Sorteernetwerken van Optimale Grootte

Mathias Dekempeneer, Vincent Derkinderen
Bachelor Informatica
Katholieke Universiteit Leuven
voornaam.achternaam@student.kuleuven.be

Abstract

Korte samenvatting van wat we doen en wat de conclusie is.
Verder werken op paper van Codish et al. Sorteer optimal size sorting network.
Tijdsverbetering van x? Bla
Bla
Bla
Bla
Bla
Bla
Bla
Bla
Bla
Bla

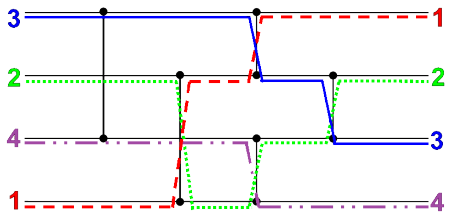
1 Introductie

Situering + bijdrage.
Sorting Network (high level), Optimal Size (high level), contributies andere papers rond deze twee, enkele getallen rond grootte orde van het probleem, wat er al geprobeerd is (SAT, generate & prune,...), hoe wij het probleem zullen aanpakken (hoe wij prunen (high level)), gebruikte hardware...
Bla
Bla
Bla
Bla
Bla
Bla
Bla
Bla
Bla
Bla
Bla
Bla
Bla
Bla
Bla
Bla

2 Probleemstelling

Een *comparator netwerk* C_k^n bestaat uit n kanalen en k *comparatoren*. Een comparator (i, j) verbindt twee verschillende kanalen i en j waarbij $0 < i < j \leq n$. We nemen x_l^m als waarde op kanaal m net voor comparator l. De l^{de} comparator vergelijkt de huidige waarden van beide kanalen en plaatst de kleinste waarde op kanaal i en de grootste waarde op kanaal j zodat $x_{l+1}^i = \min(x_l^i, x_l^j)$ en $x_{l+1}^j = \max(x_l^i, x_l^j)$. De uitvoer van een comparator netwerk verwijst naar de partieel geordende vector $\vec{x} = \{x_{k+1}^1 \dots x_{k+1}^n\}$.

Een *sorteernetwerk* is een comparator netwerk met als eigenschap dat de uitvoer gesorteerd is ongeacht de invoer. Een sorteernetwerk C_k^n van optimale grootte houdt in dat er geen ander sorteernetwerk C_l^n bestaat waarbij $l < k$. Figuur 1 is een voorbeeld van zo een netwerk waarop ook de werking gedemonstreerd wordt. Deze figuur toont ook twee parallelle comparatoren $(1, 2)$ en $(3, 4)$, comparatoren die geen kanaal gemeenschappelijk hebben en van volgorde omgewisseld kunnen worden. Om te onderzoeken of een comparator



Figuur 1: Sorteernetwerk 4 kanalen, 5 comparatoren

netwerk een sorteernetwerk is, kunnen we gebruik maken van het *nul - één principe*. Dit principe, zoals beschreven volgens Knuth [Knuth, 1973], stelt dat wanneer een comparator netwerk met n kanalen alle 2^n mogelijke sequenties van n 0- en 1-en sorteert, het een sorteernetwerk is. De optimale grootte van een sorteernetwerk met n kanalen is reeds bewezen tot en met $n \leq 10$ (Tabel 1 [Codish *et al.*, 2014]). Voor $n > 10$ zijn er bovengrenzen gekend door zowel concrete voorbeelden als de systematische constructie van Batcher [Batcher, 1968]. De ondergrenzen werden gevonden via bewijzen en lemma 1 [Voorhis, 1972].

Lemma 1. $S(n+1) \geq S(n) + \lceil \log_2(n) \rceil, \forall n \geq 1$

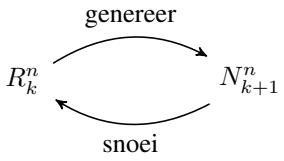
n	6	7	8	9	10	11	12
bovengrens	12	16	19	25	29	35	39
ondergrens	12	16	19	25	29	33	37

Tabel 1: Minimaal aantal comparatoren bij $6 \leq n \leq 12$ kanalen.

3 Voorgestelde oplossing

Om te bewijzen dat een sorteernetwerk C_k^n een sorteernetwerk is van optimale grootte, moeten we bewijzen dat er geen sorteernetwerk C_{k-1}^n bestaat. n kanalen zorgen voor $\frac{n(n-1)}{2}$ verschillende comparatoren. Wanneer een netwerk k comparatoren bevat, kunnen er $\binom{n(n-1)}{2}^k$ verschillende netwerken gevormd worden. Voor 9 kanalen en 24 comparatoren betekent dit 2.245×10^{37} verschillende netwerken, dit maakt het overlopen van alle netwerken niet aantrekkelijk. Door gebruik te maken van symmetrieën willen we snoeien in het aanmaken van deze netwerken.

We gebruiken de *genereer- en snoei-methode* zoals beschreven door Codish *et al.* (sectie 3, [Codish *et al.*, 2014]). Deze methode heeft een cyclisch verloop waarbij men bij elke cyclus de set R_k^n uitbreidt naar N_{k+1}^n om vervolgens te snoeien en de set R_{k+1}^n te bekomen (Figuur 2). Specifiek



Figuur 2: Genereer en snoei principe

zullen we vertrekken van een netwerk zonder comparatoren om te eindigen bij R_k^n bestaande uit één sorteernetwerk van optimale grootte. Bij de genereer-stap zullen we aan elk netwerk van R_k^n alle mogelijke comparatoren toevoegen zodat $|N_{k+1}^n| = |R_k^n| \times \frac{n(n-1)}{2}$. Bij de snoei-stap zullen we dan netwerken verwijderen volgens het subsumes principe beschreven in definitie 1.

Definitie 1 (Subsumes)**.** *We zeggen "Comparator netwerk $C_{k,a}^n$ subsumes comparator netwerk $C_{k,b}^n$" wanneer een permuntatie π bestaat zodat $\pi\left(outputs\left(C_a\right)\right) \subseteq outputs\left(C_b\right)$. Dit wordt genoteerd als $C_a \preceq C_b$ om aan te duiden dat er een permutatie π bestaat zodat $C_a \leq_\pi C_b$.*

Lemma 2. *Wanneer voor comparator netwerk $C_{k,a}^n, C_{k,b}^n$ geldt dat $C_a \preceq C_b$ en er bestaat een sorteernetwerk $C_b; C$[1] van grootte m dan bestaat er ook een sorteernetwerk $C_a; C'$ van grootte m.*

Concreet kunnen we de definitie van subsumes en lemma 2 beschreven door Codish *et al.* [Codish *et al.*, 2014] gebruiken om in te zien dat we netwerken die gesubsumed worden door andere netwerken kunnen verwijderen. Wanneer een set van netwerken een sorteernetwerk bevat, zal het snoeien van deze set resulteren in het bekomen van het sorteernetwerk. Dit kan gebruikt worden om de eindigheid van het algoritme aan te tonen.

Het overlopen van alle permutaties om na te gaan of er een permutatie π bestaat zodat $\pi\left(Outputs\left(C_a\right)\right) \subseteq Outputs\left(C_b\right)$, en dus $C_a \preceq C_b$, is een kostelijke bewerking. Om deze bewerkingen te vermijden en te versnellen, zullen we extra methoden moeten invoeren om snellere beslissingen te maken over het "subsumen van een ander netwerk".

3.1 Representatie van comparator netwerken

Bij de representatie van comparator netwerken moeten we rekening houden met geheugengebruik en de mogelijkheid om efficiënte bewerkingen te kunnen uitvoeren. Concreet zullen we comparatoren voorstellen door een sequentie van bits, waarbij twee bits op één staan. Bijvoorbeeld [010010] stelt de comparator $(2, 5)$ voor bij een netwerk van 6 kanalen. Om de hoeveelheid overbodige bits te beperken, zullen we bij de `Java` implementatie gebruik maken van shorts[2]. Buiten de comparatoren worden ook de outputs van het netwerk bijgehouden, opgedeeld per aantal 1'en. In de `Java` implementatie kiezen we er voor om een comparator netwerk voor te stellen door een tweedimensionale array van shorts, short[][], en laten we de rij van n 1'en weg. Een voorbeeld van zo een representatie staat in tabel 2.

Comparators	[0011]	[1010]	
Outputs één 1	[0001]	[0100]	[0010]
Outputs twee 1'en	[0011]	[0101]	[0110]
Outputs drie 1'en	[0111]	[1011]	

Tabel 2: Representatie C_2^4: $(1, 2)(2, 4)$

3.2 Genereren

Bij de genereer-stap lopen we over de set R_k^n en voegen we bij elk netwerk alle mogelijke comparatoren toe. Aangezien een netwerk dat wordt uitgebreid met een overbodige comparator, één waarbij de outputs ongewijzigd blijven, gesubsumed zal worden door een uitbreiding van dat netwerk met een niet overbodige comparator, kunnen we deze meteen verwijderen uit de set N_{k+1}^n. Alvorens deze beslissing te maken door alle outputs te overlopen, kunnen we ook eerst kijken of de comparator gelijk is aan de vorige in het netwerk. Wanneer 2 netwerken op de volgorde van hun parallelle comparatoren na gelijk zijn zoals in figuur 3a en 3b zullen deze elkaar subsumen en één van de twee verwijderd worden. Dit kan reeds bij de generatie-stap gemakkelijk opgevangen worden door bij het toevoegen van een nieuwe comparator x na te gaan of x een kanaal gemeenschappelijk heeft met de vorige comparator (Code 1).

Code 1: Test op parallelle comparatoren

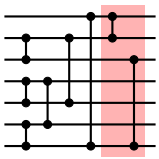
```
x & vorigeComp != 0
```

Wanneer dit niet het geval is en het dus parallelle comparatoren zijn, kunnen we bijvoorbeeld kiezen om het netwerk

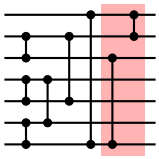
[1] $C_b; C$ is een concatenatie van netwerk C_b en C.

[2] In `Java` bestaat een short uit 16 bits.

weg te gooien waarbij de comparator kleiner is dan de vorige comparator.



(a) Netwerk 1



(b) Netwerk 2

Figuur 3

Uitleg hoe we de generate doen.
Redundant (of de comparator die je toevoegt, wel iets verandert? Uitleggen wat wij specifiek doen), unique_if uitleggen

3.3 Snoeien

Uitleg hoe we de prune doen.
Checken van alle netwerken met alle netwerken voor de prune stap.

- Aantal 1en bij $C_a > C_b \Rightarrow C_a$ subsumes niet C_b
- $|W(C_a, x, k)| > |W(C_b, x, k)| \Rightarrow C_a$ subsumes niet C_b
- Uitleggen reductie van permutaties

3.4 Parallellisatie

Parallellisatie uitleggen
Uitleg hoe generate and prune verandert door elke thread zijn stuk te laten generate en prunen en vervolgens in een groter geheel te prunen en hoe dit groter geheel prunen werkt zonder locks.

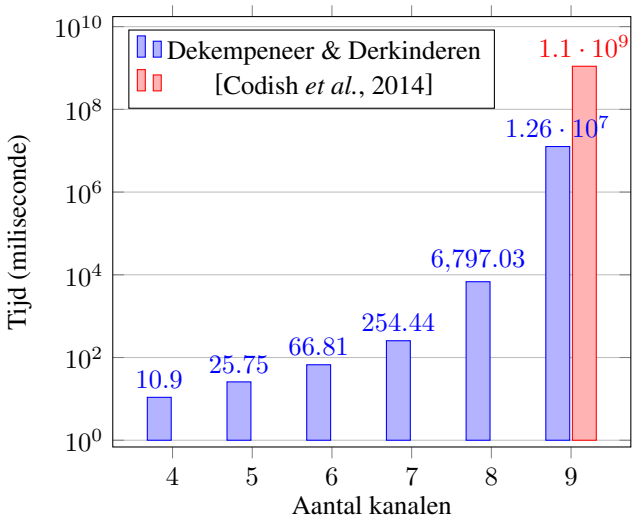
4 Evaluatie

Empirische evaluaties + grafiekjes
Tabel geven van hoeveel beslissingen er op welke plaats genomen worden.

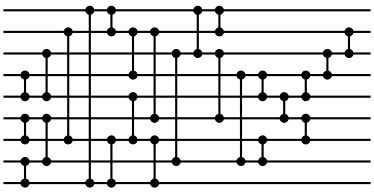
Vergelijken runtime voor 9 kanalen met Codish.
Schatting runtime voor 10 kanalen.

5 Conclusies

Conclusie[Codish *et al.*, 2014]
Conclusie van wat er bereikt is en hoe er verder aan gewerkt kan worden.[Codish *et al.*, 2015]



Figuur 4: Resultaten



Figuur 5: Sorteernetwerk 9 kanalen, 25 comparatoren

Erkenning

De rekeninfrastructuur en dienstverlening gebruikt in dit werk, werd voorzien door het VSC (Vlaams Supercomputer Centrum), gefinancierd door het FWO en de Vlaamse regering - departement EWI.
Professor Dr. Ir. Tom Schrijvers, Katholieke Universiteit Leuven.

Referenties

[Batcher, 1968] K. E. Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), pages 307–314, New York, NY, USA, 1968. ACM.

[Codish *et al.*, 2014] Michael Codish, Luis Cruz-Filipe, Michael Frank, and Peter Schneider-Kamp. Twenty-five comparators is optimal when sorting nine inputs (and twenty-nine for ten). Technical report, IEEE International Conference on Tools with Artificial Intelligence (ICTAI), November 2014.

[Codish *et al.*, 2015] Michael Codish, Luis Cruz-Filipe, and Peter Schneider-Kamp. Sorting networks: the end game. In *Proceedings of the 9th International Conference on Language and Automata Theory and Applications, LATA*, LNCS, 2015.

[Knuth, 1973] D. E. Knuth. *The art of computer programming. Vol.3: Sorting and searching*. 1973.

[Voorhis, 1972] David C. Voorhis. *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department*, chapter Toward a Lower Bound for Sorting Networks, pages 119–129. Springer US, Boston, MA, 1972.