Cercetare in limbajul AGAPIA, compilator şi aplicaţii

de

Ciprian I. Păduraru

Rezumatul tezei
depuse pentru îndeplinirea parţială a
cerinţelor pentru titlul de

Doctor în Informatică

Facultatea de Matematică şi Informatică
Universitatea din Bucureşti

2015
Bucureşti, Romania

Comisie:

Prof.Dr. Gheorghe Ştefănescu, *Coordonator Ştiintific*, Univ. din Bucureşti
Prof.Dr.Ing. Emil Slusanschi, *Referent*, Univ. Politehnica din Bucureşti
Conf.Dr. Florin Crăciun, *Referent*, Univ. Babes-Bolyai, Cluj-Napoca
Conf.Dr. Traian Şerbanuta, *Referent*, Univ. din Bucureşti

După ce aplicaţiile distribuite au devenit din ce în ce mai folosite şi mai sofisticate, a fost nevoie de crearea de noi limbaje de programare şi modele pentru programarea distribuită. Scopul principal al acestor limbaje a fost de a fi mai expresive şi de a creşte productivitatea. Cu toate acestea, există aspecte încă nerezolvate care provoacă multe eforturi programatorilor in vederea dezvoltarii aplicaţiilor distribuite. În procesul de scriere a programelor pentru sisteme paralele cu memorie distribuită, folosind o soluţie comună, cum ar fi MPI, programatorii se concentrează pe un set de paşi iterativi, crează task-uri care pot rula in paralel iar apoi se ocupă de comunicarea dintre acestea respectiv sincronizarea lor, intr-un mod explicit. Înainte de implementarea efectivă într-un limbaj de programare, programatorii gândesc arhitectura programului asemanator cu o diagramă dataflow [3], unde programul este imparţit in diferite entităţi independente ce se ocupă de calcule iar legaturile dintre acestea reprezintă procesul de comunicare. Din cauza stilului secvenţial de a scrie un program, este de multe ori greu de înţeles exact cum sunt separate entitaţile si interacţiunile dintre acestea. Acest aspect ar putea avea consecinţe negative asupra implementării programelor. În primul rând există o lipsă de modularitate, ceea ce face codul sursă mai greu de urmarit, în special pentru a înţelege comunicarea şi sincronizarea dintre diferite entitaţi. Un alt dezavantaj este acela că programele pot deveni mai predispuse la erori iar procesul de mentenanţă este mai dificil.

Limbajul AGAPIA a fost iniţial introdus în [1] (versiunea 0.1) şi a evoluat prin adaugarea recursiei in [2] (versiunea 0.2). Principalele mele contribuţii în continuarea dezvoltării acestui limbaj au fost:

1. Construirea unui set de programe pentru compilarea si executarea programelor.
2. Analiza unor categorii de aplicaţii care se pretează dezvoltării în AGAPIA.
3. Propunerea unei extensii a limbajului prin adăugarea unor pattern-uri.
4. O propunere pentru implementarea pointerilor temporali si analiza unor algoritmi de scheduling pentru executarea modulelor

Obiectivul limbajului AGAPIA este de a maximiza potenţialul de paralelism pe ambele tipuri de platforme, multi-core şi many-core, şi în acelaşi

timp să ofere un model de programare modular, transparenţă în comunicare şi cod reutilizabil. Obiectivul principal al acestei teze este de a analiza în ce categorii de probleme ar putea fi folosit cu succes limbajul AGPAPIA, prezentarea backend-ul de compilare şi execuţie, şi de a face un plan pentru dezvoltarea ulterioara. Fiind avantajat de modelul de communicare transparent şi instrucţiuni de nivel înalt pentru a simplifica procesul de dezvoltare, implementarea programelor distribuite devine mai modulară, mai usor de dezvoltat, şi mai apropiată de formularea iniţială a soluţiei problemei. Deoarece codul sursă AGAPIA este compus în mare parte din cod specific C/C++ adăugand doar câteva instrucţiuni şi specificaţii specifice limbajului, este de asteptat ca utilizatorii sa inteleagă uşor acest limbaj de programare. Prin "implementare paralelă" înţelegem ambele modele de programare paralelă - shared memory şi distributed memory. Majoritatea programelor scrise în limbajul AGAPIA au acelasi cod sursă pentru implementarea în cele doua modele - singura excepţie este situatia în care utilizatorii doresc să acceseze memoria shared folosind structuri de date proprii şi nu tipul pe care AGAPIA îl pune la dispozitie, "buffer". Folosirea unui limbaj de programare de nivel înalt vine de obicei cu un cost - o degradare a performanţei fie in timpii de execuţie şi/sau memorie ocupată. Dupa cum această teză va demonstra prin realizarea unor teste de performanţă şi compararea rezultatelor între aplicaţii implementate în AGAPIA versus MPI şi C/C++, acest cost al performanţei este minimal.

Prezentarea este structurată în două părţi: prima parte (Capitolele 2,3, 4) discută despre limbajul AGAPIA, bazele programării, arhitectura şi implementarea backend-ului de compilare şi execuţie, în timp ce partea a doua analizează categorii de aplicaţii în care utilizatorii pot fi avantajaţi de folosirea acestui limbaj.

Capitolul 2 prezintă lucruri de bază despre limbajul AGAPIA. În partea de început se explică cum se poate scrie un program, sintaxa şi cum funcţionează comunicarea între module. În continuare, sunt prezentate instrucţiuni de nivel înalt şi pattern-uri atât la nivel de sintaxă cât şi semantica lor, potenţialul acestora de folosire şi cum sunt implementate.

Setul de tool-uri pentru compilarea şi executarea programelor este prezentat în Capitolul 3. Capitolul începe prin prezentarea în ansamblu cum sunt executate modulele. Apoi se descrie cum interpreter-ul şi unealta de analiză a codului comunica împreună şi se folosesc de macro-urile de deployment. Capitolul 4 conţine o discuţie despre câţiva algoritmi de scheduling pentru

executarea modulelor. Este descris atât algoritmul default cât şi extensii precum setarea unor deadlineuri pentru module, fairness-ul dintre acestea dar şi extinderea deployment-ului pe arhitecturi ierarhice de tip ring.

Capitolul 5 conţine patru secţiuni. Primele două secţiuni prezintă tipuri de aplicaţii care se pretează cel mai bine la implementarea în AGAPIA: aplicaţii de tip dataflow şi wavefront. Secţiunea 1, conţine o motivaţie de ce AGAPIA este eficientă pentru dataflow programming, împreună cu o implementare a unei aplicaţii de procesarea semnalelor distribuit. Implementarea va demonstra că prin folosirea acestui limbaj programatorii pot deveni mai productivi, cu pierderi minimale de performanţă. Secţiunea a doua prezintă cum se poate beneficia de avantajele limbajului AGAPIA în paralelizarea pattern-ului wavefront. Programatorii pot implementa rezolvări pentru aceste tipuri de problemele, în paralel, cu un efort similar implementării seriale a aceleaşi probleme. Din nou, pierderile de performanţă sunt minime. Ultimele două secţiuni propun o extensie pentru versiunea curentă de AGAPIA şi o comparaţie cu un alt limbaj recent pentru programarea paralelă.

A treia secţiune prezintă o altă perspectivă pentru folosirea limbajului: aplicaţii ce folosesc streaming de date spaţio-temporal. Este o discuţie conceptuală ce vizează o devoltare ulterioară, nefiind inclusă în versiunea curentă de compilator. Pot fi folosite pentru implementarea mai productivă a aplicaţiilor care sunt executate pe obiecte în mişcare precum avioane, maşini, etc. Cu specificaţii de nivel înalt programatorii pot construi rapid aplicaţii ce au în componenţă componente individule ce comunica prin stream-uri de date spaţio-temporale. Prin implementarea unei librarii de module pentru manipularea datelor, AGAPIA poate ajuta dezvoltatoriii să se focuseze pe dezvoltarea aplicaţiei la nivel înalt şi să evite partea de low-level privind comunicarea şi manipularea datelor. În final, ultima secţiune containe o comparaţie între un limbaj recent de programare paralelă, X10, şi AGAPIA, incluzând totodată şi o analiză la cum s-ar putea folosi AGAPIA de X10 în dezvoltarea componentei de execuţie a modulolor.

Ultimul capitol (Capitolul 6), prezintă idei de dezvoltare a limbajului AGAPIA. Pe partea de unelte, una dintre ideile principale de continuare ar fi crearea unei aplicaţii în care se pot vedea/crea modulele, specificaţia acestora şi interacţiunea lor în mod vizual. Astfel, dat un cod existent se poate vedea uşor care sunt entităţile unei aplicaţii şi interacţiunea dintre ele. De asemena, utilă ar fi şi simularea vizuală pas cu pas a executării modulelor după un input dat. În partea de backend, teza prezintă mai multă modificări posibile precum adăugarea unui scheduler folosind task-stealing şi GPGPU,

diferite optimizări low-level, implementarea unui sistem mai user friendly de raportare a erorilor şi apelarea componentelor de compilare/executare de module direct din cod existent de C/C++.

Rezultatele prezentate în aceasta teză au fost publicate în:

1. C.I. Paduraru, *Dataflow programming using AGAPIA*, Proceedings ISPDC 2014, IEEE, CPS pp. 327-334.

2. I.T. Banu-Demergian, C.I. Paduraru, and Gh. Stefanescu, *A new representation of two-dimensional patterns and applications to interactive programming*, Proceedings FSEN 2013, LNCS 8161, pp. 183-198. Springer, 2013.

3. C.I. Paduraru, *A new online load balancing algorithm in distributed systems*, Proceedings SYNASC 2012, IEEE, CPS, pp. 327-334.

4. C.I. Paduraru, *An online load balancing algorithm for a hierarchical ring topology*, Proceedings ICCCC 2014, confirmed for publication International Journal on Computers, Communications & Control, No 6, 2014.

5. C.I.Paduraru, *Distributed programming using AGAPIA*, International Journal of Advanced Computer Science and Applications, Vol 5, No 3, 2014.

6. C.I. Paduraru, *A Greedy Algorithm for Load Balancing Jobs with Deadlines in a Distributed Network*, International Journal of Advanced Computer Science and Applications, Vol 5, No 2, 2014.

Bibliografie

[1] Dragoi, C., Stefanescu, G.: AGAPIA v0.1: A Programming Language for Interactive Systems and its Typing System, FInCo, Electr. Notes Theor. Comput. Sci. 203(3): 69-94 (2008).

[2] Alexandru Popa, Alexandru Sofronia, Gheorghe Stefanescu, High-level Structured Interactive Programs with Registers and Voices, Journal of Universal Computer Science, vol. 13, no 11 (2007) 1722-1754.

[3] Tiago Boldt Sousa Dataflow Programming Concept, Languages and Applications , Proceedings of Doctoral Symposium on Informatics Engineering, 2012.