Rapport på Oblig4 i In3030

1. Introduction – what this report is about
2. User guide – how to run your program (short, but essential), include a very simple example.
3. Parallel Radix sort – how you did the parallelization – consider including drawings
4. Implementation – a somewhat detailed description of how your Java program works & how tested
5. Measurements – includes discussion, tables, graphs of speedups, number of cores used

1.

Rapport på Oblig 4 om Parallel Radix med variabelt antall sifre.

2.

Programmet kompileres med:

Javac Oblig4.java	Dette krever at Oblig4.java og Oblig4Precode.java er i samme mappe.

Programmet kjøres med :

Java Oblig4 <N> <nTester>	N er antall tall i arrayen, nTester er hvor mange ganger programmet skal kjøre.

3.

Forklaring på paralliseringen fra steg A til steg D:

Steg A → Finn max verdi: La hver tråd finne en localMax i sin del av a[], og deretter bruke en synkronisert metode updateMax() til å oppdatere en felles max. Til slutt, synk med de andre trådene ved bruk av en cyclic barrier.

Et mellomsteg: Alle trådene skal utføre den samme algoritme for å dele opp bittene jevnt mellom sifrene, basert på hvor mange bits den max verdien har, samt med hvor mange sifre vi skal sortere på.

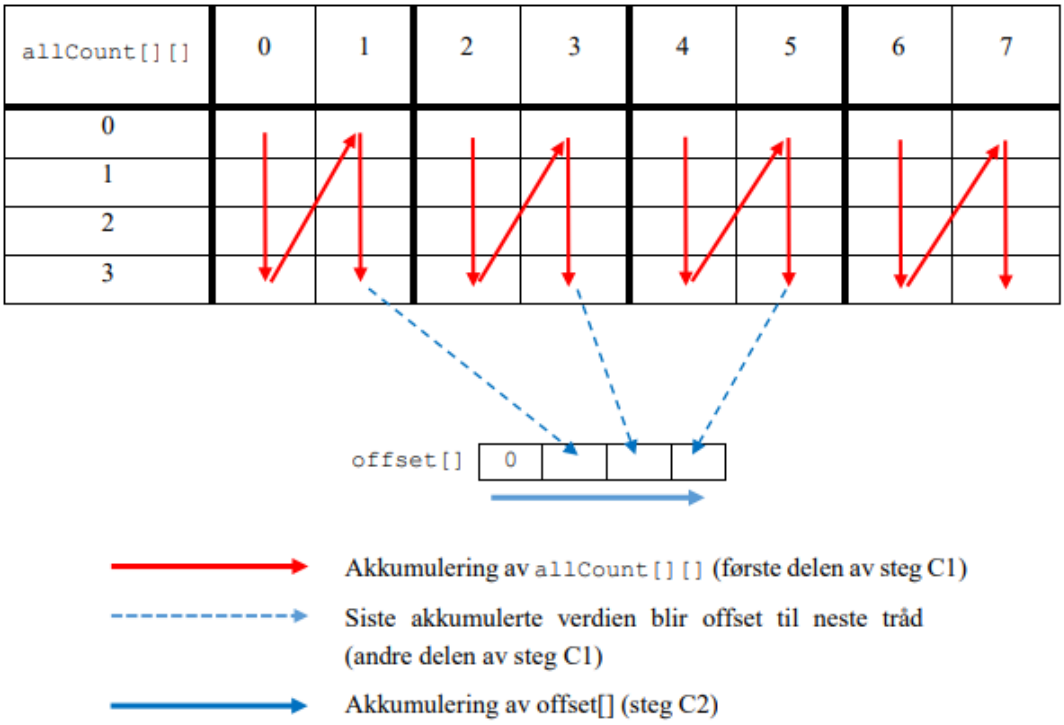
Steg B → La hver tråd telle opp hvor mange det er av hvert sifferverdi i sin del av a[]:
- Hver tråd skal lagre resultatene i en lokal count[], og deretter la den globale tabellen allCount[] peke til count[] slik at andre trådene kan også se på resultatene.
- Synk ved bruk av cyclic barrier.

Steg C → Oppdatere allCount[][] slik at denne tabellen skal få riktige akkumulerte verdiene på slutten:
- C1: La hver tråd akkumulere allCount[][] i sin del slik at den siste akkumulerte verdien blir offset til den neste tråden. Når dette er ferdig, synk ved bruk av cyclic barrier.

- C2: La bare den første tråden akkumulere verdiene i offset[] arrayen, mens andre trådene går direkte til neste synkronisering barrier. Dette trenger vi ikke å parallellisere fordi offset[] er bare en liten array (med nthreads plasser), og da går det fortere sekvensielt.
- C3: La hver tråd legge til offset til hver verdi i sin del av allCount[][]. Synk med andre trådene når ferdig.

Steg D → Hver tråd skal få én og bare én rad i tabellen allCount[][] (tråd-0 får rad-0, tråd-1 får rad-1, osv…). Hver tråd skal da gå gjennom sin del av a[] fra venstre til høyre, og flytte tallene fra a[] til b[] basert på innholdet i raden
som tråden har fått. Synk med de andre trådene når alt er ferdig.

- Antall synkroniseringer (ved bruk av en cyclic barrier) vi må gjøre i steg A er bare 1; mens for steg B-D, trenger vi å synkronisere alle trådene 5 ganger. I tillegg må vi gjenta steg B-D for alle sifrene. For eksempel, hvis vi har n_s sifre, så er antall synkroniseringer lik 1 + 5n_s. Dette betyr at en mindre n_s gir et mindre tidsbruk siden vi har færre synkroniseringer for å gjøre. Men vi må også huske at en mindre n_s leder til mer bits per siffer, og dette kan senke effektiviteten. Da er det best å kunne regne ut optimale verdier for n_s og antall bits per siffer slik at vi får en bra tradeoff. Men dette er veldig vanskelig å si fordi det avhenger av hvilken array a[] vi får.
Jeg har tegnet en liten illustrasjon over hvordan steg C ser ut. Her antar jeg at det er 3 bits i sifret (som da gir 23 = 8 mulige sifferverdier), og at det er 4 tråder i datamaskinen. Med andre ord, allCount[][] skal være en tabell med 8 kolonner og 4 rader, hvor hver tråd tar 2 av de kolonnene.



4.

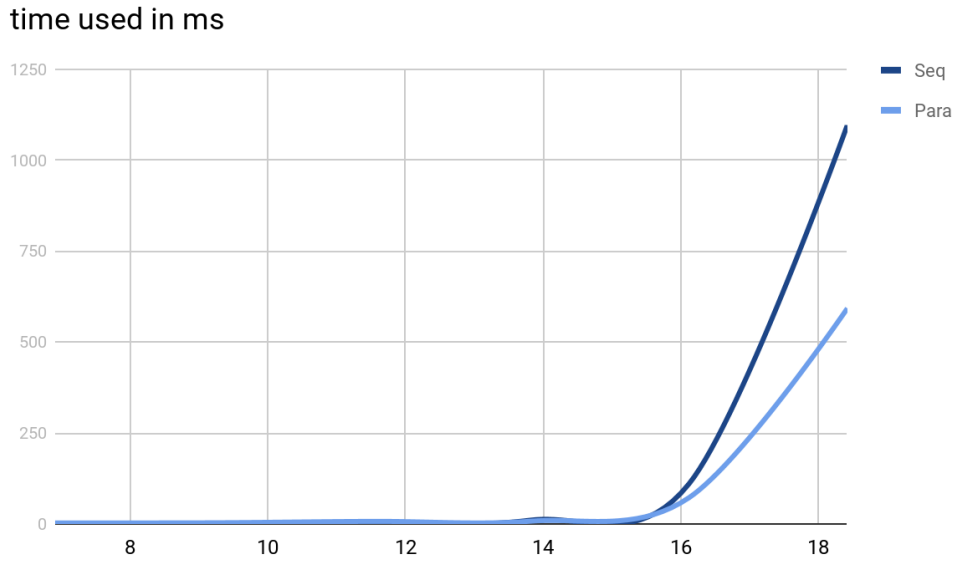
For en bedre forklaring se kommentarene i koden Oblig4.java.
Sekvensielle algoritmen for å radix sortere arrayen a[] er slik:
- Steg A: Finn det største tallet i a[]
- Steg B: Tell hvor mange det er av hvert sifferverdi og skriv dette i count[]
- Steg C: Akkumulerer verdiene i count[] slik at vi får "pekere" til b[]
- Steg D: Flytt tallene fra a[] til b[] basert på de pekere vi fikk fra steg C
- Gjenta steg B-D for alle sifrene

Den parallelle biten er forklart i 3.

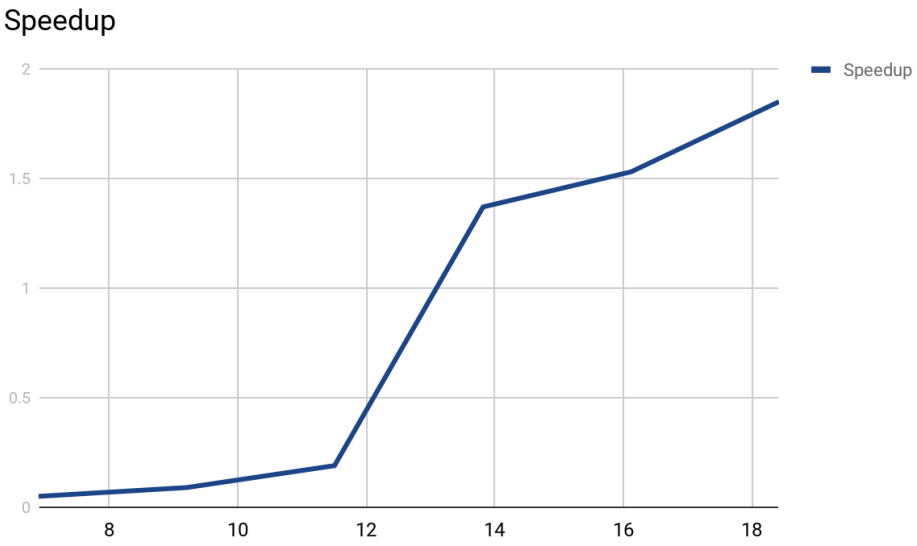
5.

Kjørt på pc med 8 Intel(R) Core(™) i7-6700HQ CPU @ 2.60GHz.

N	ln(N)	Time seq i ms	Time para i ms	Speedup
100 000 000	18.42	1096.965174	592.875296	1,85
10 000 000	16.12	110.553328	72.283683	1,53
1 000 000	13.82	10.461239	7.62746	1,37
100 000	11.51	1.362964	7.178274	0,19
10 000	9.21	0.286024	3.314964	0,09
1 000	6.91	0.149333	2.923063	0,05



Grafen er tid brukt i millisekunder delt ut over ln(N)



Grafen er speedup delt ut over ln(N)