Scheduled protocol programming

Dokter, K.P.C.

Samenvatting

Met de opkomst van multicore processoren en datacenters is computer hardware in toenemende mate parallel geworden, waardoor het mogelijk is om verschillende softwareonderdelen tegelijkertijd op verschillende machines uit te voeren. Coördinatie van deze softwareonderdelen wordt het beste uitgedrukt in een coördinatietaal als een expliciet interactieprotocol dat de interacties tussen alle onderdelen van de software duidelijk definieert.

Een expliciet interactieprotocol verbetert niet alleen de structuur van de code, maar maakt ook geautomatiseerde analyse van het protocol mogelijk om de uitvoeringsefficiëntie van de software te verbeteren. In het bijzonder bevatten interactieprotocollen belangrijke informatie die essentieel is voor efficiënte roostering, een activiteit die betrekking heeft op de toewijzing van (reken)middelen aan softwaretaken. In dit proefschrift richten we ons specifiek op het verbeteren van de uitvoeringsefficiëntie door middel van roosteren. Roosteren is bijna altijd de verantwoordelijkheid van een generiek besturingssysteem dat geen aannames maakt over de software en daardoor alle relevante roosterinformatie in die software negeert. Als gevolg hiervan kan het besturingssysteem alleen niet zorgen voor optimaal geroosterde uitvoering van de software.

In dit proefschrift stellen we een oplossing voor die het protocol in de software verandert, zodat het efficiënt wordt geroosterd door het generieke besturingssysteem. Het belangrijkste idee is om gebruik te maken van de dualiteit tussen roostering en coördinatie. Om precies te zijn, analyseren we het protocol van de software om een optimale roosterstrategie voor deze software te bepalen. Vervolgens dwingen we dit optimale rooster af door de strategie op te nemen in het oorspronkelijke protocol. Als gevolg hiervan dwingen we de onwetende roostermodule van het besturingssysteem om ons vooraf bepaalde optimale rooster te volgen.

Om dit grotere doel te bereiken, presenteren we drie kleinere bijdragen. Ten eerste verkrijgen we een uitgangspunt voor de planningsinformatie die beschikbaar is in een coördinatietaal door twee coördinatiestalen, BIP en Reo, met elkaar te vergelijken. Onze vergelijking leidt tot het voorstel van een compositieoperator voor datasensitieve BIP-architecturen en de uitbreiding van de theorie van zachte beperkingsautomaten (soft constraint automata) met geheugencellen en bipolaire voorkeurswaarden.

Vervolgens bepalen we alle onafhankelijke delen van de software (inclusief die in het protocol) die kunnen worden gepland. Hier introduceren we twee werkelijk gelijktijdige semantieken, namelijk multigelabelde Petri-netten en stroombeperkingen (stream constraints) in regelgebaseerde vorm. Door deze semantieken als interne representatie te gebruiken, verbeteren we aanzienlijk de state-of-the-art

Reo-compiler. Als bijproduct stellen we een tekstuele taal voor genaamd Treo voor, waarmee we grote protocollen kunnen opbouwen door primitieve protocollen samen te stellen.

Ten slotte representeren we alle relevante roosterinformatie van elk deel van de software door de software uit te drukken als een werkautomaat die uitdrukt hoeveel werk elk deel van de software kan/moet doen. We gebruiken deze werkautomaten om een speltheoretisch roosterraamwerk te ontwikkelen dat roosteren formaliseert als een tweespeler-nulspel dat wordt gespeeld op een graaf. We maken een kleine aanpassing op bestaande oplossingen om een roosterstrategie te berekenen voor een kleine cyclo-statistische datastroomtoepassing die de doorvoer optimaliseert. Zoals beloofd dwingen we de optimale roosterstrategie af door deze te integreren met het oorspronkelijke protocol, waardoor aangepaste wijzigingen in de standaard roostermodule van het besturingssysteem worden vermeden.

Hoofdstuk 9 geeft een uitgebreidere samenvatting van dit proefschrift.