

# REZOLVAREA PROBLEMEI ALOCĂRII PROCESELOR ÎN CONTEXTUL UNOR ARHITECTURI PARALELE CU AJUTORUL ALGORITMIILOR GENETICI

mat. Cristina Geangălă

Universitatea București

**Rezumat:** Performanțele unui program paralel, executat pe o arhitectură cu o topologie bine definită, depind în mare măsură de o bună plasare a proceselor componente pe procesoarele fizice existente. Această repartizare ce trebuie făcută, în general, de către programator este, de cele mai multe ori, dependentă de aplicația în sine și de calculatorul utilizat.

În cazul unor sisteme ce conțin un număr mare de procesoare, controlul lor este dificil, prin urmare, definirea de către utilizator a modului de amplasare poate conduce la soluții mult diferite de cea optimă. Aceasta a fost motivul pentru care s-au căutat algoritmi care să realizeze o alocare automată, în acest fel obținându-se și portabilitatea programelor pe calculatoare cu diverse arhitecturi. Articolul de față formulează problema alocării proceselor și investighează algoritmul genetic ca metodă potențial disponibilă în rezolvarea acesteia.

Cuvinte cheie: proces, procesor, repartizare optimală, alocare automată, algoritmi genetici, modele paralele pentru algoritmi genetici.

## 1. Modelarea problemei

Considerăm un sistem format dintr-o mulțime  $T$  de  $n$  procesoare  $\{t_1, t_2, \dots, t_n\}$ , care comunică între ele prin intermediul unei rețele de interconexiune statică. Considerăm, de asemenea, un program compus dintr-o multime  $P$  de  $k$  procese  $\{p_1, p_2, \dots, p_k\}$ , ce comunică între ele prin schimb de mesaje. Alocarea proceselor pe procesoarele fizice poate fi descrisă formal printr-o funcție definită pe mulțimea proceselor cu valori în mulțimea procesoarelor:

$\Pi : P \rightarrow T$ . Observăm că, a priori, există  $n^k$  posibilități de repartizare, considerând că, la o alocare, unui proces îi corespunde un singur procesor.

O căutare exhaustivă a soluțiilor poate fi reprezentată printr-un arbore de căutare în care o modalitate de alocare constă într-un drum mergând de la rădăcină, la o frunză (figura 1).

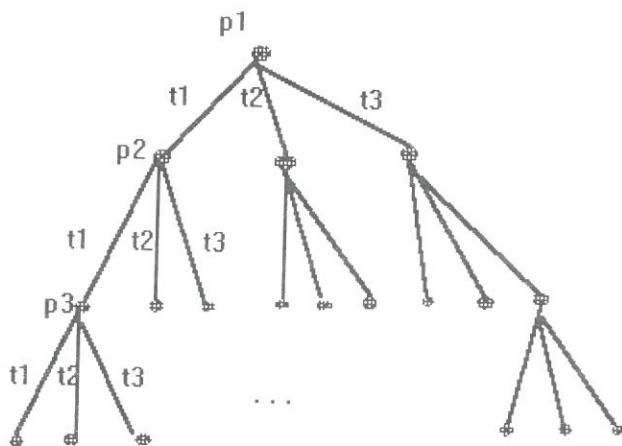


Figura 1. Arbore de căutare pentru  $k=3$  și  $n=3$

Pentru aceste posibilități este necesară introducerea unui criteriu de optimalitate.

În acest sens, se definește funcția cost  $f : \Pi \rightarrow \mathbb{R}$  care asociază o valoare fiecărui amplasament.

O repartizare optimală este cea care minimizează funcția cost. În general, optimizarea constă în căutarea minimizării timpului total de execuție al programului. Acesta depinde de timpul de execuție al diverselor procese și de timpul de comunicare între procese. Se pot utiliza și alte criterii, cum ar fi gradul de utilizare al resurselor

(procesor, memorie, legături de comunicație) sau gradul de toleranță la erori.

Problema, în general, este considerată a fi NP completă. Există însă cazuri particulare în care ea poate fi redusă la o problema polinomială (de exemplu, sisteme cu două procesoare sau sisteme în care procesoarele sunt identice, pe fiecare executându-se cel mult două procese, iar numărul total al proceselor este cel mult dublul numărului de procesoare).

Pentru rezolvarea acestei probleme s-a încercat aplicarea unor algoritmi proveniți din programarea matematică. Acești algoritmi furnizează soluții

optime, dar într-un timp relativ ridicat. Pentru ameliorarea timpului de execuție au fost propuse

## 2. Algoritmi genetici

Algoritmii genetici au fost introdusi de Holland la începutul anilor '70, fiind inspirați de evoluția biologică a speciilor. Aplicații ale acestor algoritmi au fost utilizate în Inteligența Artificială și în optimizarea combinatorie. Un obstacol mare în dezvoltarea lor l-a reprezentat costul ridicat de execuție. Apariția arhitecturilor paralele a dat posibilitatea realizării unor versiuni paralele ale acestor algoritmi pentru probleme diverse, cum ar fi : problema comis-voiajorului, optimizarea conexiunilor în rețele de neuroni, sisteme de clasificare.

### 2.1 Principii

Se consideră :

- un spațiu de căutare  $S$  de dimensiune  $MN$  : fiecare punct din acest spațiu este reprezentat de un vector (individ) de dimensiune  $N$ ;
- o funcție obiectiv  $f: \Sigma \rightarrow \mathbb{R}$  ce asociază o valoare reală fiecărui punct din  $\Sigma$ ;
- o mulțime inițială de vectori (indivizi) formează o populație inițială. Fiecare individ reprezintă o soluție potențială.

Pentru îmbogațirea populației cu alți indivizi pot fi aplicări diferenți operatori: cei mai cunoscuți sunt cei de *crossover* și de *mutație*.

*Principiul fundamental al algoritmilor genetici este : "probabilitatea de selecție a unui individ este cu atât mai mare cu cât individul este mai bun".*

Prin îmbogațirea populației cu noi indivizi, caracteristicile cele mai puternice ale celor din vechea generație se propagă, deci supraviețuiesc elementele cele mai bine adaptate.

Schema unui algoritm genetic este următoarea :

1. **Evaluare** : fiecare individ i se asociază câte un cost.

2. Repetă

2.1. **Selecție**: se stabilesc perechi de indivizi care vor da naștere unei noi populații. Criteriul de selecție favorizează cei mai buni indivizi.

alte tehnici, între care algoritmii genetici.

2.2. *Se aplică operatorii genetici tuturor perechilor selecționate.*

2.3. **Evaluare** : fiecare nou individ i se asociază un cost.

2.4. *Înlocuire*: se obține o nouă populație înlocuind indivizii în ideea de a-i păstra pe cei mai buni până când este găsită o soluție "bună".

### 2.2 Codificarea problemei alocării în ideea utilizării unor algoritmi genetici

Un program paralel poate fi modelat printr-un graf  $G_p = (V_p, E_p)$  în care nodurile reprezintă procesele, iar muchiile - comunicațiile între ele.

De asemenea, o arhitectură paralelă poate fi modelată printr-un graf neorientat  $G_t = (V_t, E_t)$  în care nodurile reprezintă procesoarele și muchiile - legăturile fizice dintre ele

Se vor folosi următoarele notații :

$M$  : numărul de procese,  $M = |V_p|$

$N$  : numărul de procesoare ale arhitecturii  $N = |V_t|$

$e_{ij}$  : costul execuției procesului  $p_i$  și  $p_j$

$c_{ij}$  : costul comunicării între procesele  $p_i$  și  $p_j$

$d_{kj}$  : distanța între procesoarele  $t_k$  și  $t_l$ , adică numărul minim de legături ce alcătuiesc un drum între aceste procesoare.

Se pot utiliza două criterii de optimalitate:

- minimizarea sumei totale  $C$  a costurilor comunicării între procese. Acest cost poate fi măsurat prin produsul costurilor de comunicație între toate perechile de procese ( $i, j$ ) și costul transmiterii mesajelor între procesoarele ( $P(i), P(j)$ )

$$MIN(C) = MIN \left\{ \sum_{i,j \in V_p} c_{ij} d_{P(i)P(j)} \right\}$$

- echilibrarea încărcării procesoarelor : aceasta revine la minimizarea diferenței între numărul de procese ce rulează pe fiecare procesor. Măsura calitativă utilizată este dispersia acestei diferențe:

$$MIN(V) = MIN \left\{ \frac{1}{N} \sum_{k=1}^N L_k^2 - L^2 \right\} \text{ unde } L = \frac{\sum_{i=1}^M e_i}{N} \text{ si } L_k = \sum_{\substack{i=j \\ \Pi(i)=k}} e_i$$

În final, funcția cost pe care o alegem va fi  $f = C + w \cdot W$ , unde  $w$  este o constantă și măsoară importanța relativă a celor două criterii

Alegerea unei valori potrivite pentru  $w$  depinde de arhitectura folosită și, în principal, de raportul între costul comunicării și costul execuției asociate arhitecturii. Valori mici pentru  $w$  sugerează utilizarea unei arhitecturi de tip monoprocesor, în acest caz costurile de comunicație fiind neglijabile.

Pentru rezolvarea problemei comunicării au fost folosiți algoritmi genetici cu următoarea codificare [4]:

*Se presupune că trebuie plasat un graf  $G$  cu  $N$  vârfuri (procese) pe  $M$  procesoare: fiecare din aceste procesoare este etichetat printr-un simbol*

enunțate mai sus. Ea reprezintă contribuția costului de comunicare la cel de distribuire a încărcării procesoarelor.

(de exemplu, un număr întreg între 1 și  $M$ ). O amplasare este reprezentată printr-un vector de mărime  $N$  având ca elemente aceste simboluri. Procesului  $i$  din program îi este asociat elementul  $i$  al vectorului. Valoarea acestui element va furniza numărul procesorului asignat.

Aplicarea operatorului de mutație schimbă amplasarea unui proces pe un alt procesor. Aplicarea operatorului de crossover conduce la combinarea a două amplasări.

În figura 2, se prezintă un exemplu pentru o astfel de amplasare :

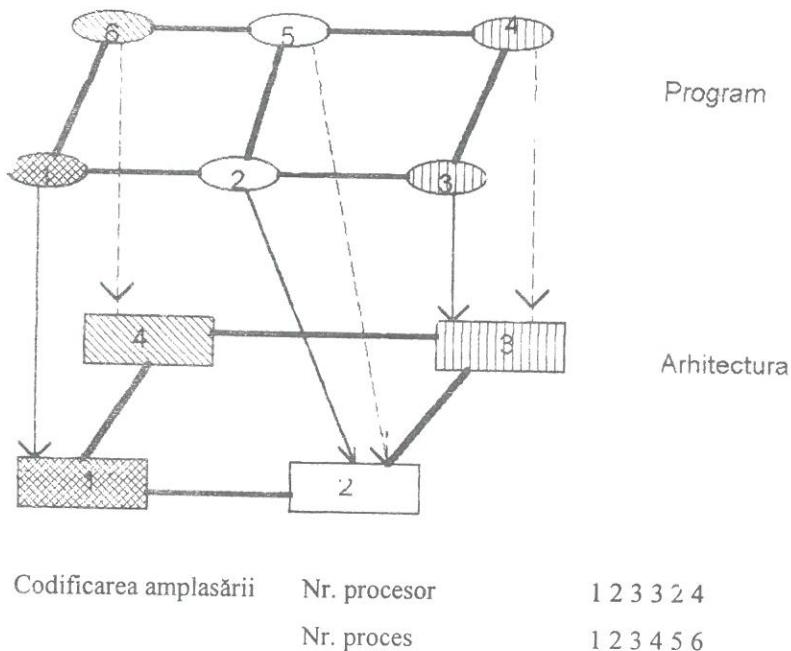


Figura 2. Un exemplu de codificare a amplasării

## 2.3 Modele paralele pentru algoritmii genetici

Algoritmii genetici standard posedă un cost de execuție ridicat. De aceea, s-au încercat numeroase variante de paraleлизare a lor. Două tipuri de soluții au fost găsite:

- modelul paralel standard
- modelul cu descompunere

### 2.3.1. Modelul standard

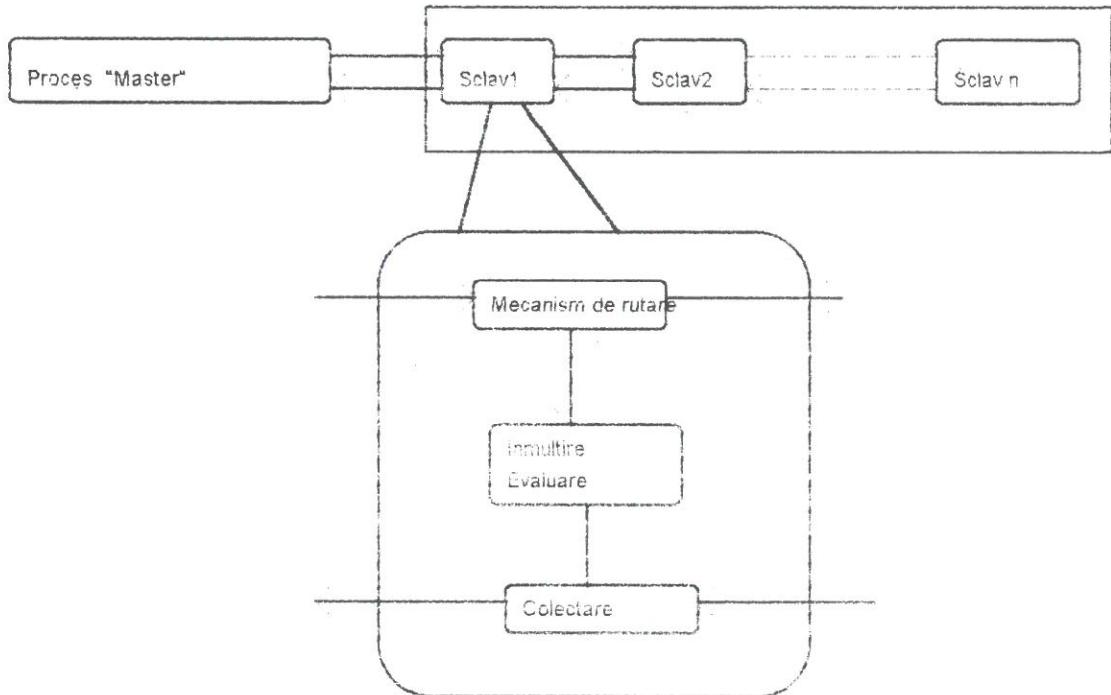
Acest model presupune utilizarea algoritmului standard pentru efectuarea etapelor de evaluare, selecție și înmulțire în paralel. Pentru etapa de selecție este nevoie de un graf complet pentru ca orice pereche de indivizi aparținând populației poate fi un candidat potențial.

Acest model nu este flexibil deoarece costul comunicării crește exponențial, în funcție de mărimea populației. Din această cauză el este greu

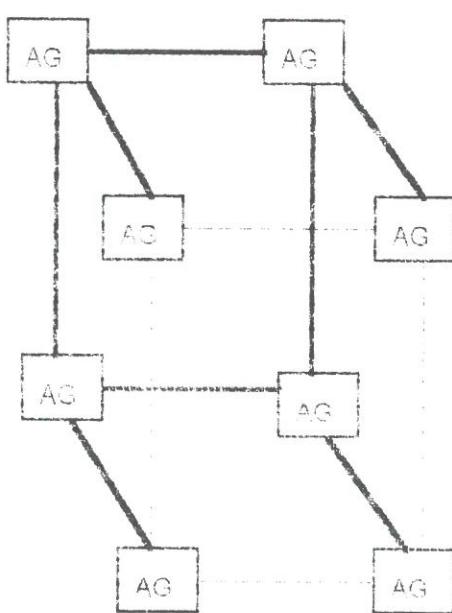
de implementat pe sistemele paralele cu memorie distribuită, pentru care un cost scăzut al comunicării este extrem de important în obținerea unor performanțe bune.

În consecință, etapa de selecție este, în general, efectuată secvențial. O implementare a acestui algoritm a fost realizată pe o arhitectură de

tip *pipeline*. Un procesor "master" efectuează selecția perechilor de indivizi și pe care le transmite procesoarelor "slave". În momentul în care un procesor "slave" primește opareche de indivizi, el aplică unul din cei doi operatori, evaluează noii indivizi și îi retransmite procesului "master". Acesta efectuează înlocuirea populației curente și reîncepe același proces cu noua populație.



**Figura 3.** Implementare pentru o arhitectură de tip pipeline



**Figura 4.** Modelul cu descompunere

### 2.3.2. Modelul cu descompunere

Acest model (vezi figura 4) constă în împărțirea populației în subpopulații de aceeași mărime. Fiecare procesor execută algoritmul standard asupra subpopulației care îi este afectată. Între aceste subpopulații se efectuează regulat un schimb de indivizi. Cei mai buni indivizi receptionați îi vor înlocui pe cei mai slabii ai populației locale. Parametrii algoritmului sunt: vecinătatea unui procesor, frecvența schimburilor, numărul de indivizi schimbați.

Deficiența acestui model este că paralelismul implicit al algoritmilor genetici nu este exploatat în totalitate, deoarece asupra subpopulațiilor instrucțiunile aplicate sunt secvențiale.

Modelul este folosit, în general, în cazul în care numărul procesoarelor disponibile este mult mai mic decât mărimea populației.

Studiile efectuate au arătat ca împărțirea populației în subpopulații nu conduce la o degradare a calității rezultatelor și nici a eficacității algoritmilor.

### 2.3.3. Cazul arhitecturilor "masiv paralele"

Acesta este cazul în care numărul de procesoare este relativ ridicat și poate varia în funcție de populația dorită.

Un factor important pentru performanțele unui algoritm genetic este găsirea unui echilibru între explorarea spațiului de căutare și exploatarea celor mai bune soluții. Utilizarea intensă a acestor soluții poate conduce la o convergență prematură a algoritmului.

În cazul algoritmului pe care îl vom descrie selecția se face local, în vecinătatea fiecărui individ. Tipul vecinătății este un parametru al algoritmului. Pentru a evita costuri suplimentare, datorate rutării mesajelor, în general, se restrâng vecinătatea la indivizi direct conectați. Algoritmul este următorul:

1. Se generează *în paralel* o populație aleatoare de indivizi

2. **Evaluare** : Se evaluatează *în paralel* fiecare individ

3. while (număr\_generații <= număr\_maxim\_generații) do

**Selectie** : se primesc *în paralel* indivizii vecini.

**Înmulțire**: fiecare individ se reproduce *în paralel* cu vecinii săi.

**Evaluare**: se evaluatează *în paralel* fiecare individ produs.

**Înlocuire**: se înlocuiește *în paralel* fiecare individ.

## 3. Concluzie

Algoritmii genetici pot fi folosiți cu succes în rezolvarea problemei alocării proceselor în sisteme având arhitecturi paralele. Aceasta și deoarece, prin natura lor, sunt ei însăși paralelizabili. Utilizarea unor algoritmi genetici paraleli poate reduce substanțial timpul de lucru. Exemplificăm această observație printr-un tabel comparativ între un algoritm genetic clasic și cel paralel, prezentat pentru arhitecturi masiv paralele.

	A G secvențial	A G paralel
Selecție	$O(n * n)$	$O(s)$
Crossover	$O(n * t)$	$O(s * t)$
Mutație	$O(n * t)$	$O(s * t)$
Înlocuire	$O(n * \log(n))$	$O(s)$
Total	$O(n^*n + n*t)$	$O(s * t)$

unde : n este mărimea populației

s : dimensiunea vecinătății

t : dimensiunea vectorului reprezentând soluțiile problemei

## Bibliografie

1. **MICHALEWICZ, Z.** : Genetic Algorithms + Data Structures = Evolution Programs, Springer Verlag, Berlin, 1992
2. **TALBI, E. G., T. MUNTEAN, T.** : Evaluation et étude comparatif d'algorithmes d'optimisation combinatoire: application au problème de placement de processus. Rapport de recherche LGI / IMAG, Avril, 1992.
3. **TANESE, R.** : Parallel Genetic Algorithms for a Hypercube. În: Proc. of the 2nd Int. Conf. on Genetic Algorithms, MIT, Cambridge, July, 1987.
4. **TALBI, E.G.** : Allocation des processus sur les architectures parallèles à mémoire distribuée. Thèse, Institut National Polytechnique de Grenoble, Mai 1993.
5. **KRISHNAMUTHY, E.V.** : Parallel Processing - Principles and Practice, Addison Wesley, Reading, Massachusetts, 1989.

