Data structures for quantum circuit verification and how to compare them

Vinkhuijzen, L.T.

Samenvatting

Kwantumcomputers zijn een voorgesteld, nieuw type computer. Het doel is om sommige berekeningen veel sneller uit te voeren dan nu mogelijk is door gebruik te maken van de unieke fenomenen die kwantumdeeltjes laten zien: superpositie en verstrengeling. Als en wanneer grote kwantumcomputers gebouwd worden, beloven ze een aantal problemen veel sneller te kunnen oplossen, met name in de natuurkunde en scheikunde, waarmee ze potentieel bijdragen aan belangrijke toepassingen zoals betere medicijnen, batterijen en zonnepanelen. Zo dragen ze bij aan de transistie naar groene energie en een duurzame economie. Om deze beloftes waar te maken, hebben we gereedschappen nodig die deze computers en de algoritmes die erop draaien kunnen analyseren, simuleren, compileren en verifiëren. Tegelijkertijd blijven we natuurlijk gereedschappen nodig hebben om klassieke (d.w.z. niet-kwantum) software te compileren, analyseren en verifiëren. Het doel van dit proefschrift is om bij te dragen aan gereedschap voor deze twee toepassingen.

Voordat we onze bijdragen beschrijven, beschrijven we eerst welk probleem we precies trachten aan te pakken, en wat haar zo moeilijk maakt.

- Een algoritme voor een kwantumcomputer bestaat uit een aantal qubits en zogehete *quantum logic gates* die op de qubits worden uitgevoerd. Wanneer we een dergelijk algoritme analyseren, willen we vragen beantwoorden zoals, "Wat is de meest waarschijnlijke output?" en "Hoe waarschijnlijk is het dat de computer in een bepaalde toestand terecht komt?"

  De grootste uitdaging ligt erin dat de toestand van een kwantumcomputer wordt beschreven door een exponentieel lange lijst getallen: Een computer met een n aantal qubits vereist een lijst van 2^n complexe getallen. Deze exponentiele groei maakt het onmogelijk om deze lijst op te slaan zonder gebruik te maken van

compressie, wanneer n groot genoeg is.

- Wanneer we een klassiek stuk software willen analyseren, willen we weten of het voldoet aan zijn specificatie. We willen bijvoorbeeld verifiëren dat een programma niet dezelfde resource uitgeeft aan twee verschillende threads. Een uitdaging is dat het aantal mogelijke toestanden waarin het programma zich kan bevinden groeit als 2^n – opnieuw exponentieel in het aantal bits.

In beide gevallen zal een stuk gereedschap slagen het systeem te analyseren wanneer het erin slaagt om te gaan met deze exponentieel grote objecten. Omdat de uitdagingen in het kwantum- en het klassieke geval zo op elkaar lijken, mogen we ons in dit proefschrift toeleggen op deze gedeelde uitdaging. Een krachtig stuk gereedschap kan vervolgens waarde opleveren voor beide toepassingen.

Het gereedschap waar wij ons in dit proefschrift in specializeren heet een *decision diagram* (DD, of beslissingsdiagram). Decision diagrams zijn een datastructuur met een rijke theorie en met veelvuldige toepassingen in verschillende delen van de computerwetenschappen, waaronder het analyzeren van klassieke- en kwantumcomputers.

We houden deze samenvatting te bondig voor een uitgebreide uitleg van hoe decision diagrams werken. In plaats daarvan sommen we kort op wat hen zo aantrekkelijk maakt voor onze doeleinden. Om te beginnen zijn DDs een vorm van verliesvrije datacompressie: een DD representeert een lijst getallen maar gebruikt, onder gunstige omstandigheden, veel minder geheugenruimte (dit is analoog aan zeggen dat een Boolese formule een gecomprimeerde vorm van zijn waarheidstabel is). Wat voor ons van belang is, en wat DDs onderscheidt van andere compressiemethoden, is dat we operaties op de gecomprimeerde data kunnen uitvoeren zonder deze eerst te decomprimeren.

Bijvoorbeeld, de volgende situatie komt een aantal keer voor in dit proefschrift. Stel dat we een kwantumalgoritme aan het analyseren zijn en we hebben een beschrijving van de huidige toestand van de qubits; we noemen deze toestand $|\varphi\rangle$. Nu willen we weten in welke toestand de qubits zich zullen bevinden nadat de volgende *quantum logic gate* is toegepast; deze nieuwe toestand noemen we $|\psi\rangle$. Als we een beschrijving van $|\varphi\rangle$ hebben in de vorm van een lijst van 2^n getallen, dan is het relatief eenvoudig om ook zo'n beschrijving voor $|\psi\rangle$ te vinden. Helaas kost dit exponentieel veel tijd; op deze manier zullen we dus geen grote systemen kunnen analyseren. Daarom gebruiken we niet een lijst maar een decision diagram om de toestand te beschrijven. Om nu

dezelfde quantum logic gate toe te passen gebruiken we een algoritme dat als input $|\varphi\rangle$ neemt en als output $|\psi\rangle$ geeft, beide in de vorm van een DD. Dit algoritme is vaak efficiënt omdat de DD van nieuwe toestand $|\psi\rangle$ gebouwd kan worden zonder dat de (kleine) DD van de oude toestand $|\varphi\rangle$ gedecomprimeerd hoeft te worden. Het ontwerpen van zulke algoritmes en de DDs waar ze op werken is het doel van dit proefschrift.

De methode hierboven stelt ons in staat om *in principe* elk kwantumcircuit te analyseren; echter *in de praktijk* zal de DD vaak veel te groot worden om in het geheugen van een computer te passen, of de algoritmes nemen teveel tijd in beslag. Wanneer dat gebeurt, is onze analyse mislukt. Hoe groot een DD wordt, en hoe lang de operaties duren, hangt namelijk af van de kwantumtoestand en van het type DD dat gebruikt wordt. Het begrijpen en verhelpen van deze beperkingen door bestaande DDs en hun algoritmes te analyseren en nieuwe te ontwerpen is daarom het primaïre technische doel van dit proefschrift.

Bestaande DDs konden veel verschillende kwantumalgoritmes niet effectief analyseren. Onder andere zogeheten *stabilizer circuits* vielen buiten de boot, ondanks dat andere methoden deze circuits wél aankonden. In dit proefschrift presenteren we daarom een nieuw type DD, die we de *Local Invertible Map Decision Diagram* (LIMDD) noemen, die alles kan dat voorgaande DDs konden, plus stabilizer circuits, en nog meer. Door ons ontwerp op papier te analyseren en vervolgens in software te implementeren, kunnen we laten zien dat ons nieuwe DD bestaande DDs zowel op papier als empirisch overtreft, in Hoofdstuk 3 en Hoofdstuk 4.

Er zijn veel verschillende ontwerpen voor decision diagrams. Dat komt zowel doordat nieuwe architecturen hebben geleerd van voorgaanden, en omdat verschillende ontwerpen verschillende toepassingen bedienen. De plus- en minpunten van deze DDs worden gemeten met drie criteria: (1) *succinctness* (beknoptheid: hoe groot wordt de DD?); (2) *tractability* (welke operaties op de DD zijn efficiënt, d.w.z. kosten slechts polynomiaal veel tijd); en (3) *rapidity* (hoe snel is één DD ten op zichte van een andere?). Wij stuitten echter op de bevindingen dat (1) bestaande methodes voor simulatie van kwantumcircuits nog niet op deze criteria in kaart waren gebracht; en (2) er geen simpele methode bestond om te bepalen welke van twee DDs de meer *rapid* is. In hoofdstuk 5 van dit proefschrift (1) beoordelen we een aantal veelgebruikte datastructuren en (2) geven we een nieuwe methode om eenvoudig de meer *rapid* van twee datastructuren aan te wijzen. Al doende ontdekken we een verrassend verband

tussen twee welbekende datastructuren die uit verschillende onderzoeksvelden komen, namelijk dat QMDDs een speciaal type matrix product state (MPS) zijn; een gevolg is dat MPS meer *rapid* zijn dan QMDDs. Voorheen waren deze twee methoden nog niet vergeleken; ons werk verbindt dus twee onderzoeksrichtingen.

In hoofdstuk 6 bekijken we een oud DD genaamd de *Disjoint Support Decomposition Binary Decision Diagram* (DSDBDD). Dit stuk gereedschap komt niet uit de kwantumwereld; het wordt gebruikt voor klassieke model checking. Hoewel de DSDBDD reeds een softwareimplementatie heeft, was nog weinig bekend over hoe krachtig hij was. Wij vullen deze kennis aan door te laten zien dat hij exponentieel meer *succinct* is dan enkele andere DDs.

Ten slotte gebruiken we in hoofdstuk 7 *Sentential Decision Diagrams* (SDD) voor klassieke model checking. In dit probleem beschouwen we een computerprogramma en worden we gevraagd welke toestanden bereikbaar zijn vanuit de begintoestand. Ons doel is om een SDD te bouwen die de verzameling bereikbare toestanden weergeeft. De grootste uitdaging is om de *variabelenboom* (vtree) goed te kiezen, omdat die bepaalt hoe groot de SDD wordt, en dus of deze past in het geheugen van een computer (en dus of de analyse lukt of niet). Hiertoe hebben we de gelegenheid om het inputprogramma te analyseren voordat we beginnen met simuleren. We kunnen dus opsommen welke variabelen interactie met elkaar hebben en zo beslissen welke variabelen dus idealiter dicht bij elkaar in de variabelenboom moeten komen. We presenteren de eerste heuristieken om een variabelenboom te kiezen en vergelijken empirisch hoe effectief deze verschillende heuristieken enkele systemen analyseren. De resultaten wijzen erop dat SDDs compacter maar trager zijn dan BDDs.