Embedded computersystemen explOratie

Pimentel, A.D.

Publication date
2023
Document Version
Final published version

Embedded computersystemen expl*Oratie*

Andy D. Pimentel

Mijnheer de rector magnificus, mijnheer de decaan, zeer gewaardeerde toehoorders,

Computers bevinden zich overal om ons heen en zijn niet meer weg te denken uit ons bestaan. Sterker nog, onze samenleving is compleet afhankelijk van het gebruik van computers en de netwerken die deze apparaten met elkaar verbinden. Dit is ook goed te merken wanneer er iets fout gaat in een van deze computersystemen. We kennen natuurlijk allemaal wel het gevoel dat ons bekruipt wanneer thuis het Internet even niet werkt of dat de treinen niet rijden vanwege een computerstoring. En dat zijn dan nog vrij onschuldige voorbeelden die hoogstens ergernis veroorzaken. Het moge duidelijk zijn dat er ook meer extreme gevallen denkbaar zijn, die bijvoorbeeld te maken hebben met vitale infrastructuur, waarbij grote delen van onze samenleving zelfs stil kunnen komen te liggen wanneer de onderliggende computersystemen niet goed werken.

Veel van de computersystemen om ons heen zijn duidelijk zichtbaar, zoals de laptop en desktop computers die wij tegenwoordig allemaal gebruiken. Maar nog veel meer zijn minder zichtbaar of zelfs compleet onzichtbaar. Goede voorbeelden hiervan zijn bijvoorbeeld het computersysteem in je mobiele telefoon, in auto's, treinen, vliegtuigen, 'smart home' apparaten in je huis, en ga zo maar door. De lijst is werkelijk ellenlang. Deze minder zichtbare computersystemen, die ingebed zitten in een ander product, noemen we daarom ook ingebedde computersystemen. Over het algemeen wordt hiervoor toch de Engelse term gebruikt: Embedded Computersystemen. Het zal geen verrassing zijn dat het volume van de markt voor dit soort embedded computersystemen vele malen groter is dan die voor laptop, desktop en server computersystemen bij elkaar. Zoals de titel van mijn oratie laat zien, zal ik mij in mijn rede voornamelijk richten op dit belangrijke domein van embedded computersystemen.

Als we embedded computersystemen vergelijken met computers voor algemeen gebruik, zoals laptops en desktop computers, dan zien we een aantal dingen. Allereerst, gebruikers van computers voor algemeen gebruik zijn eigenlijk maar geïnteresseerd in één ding: snelheid. Een dergelijke computer moet vele verschillende software applicaties, van simpele tekstverwerkers, geavanceerde games tot wetenschappelijke software, snel kunnen uitvoeren. Voor embedded computersystemen is snelheid natuurlijk ook van belang. Echter, er is een verschil. Bij embedded computersystemen is het vaak belangrijk om bepaalde garanties te kunnen geven over de snelheid waarmee ze reageren. In sommige gevallen kan dit namelijk zelfs het verschil betekenen tussen leven en dood. Neem bijvoorbeeld het geval van een simpele airbag in een auto. Deze moet precies op tijd, en niet te vroeg of te laat, opblazen in het geval van een ongeluk. Of een zelfrijdende auto, die natuurlijk op tijd moet reageren op gevaarlijke situaties op de weg.

Maar buiten de eisen op het gebied van snelheid, hebben embedded computersystemen te maken met nog veel meer eisen. Ik zal er een aantal noemen. Zo zijn veel embedded systemen afhankelijk van batterijen voor hun stroomtoevoer en kan er in deze systemen geen gebruik worden gemaakt van zogenaamde actieve koeling met behulp van een ventilator. Dit betekent dat het energieverbruik zo laag mogelijk dient te zijn om de batterij langer mee te laten gaan, of om het systeem niet te warm te laten worden. Verder dienen embedded computersystemen uitermate betrouwbaar te zijn. Het vastlopen van je televisie is vervelend, maar nog niet dramatisch. Het wordt echter een ander verhaal wanneer je te maken hebt met

veiligheids- en bedrijfskritische computersystemen zoals die in auto's, vliegtuigen, maar ook in de eerdergenoemde vitale infrastructuur zoals het elektriciteitsnet of kerncentrales. Naast deze ontwerpeisen op het gebied van snelheid, energieverbruik en betrouwbaarheid komt er vaak nog een hele reeks van additionele eisen om de hoek kijken met betrekking tot, onder andere, de veiligheid, kosten, en het onderhoud van de embedded computersystemen.

Het probleem van al deze ontwerpeisen is dat ze vaak conflicterend zijn. Het is namelijk onmogelijk om aan alle ontwerpeisen tegelijkertijd te voldoen. Om een voorbeeld te geven: Het verbeteren van de snelheid of betrouwbaarheid van een computersysteem gaat over het algemeen ten koste van een hoger energieverbruik en maakt het systeem duurder. Als ontwerper van dergelijke systemen zul je dus een goed compromis moeten vinden tussen al deze ontwerpeisen. Dit betekent dat een ontwerper in staat moet zijn om al in een vroeg stadium van het ontwerp inzicht te verkrijgen in de consequenties van verschillende ontwerpkeuzes op de uiteindelijke snelheid, energieverbruik, betrouwbaarheid, enzovoort, van het systeem. Voorbeelden van ontwerpkeuzes zijn, onder andere, het type en aantal processoren die gebruikt worden in het computersysteem, de hoeveelheid geheugen, of het type netwerk waarmee de processoren en het geheugen met elkaar verbonden zijn. De ontwerper heeft dus hulp nodig, in de vorm van technieken en software tools, die het mogelijk maken om efficiënt een groot aantal van deze ontwerpkeuzes te bestuderen en te evalueren. Dit proces noemen we *ontwerp exploratie*.

We zijn nu hopelijk op het punt aangekomen dat u de titel van mijn oratie enigszins kunt plaatsen. In het vervolg van mijn rede neem ik u mee op een korte reis door dit onderzoeksdomein.

Ontwerp exploratie is een optimalisatie proces waarin meerdere doelstellingen, zoals snelheid en energieverbruik van het systeem, tegelijkertijd geanalyseerd én geoptimaliseerd dienen te worden. Dit proces bestaat uit twee belangrijke onderdelen. Ten eerste, de *analyse* van het zogenaamd extra-functionele gedrag van een specifieke implementatie van het computersysteem. Met dit extra-functioneel gedrag bedoel ik de snelheid, het energieverbruik, de betrouwbaarheid, enzovoort, waarmee het systeem een bepaalde taak vervult. Hiervoor is een computermodel nodig die dit extra-functionele gedrag van een specifieke implementatie van het computersysteem, dus met inachtneming van bepaalde ontwerpkeuzes, kan voorspellen en kwantificeren. Het tweede onderdeel van ontwerp exploratie is de *optimalisatie* van het computersysteem met betrekking tot de eerdergenoemde extra-functionele eigenschappen. Oftewel, we willen de ontwerper de middelen geven om de juiste ontwerpkeuzes te maken die ertoe leiden dat het uiteindelijk computersysteem qua snelheid, energieverbruik, betrouwbaarheid, enzovoort, zo optimaal mogelijk is. Hiervoor is een computeralgoritme nodig die efficiënt en effectief door alle mogelijke ontwerp opties kan zoeken en een combinatie van ontwerpbeslissingen oplevert die tot het beste computersysteem leidt.

Deze twee hoofdingrediënten van ontwerp exploratie, namelijk de analyse van extra-functioneel systeemgedrag en de optimalisatie hiervan, vormen de kern van mijn onderzoek. Ik zal beide aspecten in het vervolg van mijn rede nader toelichten met een aantal voorbeelden van onderzoeksresultaten die we behaald hebben op deze gebieden. Ik begin hiervoor bij de analyse van extra-functioneel systeemgedrag. Zoals ik al eerder aangaf, is

hiervoor een computermodel nodig die het extra-functionele gedrag van een specifieke implementatie van een computersysteem kan voorspellen en kwantificeren. Een dergelijk computermodel kan min of meer op twee manieren gerealiseerd worden. Zogenaamde analytische modellen proberen het extra-functionele systeemgedrag te beschrijven door middel van wiskundige formules. Het voordeel van dergelijke modellen is dat ze vaak heel snel doorgerekend kunnen worden en dus snel een antwoord geven over hoe 'goed' een bepaalde systeem implementatie is volgens het model. Echter, het is heel lastig, en vaak zelfs onmogelijk, om deze modellen voldoende nauwkeurig te maken. Dit komt omdat de dynamiek van een computersysteem heel moeilijk te vatten is in deze analytische modellen. In de context van ontwerp exploratie, waarin de modellen gebruikt worden om ontwerpkeuzes te evalueren, zou deze onnauwkeurigheid echter kunnen leiden tot het maken van verkeerde keuzes. Daarom heeft mijn werk zich voornamelijk gericht op een alternatieve methode voor systeemmodellering, namelijk door middel van simulatie. Een simulatie probeert de werkelijkheid in software na te bootsen. Dit is vergelijkbaar met wat bijvoorbeeld gebeurt in een vliegtuigsimulator of, voor de Formule 1 liefhebbers onder ons, een simulator van een Formule 1 auto. Maar in ons geval simuleren we het gedrag van een computersysteem zelf met behulp van een andere computer. Met dergelijke simulatiemodellen kunnen we het systeemgedrag veel nauwkeuriger nabootsen, en met name ook de dynamiek in dit gedrag.

Toen ik iets meer dan 20 jaar geleden begon met mijn onderzoek op het gebied van simulatietechnieken voor embedded computersystemen was het algemeen geaccepteerd om dit soort simulaties zo gedetailleerd en dus zo nauwkeurig mogelijk uit te voeren. Het nadeel hiervan is dat zulke gedetailleerde simulaties heel veel rekenkracht vergen, en dus enorm langzaam zijn. Het gebruik van dit type simulatoren in de context van ontwerp exploratie is ongewenst. Dit omdat er heel veel simulaties uitgevoerd moeten worden om een groot aantal verschillende ontwerpkeuzes te evalueren en deze simulaties daarvoor simpelweg te langzaam zijn. Het compromis tussen nauwkeurigheid en snelheid van een simulatie, en de mogelijkheid om hier mee te spelen, kan echter ook benut worden. Zodoende hebben wij simulatietechnieken ontwikkeld waarin embedded computersystemen op een hoger abstractie niveau gemodelleerd en gesimuleerd kunnen worden en daarmee veel sneller zijn dan de eerdergenoemde simulaties. De uitdaging hierbij is natuurlijk om dergelijke hoog-niveau simulaties wel voldoende nauwkeurig te maken zodat ze nog steeds bruikbaar zijn in de context van ontwerp exploratie.

De technieken die we ontwikkeld hebben zijn gebaseerd op de zogenaamde 'separation of concerns' en Y-chart principes. Dit betekent dat we aparte modellen onderscheiden voor het modelleren van applicaties en het modelleren van de onderliggende hardware van het embedded computersysteem. Een applicatie model omvat het functionele gedrag van het systeem en onderscheidt de verschillende taken in een applicatie alsmede de interactie tussen deze taken. Het hardware model van het embedded computersysteem, aan de andere kant, beschrijft de aanwezige hardware componenten zoals de processoren en netwerken en hun extra-functionele eigenschappen. Oftewel, in dit model wordt bijvoorbeeld beschreven hoe snel en hoe energie efficiënt de verschillende hardware onderdelen bepaalde applicatie taken kunnen uitvoeren. Deze twee modellen voor applicaties en hardware worden vervolgens gekoppeld door middel van een derde model, het afbeeldingsmodel genaamd. Dit laatste model is een expliciete beschrijving van de afbeelding van applicatie taken naar hardware componenten. Deze afbeelding beschrijft dus welk hardware component welke applicatie taak

of taken uitvoert. Om nu een bepaald computersysteem te simuleren, bootst het applicatie model de activiteit van de applicatie na wat vervolgens, aan de hand van het afbeeldingsmodel, activiteit produceert in de verschillende componenten van het hardware model. Dit laatste model bootst uiteindelijk het extra-functionele gedrag (dus de snelheid, het energieverbruik, enzovoort) van deze activiteit na.

De drie genoemde modellen vormen in onze aanpak de hoofdingrediënten voor het simuleren van een specifieke implementatie van een computersysteem. Echter, doordat we de applicaties, onderliggende hardware en afbeelding van applicatie op hardware separaat modelleren, is deze simulatie techniek uitermate geschikt voor het uitvoeren van ontwerp exploratie. Zo kunnen we heel makkelijk wijzigingen maken in het hardware model, om bijvoorbeeld een andere hardware implementatie te evalueren, zonder dat dit effect heeft op de andere twee modellen. Of we kunnen heel makkelijk spelen met verschillende afbeeldingen van applicatie taken op de onderliggende hardware componenten door enkel en alleen het afbeeldingsmodel aan te passen.

Deze simulatiemethode stelt een ontwerper ook in staat om een computersysteem op meerdere abstractie niveaus te simuleren. Initieel opereren de applicatie en hardware modellen op een hoog abstractie niveau en abstraheren daarmee van veel onnodige details. Dit maakt de simulatie wellicht iets minder nauwkeurig maar wel razendsnel. Hiermee kunnen we efficiënt heel veel verschillende ontwerpkeuzes evalueren en zoeken naar zogenaamde sweetspots in de ontwerpruimte. Oftewel, hiermee proberen we heel snel verschillende implementaties van het computersysteem te vinden die er veelbelovend uitzien. Deze veel kleinere set van mogelijke implementaties kunnen we vervolgens in meer detail simuleren om de uiteindelijke ontwerpkeuzes te maken. Om simulatie op meerdere abstractie niveaus te faciliteren hebben we een reeks van technieken ontwikkeld die het, bijvoorbeeld, mogelijk maakt om de applicatie en hardware modellen gradueel te verfijnen door middel van toevoeging van meer en meer details.

Ik moet eerlijk toegeven dat de industrie in eerste instantie vrij sceptisch reageerde toen wij, en een aantal van onze collega onderzoeksgroepen, dit soort hoog-niveau simulatie methoden introduceerde. Zoals eerder gezegd, men vertrouwde in die tijd nog voornamelijk op gedetailleerde en dus langzame simulaties voor het maken van ontwerpkeuzes. Echter, deze gedetailleerde simulatoren kunnen niet of nauwelijks worden toegepast in de vroegste stadia van het ontwerp proces. En dit betekende dus dat men in een vroeg ontwerp stadium overgeleverd was aan de expertise en ervaring van de ontwerper zelf om goede ontwerp beslissingen te maken. Dit besef resulteerde toch al vrij snel in de adoptie van deze hoog-niveau simulatie technieken door grote bedrijven als Synopsys, Siemens en Intel, en deze worden tegenwoordig aangeboden als standaard onderdeel van hun portfolio van modellering en simulatie tools.

Met de beschikbaarheid van een hoog-niveau simulatie methode zoals ik die zojuist heb beschreven, kunnen we dus een groot aantal ontwerpkeuzes kwantitatief evalueren. Dat brengt mij op het tweede ingrediënt van ontwerp exploratie: de daadwerkelijke optimalisatie van een computersysteem met betrekking tot de extra-functionele eigenschappen. Deze optimalisatie betreft het automatisch zoeken naar een specifieke implementatie of configuratie van het computersysteem die qua snelheid, energieverbruik, betrouwbaarheid,

enzovoort, zo optimaal mogelijk is. Echter, de grote uitdaging hierbij is dat wat ontwerpkeuzes van een computersysteem betreft het aantal vrijheidsgraden en mogelijkheden enorm groot is. Dit betekent dat er een immens aantal verschillende systeem implementaties of configuraties doorzocht en geëvalueerd moet worden. Ter illustratie: voor een typische ontwerp exploratie van een embedded computersysteem is het aantal mogelijke systeem implementaties groter dan het aantal sterren in onze Melkweg. En in sommige gevallen kan dit zelfs oplopen naar een ontwerpruimte die groter is dan het aantal sterren in het volledige heelal. Het moge dan ook duidelijk zijn dat het praktisch gezien onhaalbaar is om alle mogelijke systeem implementaties één voor één ook werkelijk te evalueren door middel van simulaties.

Om dit probleem op te lossen maken we gebruik van computeralgoritmen die de gigantische zoekruimte op een slimme manier doorzoeken zodat er maar een relatief klein aantal systeem implementaties geëvalueerd hoeft te worden alvorens een goede oplossing, oftewel systeem implementatie, wordt gevonden. Dergelijke algoritmen kunnen echter niet garanderen dat ze de meest optimale oplossing vinden, maar ze vinden over het algemeen wel oplossingen die dicht bij het optimum zitten. Veel van dit soort zoekalgoritmen zijn geïnspireerd door de natuur en natuurlijke fenomenen. Goede voorbeelden van dergelijke algoritmen hebben aansprekende namen zoals mierenkolonie optimalisatie, particle swarm optimalisatie, simulated annealing, en genetische algoritmen. Binnen ons eigen onderzoek op het gebied van ontwerp exploratie maken we vaak gebruik van het laatstgenoemde type algoritmen: de genetische algoritmen. Dit eigenlijk om twee redenen: deze algoritmen hebben zich binnen het domein bewezen door goede resultaten te behalen, en ten tweede, deze algoritmen zijn bij uitstek geschikt om meerdere doelstellingen tegelijkertijd te optimaliseren. Het zogenaamde multi-objective optimalisatie.

Een genetisch algoritme, zoals de naam al doet vermoeden, is geïnspireerd op de evolutie zoals die in de natuur plaatsvindt. In het kort samengevat wordt binnen een genetisch algoritme een populatie van chromosomen bijgehouden die vervolgens geëvolueerd wordt over de tijd. In ons geval codeert elk chromosoom een specifieke implementatie van het embedded computersysteem. Dit gaat door middel van de reeks genen waaruit een chromosoom bestaat. Om een specifieke implementatie van een computersysteem te coderen, zal elk gen in het chromosoom een bepaalde waarde aannemen die een beslissing voor een specifieke ontwerpkeuze aanduidt. Om dit te illustreren nemen we een populaire vorm van ontwerp exploratie als voorbeeld, namelijk de exploratie van verschillende afbeeldingen van applicatie taken op de onderliggende hardware componenten. In dat geval zit er voor elke applicatie taak een gen in het chromosoom, en de waarde van deze genen geven aan op welke hardware component een specifieke taak is afgebeeld. Oftewel, de genen coderen welke applicatie taak door welk hardware component wordt uitgevoerd in een kandidaat systeemontwerp.

Elk van de kandidaat implementaties van het computersysteem die gecodeerd zijn als chromosomen in de populatie van het genetische algoritme kunnen door middel van simulatie geëvalueerd worden. Hiermee bepalen we dus hoe ‘goed’ deze verschillende kandidaat implementaties zijn. We selecteren vervolgens de beste van deze chromosomen en die gebruiken we voor het produceren van nakomelingen. Deze nakomelingen zijn het product van kleine genetische manipulaties, zoals mutaties, van de geselecteerde ouder

chromosomen. De nakomelingen, die de goede eigenschappen van de ouder chromosomen zo goed als mogelijk proberen te vererven, vormen dan de basis van een vernieuwde populatie. Dit proces van evaluatie, selectie, en productie van nakomelingen wordt herhaald om de populatie stapje voor stapje te verbeteren. Dit proces eindigt wanneer het convergeert naar een set van oplossingen die niet of nauwelijks meer te verbeteren is óf totdat een maximum aan iteraties is bereikt.

Deze genetische algoritmen hebben zich experimenteel bewezen en leveren over het algemeen goede resultaten op wanneer ze toegepast worden voor het uitvoeren van ontwerp exploratie. Dit exploratie proces blijft echter nog steeds heel rekenintensief en tijdrovend, met name als het aantal vrijheidsgraden in een ontwerp van een computersysteem groot is. We hebben daarom in de afgelopen jaren uitgebreid onderzoek gedaan naar technieken om dit exploratie proces verder te verbeteren zodat de goede kandidaat systeem implementaties sneller gevonden kunnen worden of om het mogelijk te maken om grotere zoekruimtes in korte tijd te kunnen doorzoeken. De technieken die we hiervoor bestudeerd hebben proberen óf de evaluatie van kandidaat ontwerpen sneller te maken óf het zoekproces naar nieuwe kandidaat ontwerpen te verbeteren.

Met betrekking tot het versnellen van de tijd die het kost om een kandidaat ontwerp te evalueren had ik al uitgelegd dat we dergelijke evaluaties vaak door middel van hoog-niveau simulatie doen. Deze simulaties zijn snel, maar als je ze bijvoorbeeld vergelijkt met de eerdergenoemde analytische modellen dan zijn de simulaties nog steeds meerdere ordes van grootte langzamer dan die analytische modellen. De vraag rijst dan ook in hoeverre je analytische modellen kunt gebruiken om te voorspellen hoe goed een bepaalde ontwerp kandidaat is. Ik had al eerder aangegeven dat analytische modellen beperkingen hebben in het modelleren van bepaalde vormen van systeem gedrag en daardoor fouten kunnen maken in hun voorspellingen. Echter, ze maken niet altijd fouten en dit gegeven kunnen we uitbuiten. Door bijvoorbeeld evaluaties met behulp van een analytisch model op een slimme manier af te wisselen met evaluaties door middel van simulatie kunnen we de foute voorspellingen die mogelijk door het analytische model worden gedaan ‘repareren’. Met behulp van experimenten hebben we kunnen aantonen dat deze hybride vorm van evaluatie van ontwerp kandidaten kan leiden tot een aanzienlijke versnelling van het exploratie proces zonder dat dit nadelige effecten heeft op de kwaliteit van de uiteindelijke ontwerp oplossingen.

Hiernaast kunnen we ook het zoekproces naar nieuwe kandidaat ontwerpen, dat dus bijvoorbeeld gebaseerd kan zijn op een genetisch algoritme, slimmer maken. Dit kan onder andere door domeinkennis toe te voegen aan het zoek algoritme om met behulp van deze kennis effectiever te kunnen zoeken. Ter illustratie nemen we opnieuw een genetisch algoritme als voorbeeld die naar goede afbeeldingen van applicatie taken op hardware componenten zoekt. We kunnen een dergelijk genetisch algoritme verrijken met domeinkennis in de vorm van een metriek die kwantitatief aangeeft hoe verschillend ontwerp kandidaten zijn. Hiermee kunnen we ervoor zorgen dat de populatie met ontwerp kandidaten binnen het genetische algoritme zo divers mogelijk blijft, en bijvoorbeeld geen zogenaamde ontwerp symmetrieëen bevat. Evaluatie van meerdere van deze ontwerp symmetrieëen is namelijk onnodig en daarmee ongewenst omdat ze exact hetzelfde extra-functionele gedrag tentoonspreiden. Tevens kan een hoge mate van diversiteit van de zoekpopulatie ervoor zorgen dat de zoektocht niet vastloopt in een bepaald gedeelte van de zoekruimte. Het

toepassen van dit soort vormen van domeinkennis kunnen het zoekalgoritme dus veel effectiever maken in het snel vinden van goede kandidaat ontwerpen.

Tot nu toe heb ik een paar keer gesproken over het voorbeeld van exploratie van de afbeelding van applicatie taken op de onderliggende hardware componenten. Als we klassieke embedded computersystemen beschouwen, dan implementeren deze systemen vaak maar één enkele vaste applicatie. Denk hierbij aan een eenvoudige koelkast, wasmachine, maar ook bijvoorbeeld een televisie van zo'n 15 jaar geleden. In de afgelopen jaren zijn veel embedded computersystemen echter steeds meer open systemen geworden, waarin het mogelijk is om nieuwe applicaties toe te voegen of te downloaden. Hiervan is natuurlijk een moderne smart TV een goed voorbeeld, maar ook mobiele telefoons en zelfs moderne auto's. Maar dit betekent dus wel dat dergelijke embedded computersystemen te maken hebben met een veel dynamischer applicatie gedrag waarbij de mengeling van applicaties die actief is drastisch kan veranderen over de tijd heen. Dit leidt tot de vraag hoe we ontwerp exploratie kunnen uitvoeren voor deze moderne embedded computersystemen waarin het applicatie gedrag dynamisch is.

In een naïeve aanpak zouden we het embedded computersysteem ontwerpen voor de worst case. Oftewel, we gaan er dan vanuit dat alle mogelijke applicaties tegelijkertijd op het systeem moeten kunnen executeren. Dit leidt over het algemeen tot een over-gedimensioneerd systeem omdat in werkelijkheid vrijwel nooit alle mogelijke applicaties tegelijkertijd actief zijn. Daarom hebben we onderzoek gedaan naar zogenaamde scenario-bewuste ontwerp exploratie. De term scenario refereert naar de applicatie scenario's die zich kunnen voordoen in het computersysteem. Met behulp van deze applicatie scenario's proberen we de mogelijke dynamiek in het applicatie gedrag te beschrijven. Zo kunnen deze scenario's bijvoorbeeld beschrijven welke mengeling van applicaties tegelijkertijd actief kunnen zijn. Als we dan terugkeren naar het onderwerp van ontwerp exploratie dan betekent dit dat we elke ontwerp kandidaat voor elk mogelijke applicatie scenario moeten evalueren, en dus simuleren, om een indicatie te krijgen hoe 'goed' het ontwerp is. Gegeven het grote aantal applicatie scenario's die zich kunnen voordoen, zou dit de ontwerp exploratie onnodig langzaam maken. Daarom hebben we technieken ontwikkeld die het mogelijk maken om uit de complete set van applicatie scenario's een kleine set scenario's te destilleren die representatief is voor de gehele set aan scenario's. Dit maakt het vervolgens mogelijk om tijdens de ontwerp exploratie elke kandidaat ontwerp slechts met behulp van deze kleine set van representatieve scenario's te evalueren. Als resultaat kunnen we hiermee een aanzienlijke reductie van het aantal simulaties behalen dat nodig is tijdens de ontwerp exploratie, wat natuurlijk zorgt voor een behoorlijke versnelling van het exploratie proces.

Ik heb het eerder gehad over de drie typen modellen die nodig zijn voor ontwerp exploratie, namelijk de modellen voor applicaties, de onderliggende hardware van het computersysteem, en de afbeelding van applicatie taken naar hardware componenten. Tot nu toe zijn we er eigenlijk vanuit gegaan dat met name de applicatie en hardware modellen met de hand, en dus door bijvoorbeeld de ontwerper zelf, gemaakt kunnen worden. En dit is ook het geval voor veel embedded computersystemen. Echter, er zijn ook voorbeelden van embedded computersystem te vinden die zo complex zijn dat de handmatige constructie van deze modellen niet meer mogelijk is. Zo werken wij bijvoorbeeld samen met het bedrijf ASML om te helpen met het verbeteren van hun producten. ASML is de wereldwijde marktleider in de

productie van machines die computerchips fabriceren. Deze machines maken gebruik van uitermate complexe embedded computersystemen waarin honderden processorkernen gedistribueerd door de hele machine aanwezig zijn en honderden software taken actief kunnen zijn.

Om dit soort embedded computersystemen te bestuderen en te optimaliseren is het maken van de benodigde modellen een enorme uitdaging. Dit kan eenvoudigweg niet met de hand. Hiervoor is het aantal software taken en hardware componenten in deze systemen simpelweg te groot. Daarom zijn we bijvoorbeeld op dit moment betrokken bij een project, samen met ASML en de Universiteit Leiden, waarin er, onder andere, gekeken wordt naar technieken om deze modellen volledig automatisch af te leiden en te genereren. Hiervoor maken we gebruik van, onder andere, monitoring gegevens die afkomstig zijn van de software en hardware in het computersysteem en die we opvangen tijdens het gebruik van het systeem.

Maar zelfs wanneer we het punt hebben bereikt waarop we de benodigde modellen geheel automatische kunnen produceren, en daar zijn we nog niet, dan levert het gebruik van deze modellen voor bijvoorbeeld ontwerp exploratie ons ook de nodige uitdagingen op. We hebben hier namelijk te maken met computersystemen waarvan het aantal ontwerpkeuzes, en dus de grootte van de ontwerpruimte, gigantisch is. Veel groter dan het aantal sterren in het heelal. Dit stelt de schaalbaarheid van de ontwerp exploratie technieken waarover ik het tot nu toe heb gehad enorm op de proef. We zullen daarom nieuwe technieken moeten blijven ontwikkelen om juist die schaalbaarheid verder te vergroten.

Nu zijn we op een punt aangekomen waarbij sommige toehoorders zich misschien afvragen: waarom heeft hij de term AI nog niet gebruikt? AI is tenslotte razend populair en er gaat op dit moment geen dag voorbij waarin AI niet in het nieuws is. Wanhoop niet, want ik ga het nu over AI hebben. Natuurlijk maken we binnen ons werk af en toe gebruik van AI technieken zoals machinaal leren om bijvoorbeeld specifieke aspecten van computersystemen te modelleren. Maar hier zal ik het niet over hebben. Ik wil namelijk van de mogelijkheid gebruik maken om iets verder in te gaan op het uitvoeren van moderne AI-applicaties op embedded computersystemen. Deze moderne AI-applicaties maken gebruik van zogenaamde deep learning modellen die gebaseerd zijn op diepe neurale netwerken. We kennen allemaal wel de voorbeelden van deze moderne AI-applicaties en hun impact: gezichtsherkenning, zelfrijdende auto's, smart assistants zoals Siri of Alexa, deep fakes, en natuurlijk ChatGPT. Met name die laatste heeft in de afgelopen tijd natuurlijk veel stof doen opwaaien, en u zult zich ongetwijfeld afvragen of ik deze oratie überhaupt wel zelf geschreven heb.

De neurale netwerk modellen, die het hart vormen van AI-applicaties, moeten eerst getraind worden voordat ze gebruikt kunnen worden. Dit is een enorm rekenintensief proces en gebeurt met behulp van krachtige computers. Maar ook het uitvoeren van de getrainde modellen vergt behoorlijk wat rekenkracht en veel computergeheugen. Daarom wordt het merendeel van de getrainde neurale netwerken tegenwoordig nog met behulp van Cloud servers uitgevoerd, ook als je een AI-applicatie bijvoorbeeld op je mobiele telefoon gebruikt. Een goed voorbeeld hiervan is het gebruik van Siri en Alexa: deze applicaties maken gebruik van neurale netwerken die in de Cloud uitgevoerd worden. Dit heeft echter nadelen. Het versturen en ontvangen van data naar en van die neurale netwerken in de Cloud kost tijd. Voor tijdkritische AI-applicaties vormt dit een probleem. Denk hier bijvoorbeeld aan een

zelfrijdende auto. Die moet snel, en dus lokaal, beslissingen kunnen nemen op basis van sensor data. Hier past langzame communicatie met de Cloud niet bij. Een ander probleem van het opsturen van data naar de Cloud voor het uitvoeren van neurale netwerken is privacy. Voor sommige data, zoals medische gegevens, is het simpelweg niet mogelijk dat ze naar Cloud servers worden opgestuurd vanwege privacy redenen.

Om de problemen omtrent tijdkristische applicaties en data privacy te adresseren is er een trend gaande naar wat we Edge AI noemen. In Edge AI proberen we de AI algoritmen, dus de neurale netwerken, zo dicht mogelijk bij de bron van de data uit te voeren. Dit betekent dat we de neurale netwerken zoveel mogelijk dicht bij de gebruiker op zogenaamde edge apparaten, zoals mobiele telefoons en andere Internet of Things apparaten, willen uitvoeren. Deze edge apparaten zijn over het algemeen embedded computersystemen met beperkte computationele en geheugen middelen, en hebben vaak een beperkt energiebudget omdat ze op batterijen werken. Dit maakt het uitvoeren van grote neurale netwerken op deze apparaten een enorme uitdaging. Vaak passen de ontworpen en getrainde neurale netwerken qua computationele en geheugen behoeften gewoon niet op deze systemen. Als gevolg hiervan zijn er dan meerdere handmatige, en dus kostbare, her-ontwerp cycli nodig voor de neurale netwerken voordat ze werkelijk passen en uitgevoerd kunnen worden op een edge apparaat.

Je raadt het misschien al, maar dit proces is ook te automatiseren door middel van ontwerp exploratie. In een dergelijke exploratie zoeken we automatisch naar een specifieke implementatie van een neuraal netwerk zodanig dat deze implementatie nauwkeurig genoeg werkt waarvoor het bestemd is, maar ook simpel genoeg is zodat het past op het edge apparaat. De technieken die we hiervoor ontwikkeld hebben zijn wederom gebaseerd op het gebruik van een genetisch algoritme om automatisch te zoeken naar goede kandidaat implementaties van het neurale netwerk. Hierbij proberen we vier eigenschappen tegelijkertijd te optimaliseren, namelijk de nauwkeurigheid, de doorvoersnelheid, het geheugen gebruik en het energieverbruik van een neurale netwerk implementatie. De laatste drie eigenschappen kunnen we snel voorspellen aan de hand van analytische modellen. Het snel bepalen van de nauwkeurigheid van een neuraal netwerk is echter een probleem: hiervoor dient het neurale netwerk eerst getraind te worden, wat een tijdrovende klus is. Het is onmogelijk om de tientallen tot honderden kandidaat implementaties van het neurale netwerk, die tijdens de exploratie geëvalueerd moeten worden, volledig te trainen. Daarom hebben we een techniek ontwikkeld, Piecemeal Training genaamd, die de kandidaat neurale netwerken stukje bij beetje traint. In elke iteratie van het genetische algoritme worden de kandidaat neurale netwerken die overleven binnen de populatie een stukje verder getraind. Op basis hiervan kunnen we de kandidaat neurale netwerken die niet veelbelovend zijn om een goede nauwkeurigheid te behalen uit de populatie verwijderen. Aan het einde van de exploratie houden we dan verschillende bruikbare implementaties van het neurale netwerk over die allemaal passen op het edge apparaat maar die ook allemaal verschillende afwegingen bieden tussen nauwkeurigheid, snelheid, geheugen- en energieverbruik.

Met behulp van deze exploratie methode ontwerpen we dus eigenlijk op maat gemaakte neurale netwerk implementaties die altijd passen op het edge apparaat. Maar een dergelijke exploratie kunnen we helaas niet gebruiken wanneer we bestaande neurale netwerken moeten uitvoeren op edge apparaten. In dat geval moeten we andere technieken gebruiken

om de neurale netwerken aan te passen voor de beperkte middelen van de edge apparaten. Zo wordt er bijvoorbeeld veel onderzoek gedaan naar zogenaamde compressie methoden voor neurale netwerken om ze te verkleinen en hun geheugen behoeften te reduceren. Dergelijke compressie methoden gaan echter vaak ten koste van de nauwkeurigheid van het neurale netwerk. Daarom richten wij ons op een alternatieve onderzoekslijn, waarin we grote neurale netwerken opsplitsen in verschillende delen en die verdelen over meerdere edge apparaten. Dit betekent dus dat deze edge apparaten met elkaar moeten samenwerken om een neuraal netwerk gezamenlijk uit te voeren. Hierdoor kunnen we grote neurale netwerken zonder enig verlies aan nauwkeurigheid uitvoeren op edge apparaten. Dit vergt natuurlijk wel de nodige coördinatie zodat de verschillende gedistribueerde delen van het neurale netwerk op een juiste en efficiënte manier samenwerken. Om dit handmatig uit te programmeren is veel werk en vergt veel expertise van de programmeur. Daarom werken we samen met de Universiteit Leiden aan technieken en software gereedschappen om het gedistribueerd uitvoeren van een neuraal netwerk over verschillende edge apparaten volledig te automatiseren.

En ook hier komt ontwerp exploratie weer om de hoek kijken. Er moet namelijk bepaald worden hoe het neurale netwerk in stukken opgehakt moeten worden en hoe deze stukken over de verschillende edge apparaten verdeeld moeten worden zodanig dat de doorvoersnelheid, het geheugen gebruik en het energieverbruik per edge apparaat zo optimaal mogelijk is. Dit is al een taai optimalisatie probleem op zichzelf, maar wordt verder bemoeilijkt door het feit dat de edge apparaten verschillende onderliggende hardware kunnen hebben en elk edge apparaat vaak meerdere processorkernen tot z'n beschikking heeft die allemaal mee kunnen doen in de uitvoering van het gedistribueerde neurale netwerk. Hiervoor hebben we wederom efficiënte exploratie methoden ontwikkeld die de enorme zoekruimte hiërarchisch kunnen doorzoeken. Eerst met behulp van snelle analytische modellen en daarna door middel van nauwkeurige metingen op echte edge apparaten.

Een belangrijk vraagstuk waar wij op dit moment aan werken in deze context is de robuustheid en veerkrachtigheid van de gedistribueerde neurale netwerken. Edge apparaten zijn namelijk niet zo betrouwbaar als reguliere computers en servers. De batterij van een edge apparaat kan bijvoorbeeld opraken waardoor het zichzelf tijdelijk uitschakelt, of een edge apparaat kan mobiel zijn en dus tijdelijk buiten bereik zijn van andere apparaten waardoor de communicatie tussen de apparaten onderbroken wordt. Deze problemen kunnen ernstige gevolgen hebben voor de werking van het gedistribueerde neurale netwerk. In het beste geval zorgt de uitval of onbeschikbaarheid van een edge apparaat voor een fikse achteruitgang in nauwkeurigheid van het neurale netwerk. Maar in veel gevallen zal het gehele gedistribueerde neurale netwerk simpelweg falen en stoppen met werken. Daarom werken we aan technieken om de neurale netwerken op een zodanige manier op te splitsen en te verdelen over de edge apparaten dat wanneer één of zelfs meerdere edge apparaten onbeschikbaar worden dit weinig tot geen effect heeft op de werking van het neurale netwerk. Hiervoor maken we gebruik van methoden die vanuit verschillende perspectieven het belang bepalen van de verschillende onderdelen van het neurale netwerk voor de uiteindelijke nauwkeurigheid. Op basis van deze belangrijkheid scores kunnen we de onderdelen zo opsplitsen en verdelen over de edge apparaten dat de belangrijkheid per edge apparaat min of meer hetzelfde is. In dit proces repliceren we ook sommige kleine delen van het neurale netwerk die heel belangrijk zijn voor de nauwkeurigheid. Deze strategie zorgt ervoor dat er geen zwakste schakel is in de

gedistribueerde uitvoering van het neurale netwerk en dat de belangrijkste onderdelen altijd meervoudig aanwezig zijn zodat de uitval van edge apparaten getolereerd kan worden.

Dit brengt mij bij het slot van mijn rede. Ik heb het gehad over het belang van modellerings-, simulatie-, en optimalisatietechnieken voor het ontwerp van embedded computersystemen. Dit stelt ontwerpers namelijk in staat om goede ontwerpkeuzes te maken die leiden tot een goed compromis tussen, onder meer, de snelheid, energie-efficiëntie en betrouwbaarheid van het uiteindelijke systeem. Deze technieken zijn echter niet alleen belangrijk voor embedded computersystemen. Als we kijken naar ICT systemen in het algemeen, dan speelt de duurzaamheid van deze systemen, bijvoorbeeld in de vorm van energie consumptie of CO2 voetafdruk, een steeds belangrijkere rol. Op dit moment worden er nog steeds op allerlei plaatsen in de wereld enorme energie-slurpende datacenters neergezet. Maar je kunt je afvragen hoeveel van al die data, afkomstig uit bijvoorbeeld social media applicaties, ook werkelijk langdurig opgeslagen dient te worden? Of neem AI wederom als voorbeeld. Het trainen van de eerder genoemde neurale netwerken kost enorm veel rekenkracht en dus energie. Zo kostte het trainen van ChatGPT-3, dus het oudere en kleinere broertje van de huidige versie, zo'n kleine 1300 MWh met een uitstoot van meer dan 550 ton CO2[1]. De schattingen zijn dat in 2030 de ICT sector verantwoordelijk is voor 21% van de globale energie consumptie[2]. We moeten dus echt aan de bak en serieus gaan werken aan meer duurzame ICT-systemen. Hiertoe zijn modellerings-, simulatie-, en optimalisatietechnieken, zoals ik die eerder heb besproken, instrumenteel. In onze groep zijn we recentelijk begonnen met werk in deze richting, en ik verwacht dat dit werk in de komende jaren een aanzienlijke groei zal gaan doormaken.

Ik ben nu op het punt aangekomen waarop ik een dankwoord wil uitspreken richting al die mensen die er, direct of indirect, voor gezorgd hebben dat ik hier nu sta. Allereerst dank ik het College van Bestuur, de Decaan en het bestuur van de faculteit Natuurwetenschappen, Wiskunde en Informatica voor het in mij gestelde vertrouwen als leerstoelhouder Embedded Computer Systemen. Verder wil ik de vorige directeur van het IvI, Marcel Worring, en de huidige directeur, Alfons Hoekstra, bedanken voor het creëren van de leerstoel. Ook Cees de Laat heeft hier een instrumentele rol gespeeld, waarvoor ik hem enorm dankbaar ben. Ik wil mijn promotor, Bob Hertzberger, bedanken. Na mijn studie zag hij blijkbaar iets in mij en heeft hij mij de kans gegeven om promotieonderzoek te doen, en daarna een onderzoekscarrière als universitair docent te starten. Hij heeft mij altijd de volledige ruimte gegeven om mijzelf te ontwikkelen als zelfstanding onderzoeker. In de jaren die volgden, heb ik met vele geweldige promovendi en postdocs mogen samenwerken. Een groot deel van het onderzoek dat ik vandaag besproken heb, is aan hen toe te schrijven. Het zijn er te veel om allemaal bij naam te noemen. Maar de meeste van de huidige promovendi zitten in de zaal en ook een paar oud promovendi. Ik wil jullie hartelijk bedanken voor jullie geweldige werk. Cees de Laat had ik al eerder genoemd, maar hem wil ik apart nog een keer bedanken. Hij is een aantal jaar mijn sectieleider geweest en is de promotor van veel van de promovendi die ik heb begeleid. Cees, we hebben heel wat afgelachen in de afgelopen jaren, en ik hoop dat dit nog even mag voortduren. De mede stafleden van onze PCS onderzoeksgroep wil ik natuurlijk ook niet overslaan. Clemens, Ana-Lucia, Anuj, Benny, Dolly, Simon en Martin, dank jullie wel voor de

[1] "The Carbon Footprint of ChatGPT", K.G.A. Ludvigsen, Towards Data Science, Dec. 2022.
[2] "How to stop data centres from gobbling up the world's electricity", N. Jones, Springer Nature, 561(7722), Sept. 2018.

hele fijne werksfeer binnen de groep. En het secretariaat, en met name Grace natuurlijk, bedankt voor jullie support! Verder wil ik alle andere collega's binnen het IvI die ik nog niet expliciet heb genoemd bedanken. Gezamenlijk zorgen we voor een instituut waar ik me sinds jaar en dag enorm thuis voel. Dank aan alle nationale en internationale collega's waarmee ik door de jaren heen heb mogen samenwerken. Ik wil hier echter één iemand in specifiek bedanken waarmee ik in de afgelopen 20 jaar altijd nauw heb samengewerkt: Todor Stefanov. Todor, we zijn in de afgelopen jaren door dik en dun gegaan en hebben in die tijd gezamenlijk vele mooie successen mogen beleven. Ook op dit moment doen we allerlei spannende onderzoeksprojecten samen. Ik wil je enorm bedanken hiervoor en hoop nog vele jaren met je te mogen samenwerken.

Dan ben ik nu aangekomen bij de basis waarop je altijd kunt terugvallen: vrienden, familie en schoonfamilie. Ik wil mijn vrienden bedanken, gewoon omdat ze al meer dan 40 jaar mijn vrienden zijn. Dan wil ik mijn moeder, die hier aanwezig is, bedanken. Helaas kan mijn vader er niet meer bij zijn, maar gelukkig heeft hij het nog wel mogen meemaken dat mijn benoeming op 1 september 2020 rond was. De steun van mijn ouders was altijd onvoorwaardelijk, in alles wat ik doe en deed. Lest best, wil ik natuurlijk mijn gezin bedanken, en met name mijn vrouw Caroline, voor al het geduld dat ze met mij hebben gehad in al die jaren. Ik zeg vaak dat mijn werk mijn hobby is, en ik besteed nu eenmaal veel tijd aan mijn hobby.

Ik heb gezegd.