Re-Engineering prin descoperirea automată a transformărilor aplicabile pentru paralelizare

Dr. Conor RYAN, Laur IVAN
Computer Science and Information Systems University of Limerick, Limerick, Ireland

Algoritmii combinaționali folosiți pentru paralelizarea programelor secvențiale sunt limitați din punct de vedere al performanțelor. Majoritatea programelor bazate pe acest tip de algoritmi folosesc un set de restricții drastice care au scopul reducerii numărului verificărilor dependințelor între instrucțiuni.
În articolul de față vom prezenta o abordare pe baza algoritmilor genetici. Paragen dispune de un algoritm în mai mulți pași care tratează problema paralelizării unui program secvențial prin generarea de transformări care vor fi aplicate acelui program. Paragen lucrează în două moduri: la nivel de atom și la nivel de succesiune de cicluri. Cele două moduri sunt descrise detaliat în secțiunile următoare.
Cuvinte cheie: *procesare paralelă, planificarea proceselor*

1. Introducere

Pentru generarea aplicațiilor pentru calculatoarele paralele există diferite abordări. Una dintre ele este de a construi programul paralel practic de la zero, pentru o anumită mașina paralelă, chiar dacă există o variantă secvențială a programului respectiv. Acest tip de rezolvare a problemei necesită compilatoare sofisticate și implicit, costisitoare (ca de exemplu Kai C/C++) și totodată cunoștințe aprofundate în domeniul programării paralele din partea programatorului.
O altă abordare este aceeea de a încerca transformarea programelor secvențiale în programe paralele. Problemele apărute în această metodă sunt de tipul Dependenței de Date (Data Dependencies) [Lewis 92]. Din acest punct de vedere, abordarea standard este de a impune restricții drastice programelor secvențiale pentru a elimina din numărul de combinații posibile. De asemenea este necesar și un analizor de dependințe. O direcție în modificarea programelor secvențiale către programe paralele este utilizarea Algoritmilor Evolutivi – Programare Genetică [Koza 92]. Paragen [Ryan96] [Ryan 97] este un sistem care utilizează algoritmii genetici pentru a genera liste de transformări care, aplicate unui program secvențial, generează un program paralel 100% identic din punctul de vedere al funcționalității. Datorită faptului ca nu se generează transformări efective asupra programelor, ci liste de transformări, dovedirea identității celor două versiuni este foarte ușoară și de asemenea și operațiile genetice. Această metodă de verificare a dependentelor este numită analiza direcționată a dependentelor de date (Directed Data Dependency Analysis).

2. Spațiul problemei

După cum am evidențiat, principala dificultate a convertirii programelor secvențiale este dependența instrucțiunilor din punct de vedere al datelor modificate. Dar aceasta nu estre singura problemă. Programul paralel obținut astfel trebuie distribuit astfel încât procesoarele mașinii paralele să fie utilizate cât mai echilibrat [Lewis 92]. Este o risipă de resurse dacă, spre exemplu, un procesor este utilizat la 80% din capacitate iar altul la 20%. Această problemă poate fi rezolvată prin planificarea proceselor [Lewis 92].
Există două metode de planificare: *planificarea statică* și *planificarea dinamică* a proceselor. Planificarea statică a proceselor presupune ca procesul planificator să dispună de informațiile necesare planificării (topologia rețelei de procesoare, performanțe, duratele proceselor etc.) și pe baza acestor informații să realizeze distribuția proceselor. Pe de altă parte, varianta dinamică presupune că nu există informații la în-

ceput şi planificatorul de procese realizează strângerea de informaţii pe parcursul rulării programului. Pe de-o parte, varianta statică de planificare presupune un volum foarte mare de date la începutul rulării, unele dintre ele greu de obţinut. Pe de altă parte, varianta dinamică presupune creşterea comunicaţiei în reţeaua de procesoare.

Proiectul actual este compus din mai multe module: Soft Draw, Soft Plan şi Paragen.

- ***Modulul SoftDraw***

Trasarea de grafuri este un domeniu relativ nou, cu un număr mare şi divers de aplicaţii. Dându-se un graf prin noduri, legături şi relaţiile între noduri şi legături, problema este de a reprezenta graful într-un mod "plăcut". Această caracteristică estetică a grafului se poate traduce în: minimizarea numărului de intersecţii ale arcelor dintre noduri, sau minimizarea numărului de îndoiri la margini, sau desenarea grafului astfel încât toate săgeţile arcelor să fie orientate în jos. Majoritatea algoritmilor care există sunt algoritmi dedicaţi, adică sunt proiectaţi să rezolve o anumită problemă, fiind foarte puţin flexibili în rezolvarea altora.

Acest modul poate fi privit ca o problemă de satisfacere de restricţii, unde fiecare considerent estetic este reprezentat ca o restricţie, iar soluţia este aceea care satisface cele mai multe sau cele mai importante dintre cerinţe.

Un program paralel poate fi reprezentat ca un graf din punct de vedere al interdependenţelor în cadrul fluxului de date. Considerentele estetice prezentate anterior se transpun în flux de informaţie (data flow). Din acest punct de vedere, SoftDraw aplică asupra programului paralel un algoritm de "infrumuseţare". Neajunsurile trasării de grafuri cu algoritmi dedicaţi (hard algorithms) se exprimă prin faptul că programele paralele sunt generate pentru diferite toologii de reţele de procesoare, şi chiar programele privite ca grafuri pot avea diferite organizări. De aceea SoftDraw foloseşte avantajele programării genetice.

- ***Modulul SoftPlan***

După cum am evidenţiat anterior, programul este privit ca un graf, cu conexiuni date de dependenţele între module (date, flux, control). Pentru aceste module apare problema distribuirii în reţeaua de procesoare, planificării proceselor (fiecare modul este un proces). Dată fiind generalitatea reţelelor de procesoare şi a programelor asupra cărora trebuie să acţioneze, SoftPlan realizează planificarea cu ajutorul algoritmilor genetici.

- ***Modulul Paragen***

Paragen este modulul care realizează conversia programului din secvenţial în program paralel funcţional echivalent, şi este descris detaliat în secţiunea următoare

3. Paragen

Paragen realizează transformarea programelor secvenţiale în programe paralele prin generarea de liste de transformări. Introducem noţiunea de '*segmente de program*' unde un segment este partea de program care se transformă.

Paragen a fost construit având la bază una din legile programării OCCAM:

SEQ(A,B)=PAR(A,B)

Dacă A şi B nu interacţionează între ele (nu există variabile utilizate de B şi modificate de A şi invers), atunci cele două se pot executa în paralel.

Paragen a fost proiectat iniţial pentru a procesa programe liniare (şiruri de instrucţiuni). Structura cromozomilor folosiţi de algoritmii genetici este una arborescentă, care favorizează transmiterea secvenţelor de instrucţiuni către transformări. În figura 1 este un exemplu de cromozom al Paragen:

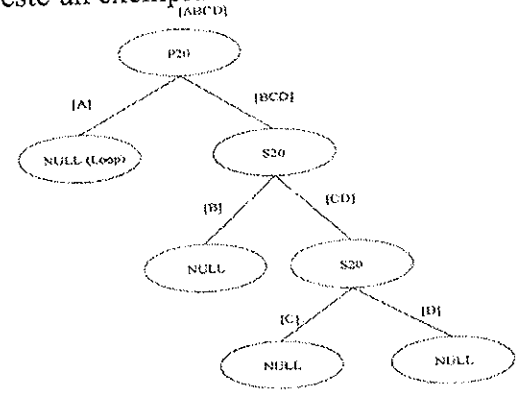

Fig.1. Exemplu cromozom Paragen

Dar programele liniare care sunt supuse paralelizării sunt foarte puţine, majoritatea conţinând cicluri, iar în interiorul acestora,

mai importantă este dependența între iterații, deoarece dacă există o astfel de dependență, ciclul nu mai poate fi paralelizat. Din această diferență între dependențele de date a apărut necesitatea a două moduri de funcționare: unul numit "*Atom Mode*" (sau mod de funcționare la nivel de atom), și unul numit "*Loop Mode*" (pentru prelucrarea ciclurilor). Definim un *Atom* ca fiind ori o instrucțiune simplă (atribuire, apelare de funcții etc) sau un set de cicluri consecutive (MetaLoop).

3.1 Modul de funcționare la nivel de atom

În modul de lucru la nivel de atom, sunt două clase de bază. Prima, numită F/L, este o clasă în care fiecare transformare este de forma Fxxx/Lxxx unde xxx este SEQ/PAR. Aceste transformări planifică o singură instrucțiune dintr-un segment și transmite restul segmentului la următoarea transformare.
Cealaltă clasă, numită P/S, împarte segmentul în două segmente noi și transmite aceste segmente către celelalte transformări. Inițial, segmentele erau împărțite în două părți egale ca număr de instrucțiuni, fiecare din ele fiind transmisă transformării următoare. Acest fapt crea neajunsuri pentru anumite cazuri particulare și, de aceea, fiecare transformare (P și S) a fost divizată în P20/P80 și, respectiv, S20/S80.
Se poate remarca faptul că prin aplicarea unor combinații P20/P80 și S20/S80 se poate obține același rezultat ca prin aplicarea P50/S50, dar reciproca este mai dificil de obținut. În tabelul următor avem un exemplu pentru aplicarea P20:

Operația	Șirul de intrare	Șirul de ieșire
P20	[ABCDE]	[A]
		[BCDE]

Mai există o clasă adițională numită SHIFT, care realizează întârzierea cu o perioadă de timp a execuției secvenței de instrucțiuni dată ca parametru. Această operație are un parametru de intrare și un parametru de ieșire.

3.2. Modul de funcționare la nivel de cicluri

Paragen rămâne în mod Atom până când se termină de efectuat toate transformările. Apoi caută în fiecare secvență de instrucțiuni un atom de tipul MetaLoop. Când găsește un astfel de atom, trece în modul de funcționare la nivel de cicluri. De asemenea, acest mod se bazează pe algoritmi genetici; cromozomii acestei etape sunt șiruri de transformări destinate ciclurilor. Aceste transformări pot fi clasificate în două categorii: optimizări de cicluri [Lewis 92] și legile OCCAM aplicate ciclurilor.
Au fost identificate 14 transformări aplicabile ciclurilor. În continuare vom descrie trei dintre cele mai importante transformări: interschimbarea ciclurilor, fuzionarea și înlocuirea.
Interschimbarea ciclurilor este o operație care nu se referă la ciclu ca o înșiruire de iterații, ci ca o entitate care are anumite legături de interdependență cu altă entitate. Presupunând că avem două cicluri A și B, acestea pot fi interschimbate dacă și numai dacă ciclul A nu depinde de ciclul B și B nu depinde de A. De notat că în acest mod de funcționare interesează mai întâi paralelizarea execuției iterațiilor din cicluri, și după aceea paralelizarea ciclurilor ca entități.
Fuzionarea a două cicluri are ca scop reducerea transferului de date adiționale în structura de procesoare. În cele mai multe cazuri, două cicluri consecutive n-au același număr de iterații. Din acest punct de vedere există două abordări pentru fuzionare: diminuarea sau minim-maxim. Diminuarea se referă la reducerea numărului de iterații al ciclului cu mai multe iterații pentru a se suprapune peste numărul de iterații al celuilalt. Varianta minim-maxim se referă la împărțirea celor două cicluri și a creării unui ciclu care să aiba un index comun:

```
for(i=alpha;i<beta;i++)
for(j=gamma;j<delta;j++)
```

și ciclul comun este:

```
for(k=max(alpha,gamma);k<min(beta,delta);
k++)
```

Înlocuirea se referă la înlocuirea unei ins-

trucțiuni 'for' cu o instrucțiune '**PARFOR**', pentru a realiza execuția în paralel a iterațiilor. Aceasta are la bază aceeași lege OCCAM ca și modul de execuție la nivel de atom al Paragen. Problema care apare în acest caz este verificarea dependențelor. În general, există probabilitatea ca o transformare să nu poată fi aplicată ciclului implicit, ci unui ciclu următor. De aceea a fost proiectat un sistem de trecere peste cicluri fără a executa și fără a pierde transformarea. Și tot din același motiv a fost proiectată o operație auxiliară '**SKIP**' care realizează trecerea marcajului, care indică ciclul asupra căruia se va efectua schimbarea, la ciclul următor.

4. Concluzii si perspective

Proiectul actual oferă o soluție viabilă în domeniul conversiei programelor secvențiale în programe paralele și prin faptul că menține deschisă topologia rețelei de procesoare (de fapt a programului rezultat). În paralelizare, cheia către obținerea unor programe performante este lucrul pentru optimizarea ciclurilor.

Paragen este proiectat să genereze cod paralel care să fie integral compatibil din punct de vedere funcțional cu codul secvențial inițial. O versiune viitoare a Paragen va conține un algoritm de autodeterminare a procentajelor utilizate de operațiile S și P, și de asemenea algoritmi pentru imbunătățirea atribuirii operatorilor pentru cicluri în modul Loop Mode.

Bibliografie

[Bur88] Burns, A. (1998): *Transforming Occam Programs*. In Programming Occam 2, Addison-Wesley

[Dav91] Davis, L.V. (1991): *Handbook of Genetic Algorithms*. Van Nostrand Reinhold

[Lewis 92] Lewis T (1992): *Introduction to Parallel Computing*.Prentice Hall

[Rya97] Ryan, C. and Walsh, P. (1997): *The Evolution of Provable Parallel Programs*. GP 97

[Rya96] Ryan, C. and Walsh, P. (1996): *Paragen: A novel technique of the Auto-parallelization of Sequential Programs using GP*. In Genetic Programming. MIT Press

[Rya95] Ryan, C. and Walsh, P. (1995): *Automatic conversion of programs from serial to parallelusing genetic programming*. In proceedings of Parallel Computing. Springer-Verlag