Modalitati de paralelizare a programelor secventiale

Prep. Felician ALECU
Catedra de Informatica Economica, A.S.E. Bucuresti

The main reason of parallelization a sequential program is to run the program faster. The easiest way to parallelize a sequential program is to use a compiler that detects, automatically or based on the compiling directives specified by the user, the parallelism of the program and generates the parallel version with the aid of the interdependencies found in the source code Even in an ideal parallel system it is very difficult to obtain a speedup value equal with the number of processors. Amdahl low says that it is very important to identify the fraction of a program than cannot be parallelized and to minimize it.
If the user decides to manually parallelize his program, he can freely decide which parts have to be parallelized and which not. Also, the user has to explicitly define in the parallel program the communication mechanisms and synchronization methods. One of the most important challenges is to parallelize loops because they usually spend the most CPU time even if the code contained is very small. A loop could by parallelized by distributing iterations among processes. Every process will execute just a subset of the loop iterations range. Usually the code contained by a loop involves arrays whose indices are associated with the loop variable. This is why distributing iterations means dividing arrays and assigning parts/blocks to processes.
Keywords: *Sequential program, parallel program, parallel compilers, parallelization of loops, partial parallelization, block and cyclic distribution.*

Evaluarea performantelor programelor paralele

Prin paralelizarea unui program secvential se urmareste în primul rând obtinerea unui timp de executie cât mai mic comparativ cu timpul secvential de executie. Cel mai important criteriu luat în considerare atunci când se doreste evaluarea performantelor unui program paralel este *accelerarea paralela* care exprima de câte ori programul paralel este mai rapid fata de varianta secventiala. Accelerarea paralela se calculeaza ca raport între timpul secvential de executie si timpul de executie paralela: $S = \frac{T_s}{T_p}$. Valoarea maxima a accelerarii paralele este egala cu numarul de procesoare din sistem. O astfel de valoare poate fi atinsa într-un sistem ideal în care nu exista costuri de comunicare iar procesoarele sunt încarcate echilibrat.

În conformitate cu legea lui Amdahl, chiar si într-un sistem paralel ideal este foarte dificil de obtinut o accelerare paralela egala cu numarul de procesoare datorita faptului ca în cadrul oricarui program exista o fractie a care nu poate fi paralelizata si care trebuie executata secvential. Restul de (1 - a) pasi de calcul se pot executa în paralel pe procesoarele disponibile în sistem. Astfel, timpul paralel de executie si accelerarea paralela devin:

$$T_p = T_s \cdot a + T_s \cdot (1-a)/p$$

$$S = \frac{T_s}{T_p} = \frac{T_s}{T_s \cdot a + T_s \cdot (1-a)/p} = \frac{1}{a + (1-a)/p} = \frac{p}{a \cdot (p-1) + 1}$$

Atunci când $p \to \infty$ avem: $\lim_{p\to\infty} S = \frac{1}{a}$. Din acest motiv, accelerarea maxima care se poate obtine atunci când o fractie a a programului nu poate fi paralelizata este 1/a indiferent de numarul de procesoare din sistem. De exemplu, daca o fractie de 20% dintr-un program nu poate fi paralelizata, timpul paralel de executie si accelerarea paralela vor fi:

$$T_p = T_s \cdot 0.2 + T_s \cdot 0.8/4 = 0.4 \cdot T_s$$

$$S = \frac{T_s}{0.4 \cdot T_s} = \frac{1}{0.4} = 2.5$$

Timpul paralel de executie va fi 40% din cel secvential, ceea ce înseamna ca programul paralel va fi doar de 2.5 ori mai rapid decât cel secvential. Accelerarea maxima care poate fi obtinuta este 1/0.2 = 5, ceea ce semnifica faptul ca timpul paralel de executie nu va fi niciodata mai mic de 20% din cel secvential, indiferent de numarul de procesoare din sistem.

Legea lui Amdahl exprima în mod clar necesitatea minimizarii fractiei a ce nu poate fi paralelizata prin stabilirea unei limite superioare a accelerarii paralele (figura 1).

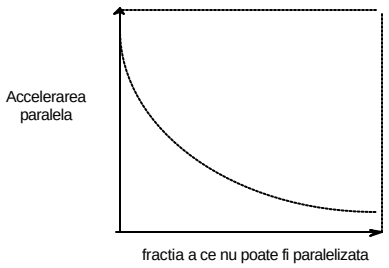

Fig.1. Limita superioara a accelerarii paralele

Atunci când programul paralel este executat pe un sistem real, apar o serie de costuri suplimentare generate de comunicatia dintre procesoare (schimb de date, sincronizare), cât si de încarcarea neechilibrata a procesoarelor. Din acest motiv timpul paralel de executie poate depasi valoarea teoretica calculata.

Paralelizarea programelor secventiale

Cea mai facila modalitate de paralelizare a unui program secvential consta în utilizarea unui compilator cu paralelizare. Acesta detecteaza în mod automat dependentele din cadrul programului secvential si pe baza acestora genereaza versiunea paralela tinând cont de faptul ca pentru a putea rula în paralel doua segmente de program este necesar ca acestea sa fie independente. Compilatorul va fi responsabil atât pentru modul de divizare a programului în segmente care sa poata fi executate concurent, cât si pentru implementarea mecanismelor de comunicare si sincronizare dintre procese. Utilizatorul are un control redus asupra procesului de paralelizare putându-l influenta doar prin folosirea anumitor directive de compilare. Cu toate acestea, dezvoltarea acestor compilatoare se confrunta cu dificultati legate de analiza si detectarea dependentelor în cadrul programelor complexe care prezinta o structura neregulata cum ar fi bucle imbricate, salturi, apeluri de functii si proceduri.

La polul opus se gaseste varianta paralelizarii manuale a programului secvential. În acest caz utilizatorul va decide care parti ale programului vor fi paralelizate si va fi liber sa prevada mecanismele dorite de sincronizare si comunicare între procese. Între aceste situatii extreme se gasesc limbajele de programare paralela care pun la dispozitia utilizatorilor constructii paralele ce faciliteaza paralelizarea unui program secvential.

Paralelizarea buclelor

Paralelizarea buclelor este o operatie de osebit de importanta datorita faptului ca de cele mai multe ori buclele consuma majoritatea timpului procesor desi nu contin decât un volum redus de cod. Paralelizarea buclelor se face prin distribuirea iteratiilor pe procesoarele disponibile în sistem. În acest fel fiecare procesor va executa doar un subset al iteratiilor initiale.

În cazul în care avem doua bucle imbricate însa doar cea interioara este mare consumatoare de resurse procesor, atunci este convenabil sa se paralelizeze doar aceasta bucla, cea exterioara ramânând neschimbata. Acest procedeu se numeste *paralelizare partiala a unei bucle* si presupune sincronizarea datelor în cazul în care bucla exterioara utilizeaza valori modificate de cea interioara. Sincronizarea este necesara din dorinta de a fi siguri ca bucla exterioara acceseaza valorile actualizate. Paralelizarea partiala a unei bucle conduce catre o echilibrare a modului de încarcare a procesoarelor în conditiile în care costurile de comunicare cresc datorita schimbului de informatii dintre procesoare ce are loc la momentul sincronizarii. Uzual, codul din interiorul buclelor utilizeaza masive ale caror indici depind de variabila de ciclare. Din acest motiv, distribuirea iteratiilor pe procesoarele sistemului înseamna de cele mai multe ori divizarea masivelor si alocarea blocurilor obtinute pe procesoarele disponibile.

În continuare vor fi trecute în revista principalele modalitatile de divizare a iteratiilor unei bucle.

- *Distributia pe blocuri* presupune divizarea plajei initiale de ciclare într-un numar de subseturi egal cu numarul de procesoare din sistem. Astfel, în cazul în care o bucla cu 100 de iteratii va fi executata pe un sistem cu 5 procesoare, fiecare procesor va executa 20 de iteratii consecutive. În cazul în care prin împartirea numarului de iteratii, n, la numarul procesoarelor din sistem, p, se obtine un rest r diferit de zero ($n = p \cdot q + r$), acesta va trebui distribuit. Astfel, primele r procesoare vor executa fiecare câte $q+1$ iteratii în timp ce ultimele $p-r$ procesoare vor executa doar câte q iteratii consecutive. Aceasta modalitate de distributie corespunde exprimarii numarului de iteratii în felul urmator:

$$n = r \cdot (q+1) + (p-r) \cdot q .$$

Utilizarea distributiei pe blocuri poate genera o încarcare neechilibrata a procesoarelor. În plus, este posibil ca unele procesoare sa nu aiba atribuite nici un fel de iteratii. Plecând de la plaja de valori ale buclei initiale si de la numarul de procesoare din sistem, putem calcula, folosind una din functiile de mai jos, plaja de valori pentru iteratiile efectuate de fiecare procesor.

```
sub range_ver1(n1, n2, p, rank, Start, End)
      q = (n2 – n1 + 1) / p
      r = MOD(n2 – n1 + 1, p)
      Start = rank * q + n1 + MIN(rank, r)
      End = Start + q – 1
      If r > rank then End = End – 1
end
sub range_ver2(n1, n2, p, rank, Start, End)
      q = (n2 – n1) / p + 1
      Start = MIN(rank * q + n1, n2 + 1)
      End = MIN(Start + q – 1, n2)
end
```

unde *n1*, *n2* – variabile de intrare – valoarea minima si maxima a variabilei de ciclare a buclei initiale; *p* – variabila de intrare – numarul de procesoare din sistem; *rank* – variabila de intrare – numarul procesorului pentru care dorim sa calculam plaja de ciclare; *Start*, *End* – variabile de iesire – valoarea minima, respectiv maxima a plajei de ciclare aferente procesorului dorit. În tabelul 1 se prezinta modul în care functiile *range_ver1* si *range_ver2* distribuie o bucla cu 14 iteratii pe 4 procesoare.

Tabelul 1– Distributie pe blocuri

Iteratie		1	2	3	4	5	6	7	8	9	10	11	12	13	14
Procesor	range_ver1	0	0	0	0	1	1	1	1	2	2	2	3	3	3
	range_ver2	0	0	0	0	1	1	1	1	2	2	2	2	3	3

- *Distributia ciclica* implica alocarea într-o maniera round-robin a iteratiilor buclei initiale pe procesoarele sistemului. Astfel, daca se porneste de la bucla

```
for i = n1, n2
      calcule
end
```

procesorul cu rangul *rk* va executa urmatorul subset de iteratii:

```
for i = n1 + rk, n2, p
      calcule
end
```

În tabelul 2 se prezinta distributia ciclica a unei bucle cu 11 iteratii pe 3 procesoare.

Tabelul 2– Distributie ciclica

Iteratie	1	2	3	4	5	6	7	8	9	10	11
Procesor	0	1	2	0	1	2	0	1	2	0	1

Se observa ca iteratiile oricarui subset nu mai sunt consecutive. Distributia ciclica asigura o mai buna echilibrare a încarcarii procesoarelor decât distributia pe blocuri.

- *Distributia ciclica pe blocuri* combina cele doua metode de distributie prezentate anterior. Astfel, iteratiile buclei originale sunt divizate în blocuri de dimensiune egala care sunt alocate procesoarelor sistemului într-o maniera round-robin. Daca presupunem ca bucla ce se doreste a fi paralelizata este

```
for i = n1, n2
      calcule
end
```

atunci procesorul de rang *rk* va executa urmatorul subset de iteratii

```
for ii = n1 + rk * block, n2, p * block
      for i = ii, MIN(ii + block – 1, n2)
            calcule
      end
end
```

unde *block* semnifica dimensiunea blocului. Modul în care o bucla cu 10 iteratii este distribuita pe 3 procesoare utilizând blocuri de dimensiune 2 este prezentat în tabelul 3.

Tabelul 3– Distributie ciclica pe blocuri

Iteratie	1	2	3	4	5	6	7	8	9	10
Procesor	0	0	1	1	2	2	0	0	1	1

Distributia ciclica pe blocuri combina avantajele celor doua metode din care se compune.

Bucle imbricate

Atunci când se efectueaza paralelizarea unei bucle este recomandabil sa se tina cont de faptul ca accesarea masivelor în ordinea în care sunt stocate în memorie este mult mai eficienta decât accesarea acestora în orice alta ordine. Astfel, daca avem o matrice a carei elemente sunt memorate pe coloane, bucla **B1** va fi mai lenta decât bucla **B2**:

B1

```
for i = 1, n
      for j = 1, n
            a(i, j) = …
      end
end
```

B2

```
for j = 1, n
      for i = 1, n
            a(i, j) = …
      end
end
```

La rândul ei, bucla **B2** poate fi transformata într-o forma paralela în doua moduri:

B21

```
for j = jSTART, jEND
      for i = 1, n
            a(i, j) = …
      end
end
```

B22

```
for j = 1, n
      for i = iSTART, iEND
            a(i, j) = …
      end
end
```

Din aceleasi considerente, varianta **B21** este preferabila variantei **B22** deoarece este mult mai rapida.

Bibliografie

[Ber66] A.J. Bernstein, *Analysis of Programs for Parallel Processing*, IEEE Transactions on Computers, October 1966

[Dod99] Gh. Dodescu, *Parallel Algorithms for Matrix Multiplication*, The Preceedings of the Fourth International Symposium on Economic Informatics, Inforec, 1999

[Dod01] Gh. Dodescu, *Systolic Arrays*, The Preceedings of the Fifth International Symposium on Economic Informatics, Inforec, 2001

[Gre97] J.S. Gray, *Interprocess Communications in UNIX*, Prentice Hall, 1997

[Qui94] M. Quinn, *Parallel Computing*, McGrow-Hill, 1994