Sieve of Eratosthenes

Reinier van Leeuwen

AI V2A

Ontwerp

MPI (MPI4PY)

Om het programma over verschillende processoren/processen te verdelen is de module MPI4PY gebruikt. Omdat elk proces hetzelfde script draait, moet er aan de hand van de process id (rank) een gedeelte van het werk toegewezen worden.

Elke proces krijgt een range toegewezen van getallen. Elke range is even groot en het laatste proces krijgt de rest van N / aantal processen.

Rank 0 berekend elke iteratie in de while loop een nieuwe (hogere) K en geeft deze door aan alle overig processen. Er wordt vervolgens met deze K gekeken of een van de getallen in de toegewezen range deelbaar is met K. Als dit het geval is wordt dit getal of niet-priem gezet.

Door steeds K te verhogen wordt bij elke iteratie de te doorlopen range kleiner en de runtime korter.

Om het in stappen op te sommen:

- Wijs een range toe aan elk proces
- Laat elk proces de deelbare getallen wegstrepen in een eigen lijst met de meest recente K
- Laat rank 0 de beste K zoeken en geef deze door aan alle andere processen
- Voeg alle lokale proces lijsten samen met een gather functie
- Extraheer de resultaten

OPENMP (PYMP)

Binnen elk proces kan de toegewezen range ook nog in threads verwerkt worden. Elke proces doorloopt de toegewezen range van getallen om niet-primes weg te strepen. Dit wegstrepen gebeurd in een simpele for loop met een niet veranderende K.

Ik heb een PYMP geïmplementeerd in een apart bestand. Deze implementatie zorgt echter voor alleen maar tijdsverlies. Daarnaast is de library zo slecht dat de code die je ermee schrijft niet bepaald logisch/intuïtief wordt. Dit heeft vooral de maken met de kleine selectie bruikbare data types (en bijbehorende functies).

Ik vraag mij sowieso af hoeveel nut het heeft om zowel over processoren als threads te verdelen. Misschien zijn mijn computers te oud om dit goed te kunnen benutten.

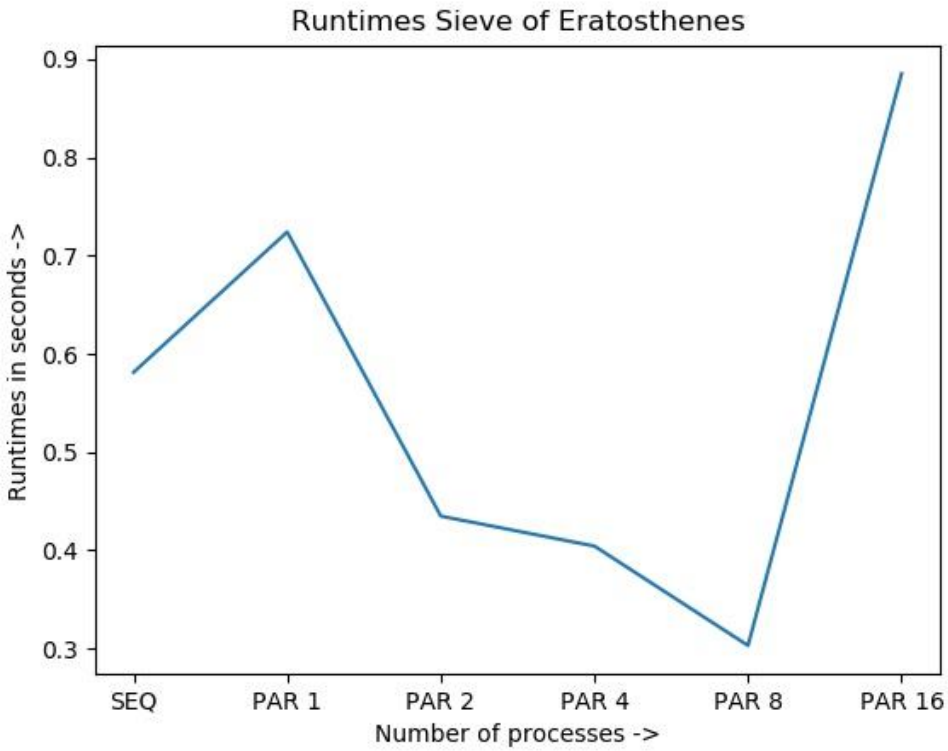
Runtimes

Ik heb een wat oudere 8-core CPU gebruikt voor het testen, waardoor ik een maximale N van 10 miljoen heb gebruikt. Ik ben begonnen vanaf 100000 omdat kleine lijsten door opzetten MPI sowieso geen verbetering gaan opleveren. Ik ben gestopt met hoger aantal cores meten toen de runtime opliep.

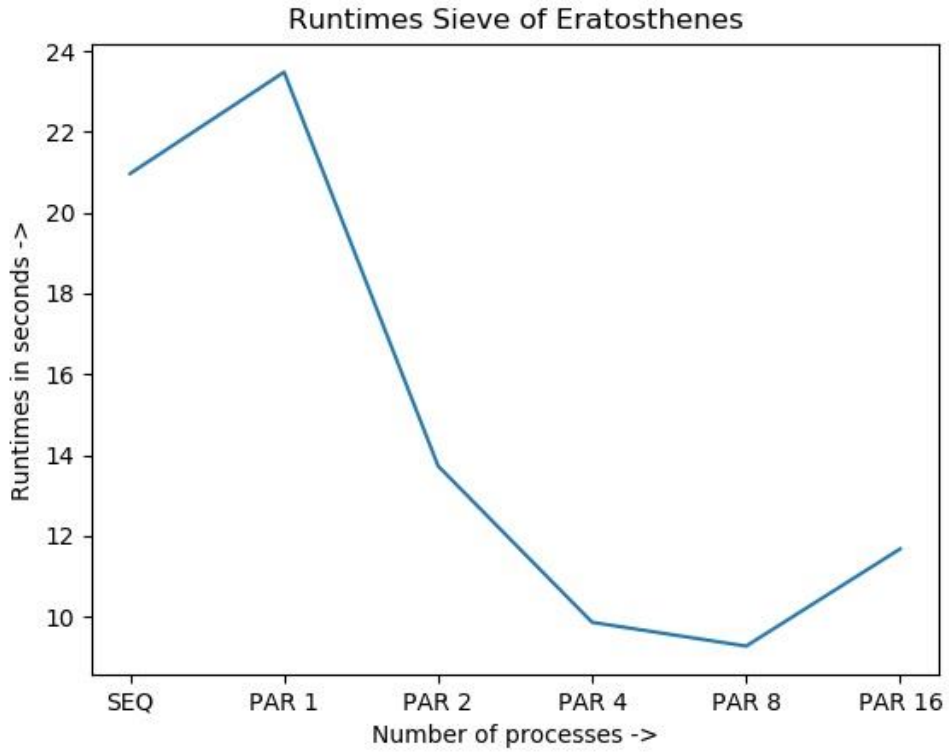
Ik merkte ook dat de runtimes erg afhankelijk zijn van andere openstaande processen. Zelfs al waren alle (zichtbare) omstandigheden gelijk aan elkaar, kwamen er steeds andere tijden uit. Hieronder is in een tabel een van de runs te zien die ik gedaan heb. Onder de tabel is een grafiek getekend per N.

	SEQUENTIAL	PARALLEL 1 CORE(S)	PARALLEL 2 CORE(S)	PARALLEL 4 CORE(S)	PARALLEL 8 CORE(S)	PARALLEL 16 CORE(S)
100000	0.58134	0.72418	0.43484	0.40433	0.30308	0.88535
1000000	20.96852	23.48197	13.725178	9.85413	9.26566	11.67370
10000000	574.22539	716.24521	418.00950	316.41230	312.063257	316.33694

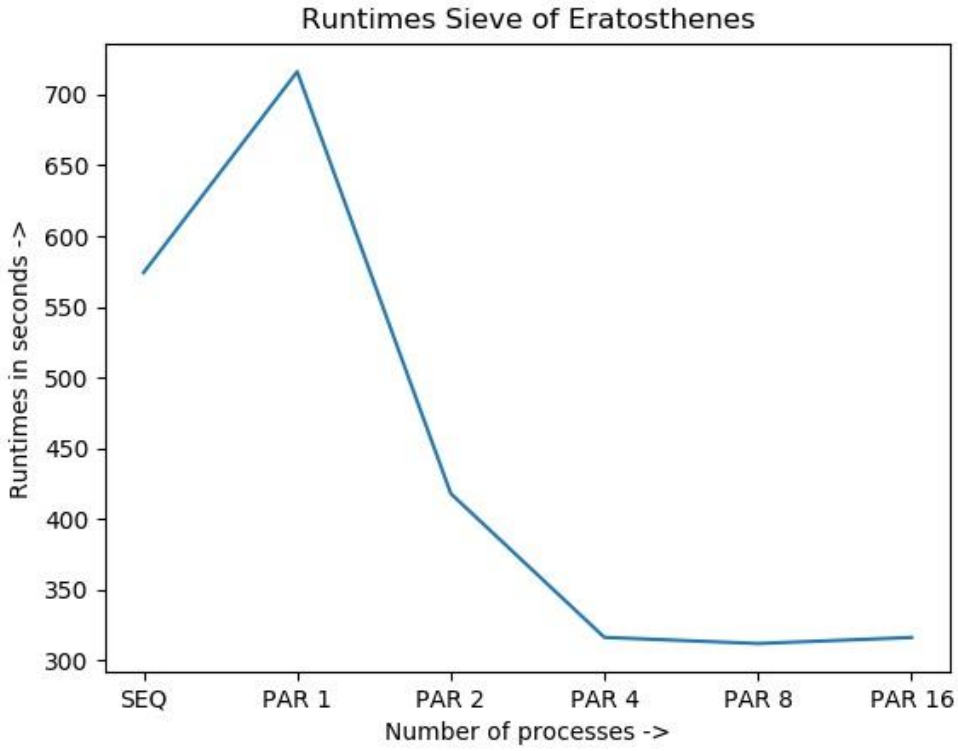
100000:



1000000:



10000000:



Toelichting resultaten

Zoals in de bovenstaande grafieken te zien is, is het verloop voor elke N priemgetallen enigszins hetzelfde.

Zoals de verwachten is de runtime met gebruik van 1 proces net iets hoger dan de sequentiële variant. Dit komt omdat het algoritme hetzelfde is, maar de MPI module wel het een en ander aan extra werk moet uitvoeren.

Vervolgens loopt de runtime af als het aantal processen oploopt. Er kan tijdswinst behaald worden tot het aantal fysieke processoren bereikt is. De computer waarmee is de testen heb uitgevoerd heeft 8 fysieke processoren en zoals te zien is, loopt de tijd alleen maar op bij het uitvoeren op meer dan 8 processoren.

Zoals bij runtimes al vermeld is het resultaat erg verschillend per run. Daarnaast zijn de resultaten ook erg computer afhankelijk. Ik zag dat nieuwere computers soms de tijd werkelijk halveren als het aantal processoren verdubbeld werd. Bij mijn computer is het resultaat elke keer weer (compleet) anders.

Reflectie

De opdracht met het gekozen algoritme vond ik erg leuk. Het schrijven van het algoritme was niet al te moeilijk, wat het parallelliseren makkelijker maakte (niet per se makkelijk). Het moeilijkste vond ik het denken in processen. Omdat elk proces hetzelfde script draait moet je op een hele andere manier denken dan je gewend bent met programmeren. Ik heb hier wel veel van geleerd en zal een volgende keer dit veel sneller kunnen.

Uiteindelijk denk ik dat de opdracht best goed gelukt is, nadat ik om een aantal tips had gevraagd. Misschien zou ik nog wat meer tijd kunnen besteden aan de ruimte complexiteit maar dat was niet de opdracht. Stiekem heb ik dit al gedaan want in het werkelijke algoritme worden steeds niet-primes weggestreept en ik heb dit om een andere manier uitgewerkt met data type set.

Ik had misschien wat eerder om hulp/tips moeten vragen, dat had mij veel tijd gescheeld… Ik liep steeds tegen mijn missende kennis van de MPI module aan, en nam ook niet te tijd om mij beter in te lezen. Daarnaast was het algoritme op zoveel manieren uit te werken dat ik soms even de draad kwijt raakte met welke variant ik aan het werk was.