakt die aantal\_per\_pi lang is. Door communicatie tussen de RPI’s onderling wordt ervoor gezorgd dat in de variabele local\_a op elke RPI een ander deel komt van de volledige array a. Na deze verdeling wordt parallel op elke RPI afzonderlijk het gemiddelde van de array local\_a berekend: gem = np.array(np.mean(local\_a)).

Op een analoge wijze worden deze drie parallel berekende gemiddelden van elke een deel van de array a terug verzameld op de master. Hiervoor wordt enkel op de master een array recv\_buf = np.empty([size, 1]) aangemaakt, waarin de berekende tussenresultaten worden verzameld met de methode comm.Gather(gem, recv\_buf ,root=0). Het totale gemiddelde is vervolgens het gemiddelde van deze (in dit geval drie) tussenresultaten: totaalgem = np.mean(recv\_buf).

Enkel op de master wordt de timer op te benchmarken gestopt en worden de betreffende gegevens afgedrukt.

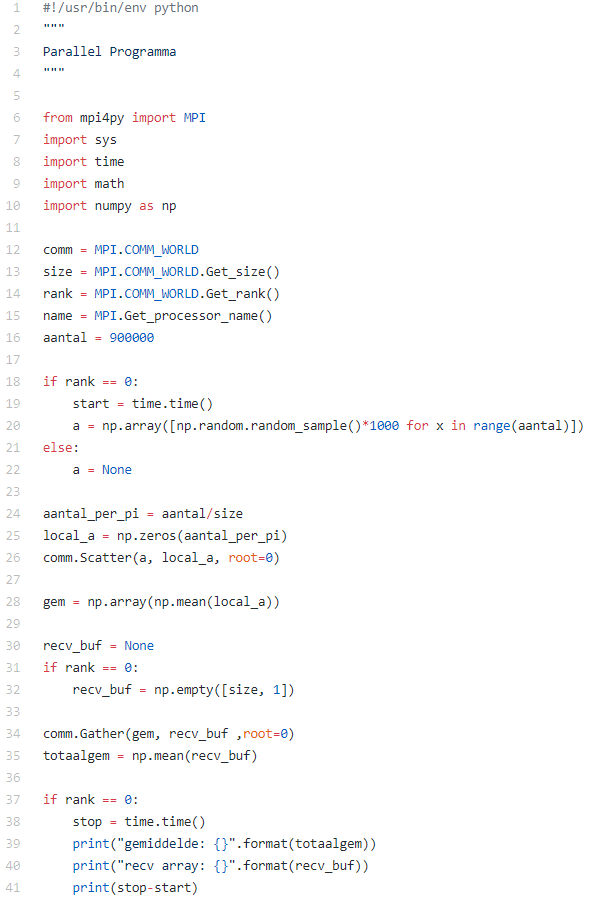
**Serieel programma**

Om een vergelijking te maken wordt de berekening van het gemiddelde ook serieel uitgevoerd en gebenchmarkt.

# Besluit

Uit de resultaten blijkt dat het seriële programma hier toch sneller uitvoert dan het parallelle programma. Dit komt omdat de berekening, namelijk het bepalen van het gemiddelde, een te weinig zware berekening is. Hierdoor weegt de communicatie van de gegevens zwaarder door dan de tijdswinst die gemaakt wordt door parallelle berekening. Zwaardere rekenkundige bewerkingen zijn dus nodig vooraleer clusters tijdswinst opleveren.

# Bijlage A – code parallel programma



# Bijlage B – code serieel programma

