Modalităţi de analiză şi dezvoltare a programelor în sisteme cu prelucrare paralelă

Ing. Georgios KAPENEKAS
Technical Education Institute, Halkida, Greece

Prelucrarea paralelă este un mod de calcul care poate fi materializat numai în contextul unor caracteristici arhitecturale specifice. În definirea conceptului de sistem cu prelucrare paralelă trebuie avute în vedere resursele fizice şi logice, controlul şi datele. Există mai multe componente ale unui sistem care pot determina paralelismul, fiecare dimensiune putând avea un spectru larg de valori. Articolul abordează modelele de calcul în arhitecturi paralele, implementarea modelelor pe diverse arhitecturi şi comunicarea între procesoare.
***Cuvinte cheie**: paralelism, arhitecturi, procesoare, partajare, sisteme distribuite.*

1 Modelul de calcul al arhitecturilor paralele

Modelul de calcul utilizat în cadrul arhitecturilor paralele constă dintr-un sistem ce realizează o serie de transformări iterative nelimitate. Sistemul nu este un limbaj de programare în sensul clasic, el reprezintă o notaţie utilă pentru a ilustra ideile ce stau la baza implementării într-un limbaj de programare dorit. Un astfel de program constă dintr-o zonă de declarare de variabile, cu specificarea valorilor iniţiale şi o serie de instrucţiuni de tip atribuire. Execuţia unui astfel de program porneşte dintr-o stare ce îndeplineşte condiţiile iniţiale şi execută continuu instrucţiunile conţinute de program. La fiecare pas este selectat aleatoriu un număr de instrucţiuni care va fi executat. Singura condiţie ce se impune este ca toate instrucţiunile să fie executate în mod continuu. Un program va descrie *ce* anume trebuie făcut, în sensul precizării condiţiilor iniţiale şi a transformărilor care vor fi urmate, dar nu va preciza *când* anume trebuie să aibe loc aceste atribuiri şi nici *unde* se vor executa.

Prin separarea conceptelor se obţine o notaţie simplă de programare ce corespunde la o varietate largă de arhitecturi. Se pot lua în discuţie pentru implementare orice tip de arhitecturi: arhitectura von Neumann, arhitecturi multiprocesoare sincrone cu partajarea memoriei sau arhitecturi multiprocesoare asincrone cu partajerea memoriei. Similar cu asocierea transformărilor iterative unei arhitecturi date, se poate realiza şi implementarea programelor într-un limbaj de programare convenţional.

Deoarece am presupus că execuţia programelor nu se încheie, se pune întrebarea cum se determină faptul că programul a fost executat. Pentru aceasta se observă că există o stare a programului numită *punct fix* în care orice execuţie a unei noi instrucţiuni lasă starea neschimbată.

Modelul arhitecturii pentru sisteme sincrone cu partajarea memoriei presupune un număr fix de procesoare identice care împart o memorie comună care poate fi citită sau scrisă de oricare procesor. În sistem există un ceas comun care determină execuţia de către procesoare a unui nou pas din program. În acest caz, atribuirile multiple pot fi implementate uşor: fiecare procesor calculează o expresie din partea dreaptă a atribuirii ce corespunde la o valoare şi atribuie valoarea calculată variabilei. Arhitectura este utilă la calcularea valorilor pentru o expresie care este definită de un

operator de tip asociativ cum ar fi: sumă, minim, maxim, produs cartezian, joncțiune care se aplică la o secvență de articole.

Modelul arhitecturii pentru sisteme asincrone cu partajarea memoriei constă dintr-un număr fix de procesare ce pot accesa o memorie comună, dar care nu au un ceas comun. În acest caz dacă două procesoare accesează aceeași locație de memorie simultan, accesul se face într-o ordine arbitrară. Implementarea programului pe o astfel de arhitectură se face prin împărțirea instrucțiunilor din program între procesoare. În plus, trebuie asigurat un desfășurător al execuțiilor pentru fiecare procesor, astfel încât rezultatul să fie execuția întregului program. În acest mod două instrucțiuni nu pot fi executate în paralel dacă una modifică o variabilă care este utilizată de cealaltă instrucțiune. Efectul execuției într-o arhitectură multiprocesor este același cu execuția individuală a fiecărei instrucțiuni. Descrierea structurii unui program este:

program →	**Program**	nume_program
	declarare	secțiune_declarații
	always	secțiune_always
	inițial	secțiune_inițializare
	atribuiri	secțiune_atribuiri
	end.	

1.1 Instrucțiunea de atribuire

instr_atribuire → componentă_atribuire
{|| componentă_atribuire}
componentă_atribuire → atribuire_enumerată
| atribuire cuantificată

O instrucțiune de atribuire constă din una sau mai multe componente de atribuire, separate prin ||. Există două tipuri de componente: tipul *enumerat* în care variabilele cărora li se atribuie expresii sunt numite explicit și tipul *cuantificat* în care variabilele care vor fi atribuite și expresiile sunt obținute prin particularizarea unei atribuiri generice cu toate valorile posibile. Din punct de vedere operațional execuția instrucțiunilor de atribuire se poate realiza prin execuția simultană și independentă a fiecărei componente de atribuire.

atribuire_enumerată →
listă_variabile:= listă_expresii
listă_variabile →
variabilă {, variabilă}
listă_expresii →
listă_expresie_simplă
| listă_expresie_condițională
listă_expresie simplă →
expresie {, expresie }
listă_expresie_condițională →
listă_expresie_simplă **if** expresie_booleană
{ ~ listă_expresie_simplă
if expresie_booleană }

Exemple:

a) pentru interschimbare x cu y

x,y := y,x

b) calcularea sumei unui vector A[I] de dimensiune N:

sum, i := sum + A[I], i + 1 if i < N

atribuire_cuantificată →
<|| cuantificare instr_atribuire>
cuantificare →
listă_variabile : expresie booleană ::

Variabilele din *lista_de_variabile* din definirea cuantificării se numesc *variabile de cuantificare*. Cuantificatorul are un domeniu de existență delimitat de "<" și ">". Expresia booleană poate denumi variabile declarate în program, constante și variabile de cuantificare, indiferent de domeniu de existență a acestora. O instrucțiune de atribuire cuantificată desemnează zero sau mai multe componente de atribuire care sunt obținute din instrucțiunea de atribuire prin înlocuirea variabilelor de cuantificare. Trebuie asigurată obținerea unui număr finit de componente. În cazul în care nu există nici o componentă se obține instrucțiunea nulă.

Exemplu:

a) Fie vectorii A [0 ... N] și B [0 ... N] de întregi. Atribuim lui A[i] maxim (A[i], B[i]) pentru toți i $0 \le i \le N$

<| i: 0≤ i ≤ N :: A[i] := max (A[i], B[i])>

b) Obținerea matricei unitate U[0...N, 0...N]

<|| i,j : 0 ≤ i ≤ N ∧ 0 ≤ j ≤ N ::
U[i,j] :=0 if i ≠ j
~ 1 if i = j >

sau

<|| i,j : 0 ≤ i ≤ N ∧ 0 ≤ j ≤ N ∧ i ≠ j ::
U[i,j] :=0>
|| <| i : 0 ≤ i ≤ N :: U[i,i] :=1>

1.2 Secțiunea atribuire

secțiune_atribuire→ listă_instrucțiuni
listă_instrucțiuni →
instrucțiune { | instrucțiune }
instrucțiune →
instrucțiune atribuire |
listă_instr_cuantificate
listă_instr_cuantificate →
<| cuantificare listă_instrucțiuni>

O listă de instrucțiuni cuantificate specifică o mulțime de instrucțiuni obținută prin particularizarea listei de instrucțiuni cu valori corespunzătoare pentru variabile. O condiție care se pune este aceea ca numărul instrucțiunilor să fie finită, iar expresiile de tip boolean ce cuantifică anumite instrucțiuni nu trebuie să se modifice pe parcursul execuției. Aceste restricții sunt impuse pentru a nu schimba numărul instrucțiunilor din cadrul programului pe parcursul execuției sale.

1.3 Secțiunea inițializare

Sintaxa este asemănătoare cu cea din secțiunea de atribuire, cu deosebirea că := este înlocuit cu =. În această secțiune se definesc valorile inițiale pentru anumite variabile. Ecuațiile care definesc valorile inițiale nu trebuie să fie circulare, sau cu alte cuvinte se îndeplinesc condițiile:

1. variabilele apar cel mult odată în partea stângă a unei ecuații;
2. există o ordine a ecuațiilor astfel încât orice variabilă care este cuantificată este fie o variabilă fixă fie este una care apare în stânga unei ecuații deja definite;
3. există o ordine a ecuațiilor astfel încât toate variabilele care apar în partea dreaptă a unei ecuații au apărut în partea stângă într-o ecuație precedentă.

1.4 Secțiunea always

Această secțiune este folosită pentru a defini anumite variabile din program funcție de alte variabile. Sintaxa este identică cu cea a secțiunii de inițializare. O variabilă care apare în partea stângă a unei ecuații se numește *variabilă transparentă*. O variabilă transparentă este funcție de variabile netransparente și, deci, nu apare în partea stângă a unei inițializări sau atribuiri, dar poate apare în partea dreaptă. Este evident că această secțiune nu este necesară, ea putând fi rescrisă în cadrul secțiunii de atribuire cu inițializările corespunzătoare. Totuși ea este utilă deoarece în ea se pot defini o serie de invarianți ai programului.

2. Implementarea pentru diferite arhitecturi

2.1 Implemetarea în arhitecturi asincrone cu partajarea memoriei

Un calculator asincron cu partajarea memoriei este compus dintr-un număr fix de procesoare și un număr fix de memorii. Asociat cu fiecare memorie este un set de procesoare care pot citi sau scrie din aceasta. Implementarea unui program pentru o astfel de structură presupune:

1.alocarea fiecărei instrucțiuni din program pentru un procesor;
2.alocarea fiecărei variabile pentru o memorie;
3.specificarea controlului execuției pentru fiecare procesor.

Implementarea trebuie să țină cont de următoarele restricții: toate variabilele din partea stângă a fiecărei instrucțiuni alocate unui procesor se află în memorii care pot fi scrise de procesor; toate variabilele din partea dreaptă sunt în memorii care pot fi citite de procesor.

2.2 Implementarea în arhitecturi distribuite

Un sistem distribuit este compus dintr-un număr fix de procesoare, un număr fix de canale și o memorie locală pentru fiecare procesor. O memorie este *locală* pentru un procesor dacă doar el poate să o citească sau să o scrie. Canalele transmit mesajele în ordinea în care sunt primite. Pentru fiecare canal există un singur procesor care transmite mesaje și un singur procesor ce recepționează mesajele din canal. Asociat cu fiecare canal există un buffer. Singura acțiune pe care o poate realiza un procesor ce transmite este să depună în buffer informația, dacă acesta nu este plin, iar singura operațiune pe care o execută un procesor ce primește mesaje este preluarea unui mesaj din buffer, dacă nu este gol.

Implementarea unui program pentru un sistem distribuit este asemănătoare cu cea din arhitectura asincronă, cu singura diferență că variabilele pot fi alocate fie în memoria unui procesor, fie pentru un canal. Există și condiții suplimentare pentru această structură:

1. cel mult o variabilă poate fi alocată pentru fiecare canal, iar variabila este de tip șir.
2. variabila alocată unui canal apare în instrucțiunile a două procesoare (exact două).

Instrucțiunile unui procesor modifică variabila prin adăugarea unui nou articol, dacă lungimea secvenței nu depășește o valuare, iar instrucțiunile celuilalt procesor modifică variabila prin ștergerea unui articol din virful secvenței, dacă secvența nu este vidă.

2.3 Implementarea în arhitecturi sincrone

Un calculator paralel sincron are aceeași structură procesor-memorie ca la un calculator asincron cu partajarea memoriei. În plus, procesoarele au un ceas comun, la fiecare pas al acestuia, fiecare procesor execută o instrucțiune. Un număr arbitrar de procesoare pot citi o celulă de memorie în același pas. Scrierea unei aceeași celule de memorie de către mai multe procesoare este permisă, dar numai dacă toate procesoarele înscriu aceeași valoare. Nu se permite citirea și scrierea simultană a aceleiași celule de memorie. Un număr arbitrar de procesoare poate coopera în execuția în paralel a unei instrucțiuni. Implementarea într-o astfel de structură este relativ complexă, de aceea, pentru prezentare, vom introduce o restricție. Implementarea impune execuția unei singure instrucțiuni din program la un moment dat, indiferent de numărul de procesoare disponibile. Doar după ce toate procesoarele au executat instrucțiunea se trece la următoarea instrucțiune din program. Dacă un procesor nu are o instrucțiune atribuită, rămâne în stare de așteptare. Implementarea presupune:

- o descriere a modului în care operațiile din fiecare instrucțiune se execută de către procesoare;
- o alocare a fiecărei variabile din program în memorie;
- specificarea unui mod comun al controlului programului pentru toate procesoarele.

3. Modele schemă de program

Un model schemă pentru un program constă din programul prelucrat și implementarea asociată pentru o anumită arhitectură. În continuare se

prezintă o serie de strategii de execuție pentru câteva tipuri de arhitecturi.

3.1 Modelul read-only

Se spune că un program implementat pe o structură asincronă cu partajarea memoriei este un model *read-only* dacă fiecare variabilă din program este mo-dificată de cel mult un procesor. Practic, fiecare variabilă din program apare în partea stângă a instrucțiunilor pentru cel mult un procesor. Programele dezvoltate în acest model pot fi executate doar pe structuri în care fiecare memorie este scrisă de cel mult un procesor. Se pot distinge două categorii de modele read-only, funcție de tipul instrucțiunilor:

- **atomicitate fină.** Partea dreaptă a atribuirii pentru un procesor folosește cel mult o variabilă modificată de alte procesoare. Aceasta se poate implementa direct în arhitecturi în care, pentru o acțiune atomică, un procesor poate accesa cel mult o locație de memorie scrisă de alte procesoare.
- **atomicitate grosieră.** Partea dreapta a atribuirii pentru un procesor folosește un număr arbitrar de variabile modificate de alte procesoare. Aceasta se poate implementa direct în arhitecturi în care, într-o acțiune atomică, un procesor poate accesa un număr arbitrar de locații de memorie care pot fi scrise de alte procesoare.

3.2 Modelul cu partajarea variabilelor

Pentru un program dat, o variabilă se numește *locală* pentru un procesor, dacă variabila respectivă apare doar în instrucțiunile alocate procesorului respectiv. Un model este cu partajarea variabilelor dacă fiecare instrucțiune utilizează cel mult o variabilă care nu e locală. Aceste variabile pot apare fie în partea stângă, fie în partea dreaptă a instrucțiunii.

Implementarea acestui model pentru arhitecturi distribuite impune utilizarea de *lock-uri* pentru variabile partajate.
Un lock poate fi atribuit la un moment dat cel mult unui procesor. Un procesor poate executa o instrucțiune care conține o variabilă comună dacă deține lock-ul pentru acea variabilă. Pentru a implementa lock-urile se poate seta o mulțime de procesoare a căror sarcină este gestionarea lock-urilor pentru variabilele comune. Pentru aceasta, variabila comună este alocată în memoria locală a unuia din aceste procesoare notat A. Pentru a executa o instrucțiune care implică o variabilă comună, un procesor oarecare, B, va trebui să parcurgă următoarele acțiuni:

- procesorul B transmite un mesaj către procesorul A, de cerere a lock-ului;
- după recepționarea confirmării, procesorul B execută instrucțiunea iar la sfârșit transmite cu mesaj de eliberare a lock-ului și își continuă execuția.

Implementarea pentru o arhitectură asincronă cu partajarea memoriei se realizează în mod similar. O instrucțiune ce utilizează o variabilă comună va fi implementată prin blocarea variabilei comune, execuția atribuirii și eliberarea variabilei. Cazul mai general al utilizării mai multor variabile comune în cadrul aceleiași instrucțiuni presupune luarea de precauții suplimentare privind evitarea deadlock-ului.

3.3 Modelul ecuațional

Un program care respectă modelul ecuațional constă doar din secțiunea de declarare și secțiunea always. Acest model poate fi implementat în toate arhitecturile prezentate anterior.
Pentru a implementa modelul pentru un calculator asincron cu partajarea memoriei, se ordonează ecuațiile într-o secvență de instrucțiuni în care fiecare variabilă din partea dreaptă a ecauției este în partea stângă a unei ecuații care apare înainte în secvența de instruc-

ţiuni. În continuare, secvenţa de ecuaţii se transformă într-o serie de atribuiri. În acest caz, valorile variabilelor, după execuţia atribuirilor vor satisface mulţimea ecuaţiilor. Implementarea va aloca fiecare atribuire din secvenţă câte unui procesor şi se specifică ordinea de execuţie a atribuirilor, pentru a păstra ordinea din secvenţa iniţială.
Pentru a implementa programul pe o arhitectură distribuită fiecare variabilă din partea stângă trebuie să apară doar într-o singură atribuire. Fie x o variabilă ce apare în partea stângă a unei atribuiri pentru procesorul A. Variabilă x va fi alocată în memoria locală a procesorului A, iar pentru restul procesorelor B se introduce o copie locală B.x. După ce se calculează valoarea lui x, procesorul A transmite valoarea lui x către fiecare procesor B pentru a fi depus în B.x. Toate apariţiile lui x din instrucţiunile alocate procesorului B vor fi înlocuite cu B.x. În acest fel, în instrucţiunile procesorului B apar doar variabile locale.
Pentru arhitecturi sincrone se parcurg aceleaşi transformări descrise anterior. Toate procesorele vor executa secvenţe de instrucţiuni, câte una la un moment dat. Implementarea va trebui să specifice şi modul în care procesoarele cooperează în execuţia fiecărei instrucţiuni.

4. Comunicaţia proceselor

Prelucrarea unui şir de articole de intrare pentru a produce un şir de articole de ieşire este un model de programare important. Un proces este util pentru acest caz. Procesul poate primi mesaje, pe canale conectate la porturile de intrare, pe care le prelucrează local şi transmite mesaje pe canale conectate la porturile de ieşire. Multe programe pot fi structurate ca reţele de procese ce comunică mesaje. Aceste programe pot fi implementate deosebit de eficient în calculatoare asincrone, dar principalul avantaj este că interfaţă între două procese este o secvenţă de mesaje prin care comunică. Interfaţa este foarte strictă, ceea ce impune o disciplină deosebită în construcţia modulară a programelor.

4.1 Comunicaţie asincronă punct la punct

În comunicaţia asincronă punct la punct, un proces transmite un mesaj către altul prin depunerea acestuia într-un canal, direcţionat de la primul proces către cel de-al doilea. Mesajul poate fi depus în canal dacă există spaţiu liber în bufferul asociat canalului. În modelul cu buffer nelimitat se presupune că fiecare canal are un spaţiu disponibil nelimitat pentru a păstra mesajele, caz în care procesul transmiţător nu va fi impiedicat să furnizeze un nou mesaj datorită lipsei de spaţiu. În modelul cu buffer limitat sunt fixate limite pentru spaţiul pe care îl are la dispoziţie fiecare canal, caz în care procesul transmiţător va trebui să aştepte, dacă bufferul este plin. În ambele cazuri, procesul receptor poate prelua un mesaj doar dacă în canal există unul. Un mesaj este eliminat din canal după ce a fost recepţionat de proces.

4.2 Comunicaţie sincronă

În comunicaţia sincronă între procese, un proces este transmiţător şi restul sunt receptori. O comunicaţie poate avea loc doar dacă toate procesele care participă la comunicaţie aşteaptă comunicare, fenomen numit şi *rendezvous*. Efectul comunicaţiei este ca toţi receptorii să primească simultan mesajul transmis. În continuare toate procesele pot prelucra calculele care urmează după *rendezvous*.
Un model al unei astfel de comunicaţii este următorul: fie x o variabilă a transmiţătorului a cărei valoare reprezintă mesajul de transmis; x = empty

înseamnă că transmiţătorul nu are mesaje de transmis. Fie N receptori şi y[i] cu $0 \le i \le N$ variabilele în care va fi primită valoarea mesajului ce trebuie recepţionat; y[i] = empty semnifică faptul că receptorul i este în stare de aşteptare mesaj. Comunicaţia sincronă poate fi sintetizată prin următoarea instrucţiune:

$<\| \; i : 0 \le i < N ::$
$y[i] := x$
$\quad if < \wedge \; i \le i < N :: y[i] = empty \wedge x \ne empty >$
$\| \; x := empty$
$\quad if < \wedge \; i \le i < N :: y[i] = empty \wedge x \ne empty$

În cadrul unei comunicaţii sincrone punct la punct există un singur transmiţător şi un singur receptor. Fie x şi y variabilele asociate celor doi participanţi; $x \ne empty$ semnifică faptul că transmiţătorul aşteaptă să emită, iar $y \ne empty$ că receptorul aşteaptă să primească mesajul. Comunicaţia sincronă este realizată de execuţia instrucţiunii:

$x, y := empty, x \quad if\ x \ne empty \wedge y = empty.$

Comunicaţia sincronă poate fi utilizată pentru a simula comunicaţii asincrone. Fie un program în care un transmiţător şi un receptor comunică asincron folosind un canal. Canalul poate fi privit ca un proces de comunicare sincronă cu transmiţătorul şi receptorul. Fie x şi y variabilele celor două procese şi u o variabilă asociată canalului; u = empty semnifică faptul că mesajele transmise prin canal au fost recepţionate, iar $u \ne empty$ indică faptul că ultimul mesaj transmis nu a fost încă recepţionat. În acest caz folosind comunicaţia sincronă se poate implementa operaţia de transmisie şi recepţie astfel:

$x, u := empty, x$
$\quad if\ x \ne empty \wedge u = empty$ [transmisie]
$|\ u, y := empty, u$
$\quad if\ u \ne empty \wedge y = empty$ [recepţie]

Concluzii

Modelul avansat anterior permite analiza şi dezvoltarea programelor fără a se ţine cont de arhitectura particulară a sistemului de calcul. Se obţin astfel programe independente de structura hardware care ulterior pot fi implementate pe o arhitectură concretă. O schimbare ulterioară a arhitecturii pe care se doreşte implementarea nu impune reluarea analizei şi dezvoltării programului din faza iniţială, ci doar schimbarea modului de implementare a programului obţinut anterior funcţie de tipul şi restricţiile arhitecturii paralele.