Universitatea “Lucian Blaga” din Sibiu
Facultatea de Inginerie
Departamentul de Calculatoare și Inginerie Electrică

Dezvoltarea unor metode eficiente de optimizare multi-obiectiv, destinate sistemelor de calcul complexe

- Rezumat -

Autor:
Ing. Radu Chiş

Conducător științific:
Prof. univ. dr. ing. Lucian N. Vinţan, membru titular al Academiei de Științe Tehnice din România

SIBIU, Septembrie 2017

Cuprins

Rezumat

În ultimul deceniu sistemele de calcul au cunoscut numeroase îmbunătățiri, într-un ritm mult mai alert, dacă este să le comparăm cu programele ce rulează pe ele. Astfel, acestea au devenit din ce în ce mai complexe [1], [2], [3], [4], [5]. Cei care proiectează aceste sisteme de calcul trebuie să aibă în vedere un spațiu de căutare (proiectare) ce este constituit din milioane de miliarde de configurații diferite, spațiu dat de multitudinea de parametri variabili. Înainte ca un microprocesor sa fie scos pe piață, se implementează un simulator care îl modelează precis și în care parametrii de intrare pot fi variați cu ușurință. Configurațiile simulate sunt evaluate în funcție de mai multe obiective: performanța, temperatura disipată, consumul de energie, aria de integrare, costul etc. Benchmark-urile pe care se realizează aceste evaluări pot necesita câteva zile, astfel încât o căutare exhaustivă în întregul spațiu de proiectare este nefezabilă.

Există deja câteva *framework*-uri dedicate explorării automate a spațiului configurațiilor (*automatic design space exploration* - ADSE) și care ajută arhitecții sistemelor de calcul să găsească cele mai bune configurații, în funcție de obiectivele pe care le setează. Pentru a rezolva aceste probleme complexe de tip NP-hard, *framework*-urile actuale folosesc căutarea euristică și diferiți algoritmi de optimizare multi-obiectiv. Odată cu creșterea complexității și a numărului de core-uri, problema găsirii celor mai bune soluții devine însă tot mai dificilă. *Framework*-urile au nevoie de îmbunătățiri și optimizări atât în ceea ce privește algoritmii meta-euristici de căutare, cât și simulatoarele ce le rulează.

Scopul acestei teze de doctorat este de a folosi instrumentul software FADSE (*Framework for Automatic Design Space Exploration*), dezvoltat în grupul de cercetare condus de domnul profesor Lucian Vințan [6], pentru a dezvolta metode originale și eficiente de optimizare multi-obiectiv pentru sisteme de calcul complexe. Pentru aceasta a fost necesară îndeplinirea următoarelor **obiective** subsidiare:

- analiza metodelor de ultimă generație de explorare a spațiului configurațiilor, inclusiv a meta-euristicilor multi-obiectiv și a metricilor de performanță aferente acestora;
- analiza *framework*-urilor de ultimă generație de explorare a spațiului configurațiilor si compararea lor cu instrumentul software FADSE;
- îmbunătățirea *tool*-ului FADSE astfel încât să poată fi folosit cu mai multe simulatoare (pentru sisteme complexe cu microprocesoare dar și pentru alte sisteme, de exemplu actuatoare magnetice) prin integrarea unor mecanisme noi de *checkpointing*, a unor constrângeri pentru obiectivele considerate și a unor parametri de tip flotant;
- integrarea unor meta-euristici noi și compararea lor cu algoritmii deja implementați în FADSE;
- integrarea unor noi metrici care evaluează mai adecvat calitatea soluțiilor găsite;
- integrarea unui simulator multi-core x86 de ultimă generație, care poate expune ca obiective performanța, aria, energia consumată și temperatura, astfel încât să poată rula un proces de simulare inspirat din activitatea IT reală;
- îmbunătățirea algoritmilor multi-obiectiv de căutare și optimizare integrați în FADSE, prin rularea concomitentă a doi sau mai mulți algoritmi (meta-optimizare), dar într-un timp necesar rulării unuia singur;
- îmbunătățirea timpului necesar pentru simulare în FADSE, fie prin folosirea unor simulatoare mai rapide, lucru care nu este întotdeauna posibil, fie prin adaptarea și

încapsularea unor tehnici provenite din domeniul învățării automate care scad timpul de simulare sau chiar evită simularea cu totul;
- integrarea și optimizarea altor simulatoare, din afara domeniului sistemelor de calcul, pentru a demonstra versatilitatea *tool*-ului FADSE;

Capitolul 2 prezintă și analizează în mod critic câteva metode deja cunoscute de explorare a spațiului configurațiilor și care au fost folosite pe parcursul capitolelor următoare. Ne-am focusat pe meta-euristici de tipul multi-obiectiv, eficiență Pareto și metrici multi-obiectiv de evaluare a performanței.

În Capitolul 3 am prezentat *tool*-ul FADSE (*Framework for Automatic Design Space Exploration*), caracteristicile sale principale și îmbunătățirile pe care le-am adus acestuia și pe care le-am implementat. Optimizările aduse și folosite în procesele următoare de optimizare includ un mecanism de *checkpointing* pentru algoritmul SMPSO de optimizare multi-obiectiv, constrângeri pentru parametrii de ieșire (obiectivele generate de simulator) și permiterea utilizării parametrilor de tip flotant. De asemenea, sunt prezentate câteva meta-euristici, CNSGA-II și MOCHC, precum și o nouă metrică de evaluare a performanţei: diferența de hipervolum a două mulțimi. La final am implementat și o distribuire dinamică a simulărilor pe clienți, cu beneficii semnificative asupra timpului de optimizare.

Capitolul 4 prezintă un proces de optimizare a procesorului *Grid ALU Processor* (GAP) [7], unde am comparat meta-euristicile prezentate în Capitolul 3 utilizând metrici de performanţă bine-cunoscute. În acest capitol am rulat mai mulți algoritmi de optimizare multi-obiectiv pornind de la aceeași populație, luată la întâmplare, pentru a examina care dintre algoritmi generează cele mai bune rezultate. Pentru aceasta am rulat algoritmi genetici multi-obiectiv precum NSGA-II și SPEA2, implementările noastre CNSGA-II (în mai multe versiuni) și MOCHC, dar și un algoritm bio-inspirat, anume SMPSO. Am comparat rezultatele folosind metrici bine cunoscute: acoperirea, hipervolumul și implementarea noastră a diferenței de hipervolum a doua seturi (mulțimi). Am descoperit la finalul procesului de optimizare că algoritmul SMPSO produce indivizii de cea mai bună calitate, NSGA-II e mai bun din punct de vedere calitativ decât SPEA2 dar și că algoritmul MOCHC are cea mai mare viteză de convergenţă. În ultima parte am integrat metode de extragere a trăsăturilor (*feature selection*) pentru a putea micșora spațiul de căutare de la mai mult de 1 milion de configurații la doar 17.000 de configurații. Această modificare a scăzut timpul de simulare dar și calitatea soluțiilor.

Capitolul 5 introduce un simulator aferent unui sistem de tip multi-core / many-core relativ nou, Sniper, cu ajutorul căruia am proiectat și rulat o co-optimizare hardware-software pe 3 obiective: performanță, arie și energie. Creșterea în complexitate a hardware-ului, posibilă prin micșorarea mărimii tranzistorilor, ar trebui suplimentată prin îmbunătățiri software, dar nu se întâmplă tot timpul astfel. În cercetarea noastră am investigat atât optimizarea hardware, cât și optimizarea hardware-software (HW-SW). Am început printr-un proces de căutare pentru a evalua impactul parametrilor software (moduri de optimizare a gcc, numărul de thread-uri și planificatorul sistemului de operare) asupra performanţei și energiei. Am găsit că unii parametrii au un impact relativ mare asupra obiectivelor. Apoi am rulat un proces de căutare automatizată în două configurații diferite: în prima am variat doar parametrii hardware, iar în a doua am variat atât parametrii hardware cât și software, deopotrivă. În final am comparat cele

două rulări și am concis că optimizarea HW-SW produce rezultate mai bune și deci, co-optimizarea ar trebui să fie paradigma aleasă când se dorește crearea de noi sisteme cu microprocesoare avansate.

Capitolul 6 îmbunătățește mai departe procesul de explorare a spațiului de căutare pe simulatorul Sniper prin adăugarea unui al patrulea obiectiv: temperatura. Aceasta reprezintă la ora actuală o limită fizică pe care nu o putem depăși: la în jur de 100 de grade Celsius, timpul de viață al microprocesoarelor scade, iar la peste 115 grade Celsius apar probleme ireparabile. Am îmbunătățit simulatorul Sniper prin adăugarea unor scripturi pentru HotSpot, un program de modelare termică, unde utilizând *trace*-uri de putere și ariile unităților funcționale din microprocesor se pot calcula temperaturile aferente diferitelor module din arhitectura Sniper. O provocare pe care a trebuit să o depășim a fost crearea automată a *floorplan*-ului, reprezentarea unităților funcționale ale procesorului, pentru a putea rula apoi o optimizare automată în 4 dimensiuni (deci cu 4 obiective). Din câte știm, suntem primii care au rulat o căutare în 4 dimensiuni, printre care se numără și temperatura, utilizând simulatorul multi-core Sniper. Am comparat rezultatele noastre 4D cu o rulare 3D, în care temperatura se calculează separat, după optimizarea automată, și am tras concluzia că noua noastră abordare originală obține rezultate mai bune.

Capitolul 7 introduce un nou simulator dintr-un cu totul alt domeniu: un model software creat în MATLAB și COMSOL al unui actuator magnetic pus la dispoziție de Continental Automotive Systems din Sibiu. Pentru acest nou simulator am scris un nou conector la FADSE și am utilizat constrângerile ieșirilor și parametri rațional prezentați in Capitolul 3. Optimizarea acestui simulator a adus noi provocări pentru că trebuiau optimizate 8 obiective, unele având constrângeri. FADSE a făcut față acestui număr mare de ieșiri și a generat rezultate bune după aproximativ o săptămână de procesări. Apoi am utilizat tehnici de învățare automată pentru a reduce spațiul de căutare, prin excluderea parametrilor de intrare care erau dependenți, dar și tehnici de tip *Response Surface Methodology* (RSM) pentru scăderea timpului de simulare. Am reușit să învățăm rețele neuronale în vederea implementării metodelor de RSM, pentru a avea erori de aproximativ 1%. Apoi am rulat un proces de optimizare al actuatorului unde în loc de simulări am folosit predicțiile rețelelor neuronale implementate și am avut predicții cu o eroare de maximum 2%, ceea ce este remarcabil. Am concluzionat că în acest caz scăderea timpului de simulare și o eroare foarte mică, merită tot acest efort de cercetare-proiectare-implementare.

Capitolul 8 prezintă o îmbunătățire majoră a FADSE: introducerea unui nivel de meta-optimizare, care poate rula mai mulți algoritmi de optimizare în paralel, în timpul necesar rulării unui singur algoritm. Am realizat aceaste simulări pe simulatorul de procesor superscalar GAP, pentru că simulările durează minute și nu ore, ca în alte cazuri. Inițial am implementat o variantă naivă, care selecta indivizii pentru următoarea generație la întâmplare. A doua variantă determină numărul de indivizi generat de un anumit algoritm de optimizare, conform unei ponderi. Ponderea este determinată de calitatea soluțiilor generate de acest algoritm în generația precedentă. Astfel, algoritmii care generează indivizi mai buni, vor avea un procentaj mai mare de indivizi creați în cadrul noii generații. Am analizat rezultatele și am detectat o sinergie în nivelul nostru de meta-optimizare, pentru că meta-optimizarea a generat soluții mai bune calitativ decât super-poziția a doi algoritmi care conțin de două ori mai mulți indivizi și au durat, în rulare, de două ori mai mult.