MODELE TEORETICE ALE CALCULULUI PARALEL

Lector dr. Bogdan Oancea
Universitatea Artifex, Bucureşti

Lector dr. Răzvan Zota
Academia de Studii Economice, Bucureşti

Rezumat: Acest articol prezintă trei modele teoretice ale calculului paralel: PRAM, LogP şi BSP. În cazul algoritmilor seriali, modelul universal de programare este modelul von Neumann, care face posibilă portarea algoritmilor pe calculatoare cu arhitecturi diferite. În domeniul calculului paralel, nu există un astfel de model general acceptat, care să poată fi folosit la proiectarea algoritmilor pentru un număr mare de arhitecturi paralele. Cele trei modele teoretice prezentate nu întrunesc toate condiţiile de generalitate, dar, în prezent, acestea se bucură de o largă acceptare în domeniul modelării teoretice a calculului paralel.

Cuvinte cheie: calcul paralel, PRAM, LogP, BSP

1. Introducere

Scopul principal al unui model este acela de a îngloba principalele caracteristici ale fenomenului modelat cu un grad înalt de acurateţe, astfel încât, modelul să poată fi utilizat pentru analiză şi predicţie. În domeniul ştiinţei calculatoarelor au fost create nenumărate modele care să ajute la proiectarea unor instrumente eficiente pentru rezolvarea problemelor. Alternativa la modelare o constituie implementarea care poate fi, însă, foarte costisitoare în multe probleme practice. Modelele pot furniza o perspectivă mai amplă, fără a implica şi costuri prea mari, dar ele suferă, uneori, de lipsă de acurateţe.

În cazul algoritmilor seriali, modelul universal de programare este modelul von Neumann. Existenţa acestui model face ca un algoritm să poată fi portat pe mai multe maşini, cu toate că arhitectura procesorului sau organizarea memoriei diferă de la o maşină la alta. Dacă algoritmii ar fi trebuit să fie reproiectaţi pentru fiecare tip nou de procesor, costul software-ului ar fi fost foarte mare, ceea ce ar fi stopat dezvoltarea industriei calculatoarelor.

Domeniul calculului paralel nu a cunoscut un model general acceptat, care să poată fi folosit la proiectarea algoritmilor pentru un număr mare de arhitecturi paralele. Un model general al calculului paralel ar trebui să ţină seama de două aspecte esenţiale.

În primul rând, un astfel de model ar trebui să reflecte caracteristicile specifice calculatoarelor paralele, deci, să fie un model *descriptiv*. În al doilea rând, modelul ar trebui să prezică cu acurateţe performanţele programelor paralele, indiferent de arhitectura pe care acestea sunt executate, deci, să aibă proprietatea de *predictivitate*. Din păcate, aceste două obiective se exclud unul pe altul. Pentru a ne convinge că într-adevăr aşa este, să considerăm numai aspectul conectivităţii dintre procesoarele unui calculator paralel. Un model general ar trebui să înglobeze gradul de conectivitate. Dacă modelul porneşte de la ipoteza că acest grad este mic, atunci algoritmii proiectaţi vor rula bine pe calculatoare care respectă ipoteza de plecare, dar ei pot fi suboptimali dacă vor fi rulaţi pe calculatoare cu grad mare de conectivitate între procesoare deoarece resursele de comunicaţie vor fi subutilizate.

Eforturile depuse pentru găsirea unui model independent de arhitectură al calculului paralel nu au condus, până în prezent, la definitivarea unui astfel de model. Totuşi, s-au conturat mai multe modele de calcul paralel, fiecare având, însă, anumite limitări. În continuare, vom prezenta trei dintre modelele de calcul paralel, care s-au bucurat de atenţie din partea cercetătorilor din domeniu: PRAM, LogP şi BSP.

2. Modelul PRAM

Modelul **PRAM** [6] (Parallel Random Access Machine) este o generalizare a modelului **RAM** (Random Access Machine) de calcul secvenţial. Pe scurt, o maşină PRAM poate fi privită ca fiind alcătuită din p procesoare secvenţiale, care partajează o memorie comună. Fiecare dintre aceste p procesoare pot accesa memoria în mod independent şi într-o perioadă de timp constantă. Comunicaţia între procesoare are loc prin intermediul memoriei comune. Orice operaţie de comunicaţie între procesoare implică o scriere şi o citire a unei locaţii de memorie, operaţie care are loc, de asemenea, într-un timp constant. În figura 1, este prezentată schema de principiu a unei maşini PRAM. După cum se poate observa, toate procesoarele lucrează în mod sincron, sub controlul unui ceas unic.

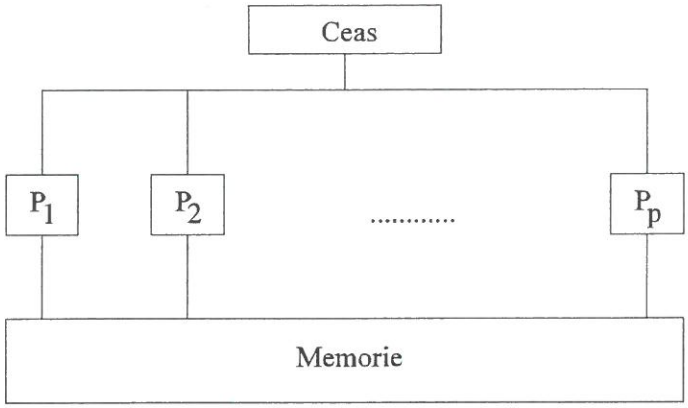

Figura 1. Maşina PRAM

Modelul PRAM este foarte uşor de analizat, dar este foarte greu de implementat practic un calculator care să respecte acest model. De aceea, el are o importanţă mai mult teoretică.

Cele *p* procesoare ale modelului PRAM sunt procesoare obişnuite, dotate cu registre ce pot fi citite/scrise, un *Contor de Instrucţiuni* şi un registru special, care conţine identificatorul procesorului. Faptul că fiecare procesor îşi cunoaşte propriul identificator este foarte important, pentru că, pe baza lui, se pot lua decizii la executarea unor instrucţiuni de salt în interiorul programului. În acest mod, practic, fiecare procesor poate executa propriile secvenţe de instrucţiuni.

Calculul complexităţii unui algoritm scris pentru o maşină PRAM este foarte simplu, întrucât se consideră că fiecare instrucţiune se execută pe durata unui tact de ceas, iar accesul la memoria comună necesită un timp constant. Faptul că mai multe procesoare pot accesa o memorie comună ridică problema concurenţei. În funcţie de modul în care este rezolvată problema concurenţei accesului la memorie, se deosebesc următoarele tipuri de maşini PRAM:

- **CCSC** - *Citire Concurentă Scriere Concurentă*: modelul permite atât scrieri, cât şi citiri simultane, în aceeaşi locaţie de memorie. Se poate presupune că citirea concurentă este rezolvabilă, în schimb, scrierea concurentă în aceeaşi locaţie de memorie ridică problema valorii acelei locaţii de memorie în urma scrierii. În funcţie de modul în care este rezolvată această problemă, avem mai multe subtipuri de modele:
  - **CCSC slab** - scrierea valorii zero de către unul sau mai multe procesoare este permisă, iar valoarea celulei de memorie respective va fi zero. Scrierea oricărei alte valori într-o locaţie de memorie de către mai multe procesoare este interzisă, iar operaţia de scriere se va încheia cu eşec.
  - **CCSC cu scriere comună** - scrierea aceleiaşi valori de către două sau mai multe procesoare în aceeaşi locaţie de memorie este permisă. Încercarea de scriere a unor valori diferite de către mai multe procesoare în aceeaşi locaţie de memorie este interzisă, operaţia de scriere terminându-se cu eşec.
  - **CCSC tolerant** - dacă mai multe procesoare încearcă să scrie simultan, în aceeaşi locaţie de memorie, valoarea celulei respective nu se schimbă. Valoarea unei celule de memorie se poate schimba doar dacă un singur procesor scrie o valoare în acea celulă.
  - **CCSC cu coliziune** - dacă mai multe procesoare încearcă să scrie simultan, în aceeaşi locaţie de memorie, atunci în celula respectivă va fi scris un simbol special de coliziune. Scrierea reuşeşte doar dacă un singur procesor efectuează această operaţie.
  - **CCSC arbitrar** - atunci când mai multe procesoare scriu la aceeaşi adresă în memorie se va alege un singur procesor pentru care operaţia de scriere reuşeşte, iar pentru celelalte, scrierea se încheie cu eşec.
  - **CCSC cu priorităţi** - modul de rezolvare al conflictului de scriere se bazează pe priorităţi: operaţia de scriere va reuşi doar pentru procesorul cu prioritatea cea mai mare. O modalitate de a atribui priorităţi procesoarelor este aceea de a folosi chiar identificatorul procesorului - prioritatea cea mai mare o va avea, de exemplu, procesorul cu identificatorul cel mai mic.
- **CCSE** - *Citire Concurentă Scriere Exclusivă*: se permite existenţa concurenţei la operaţiile de citire, în timp ce scrierea unei locaţii de memorie de mai mult de un procesor este interzisă.
- **CESC** - *Citire Exclusivă Scriere Concurentă*: operaţia de citire a unei locaţii de memorie se poate efectua doar de către un singur procesor, iar la scriere se permite concurenţa, problema de conflict fiind rezolvată similar modelului CCSC.

- **CESE** - *Citire Exclusivă Scriere Exclusivă*: acest model nu permite nici citirea, nici scrierea simultană de către mai multe procesoare a unei locații de memorie, fiind modelul PRAM cel mai apropiat de realitate.

Chiar dacă modelul PRAM este departe de calculatoarele reale, el se dovedește util în proiectarea algoritmilor astfel încât aceştia să exploateze la maxim paralelismul inerent unei probleme. Lipsa, însă, a unui cost asociat operațiilor de comunicație (de acces la memorie) face ca modelul PRAM să aibă, totuşi, o aplicabilitate redusă în practică. Realitatea dovedește că, în cazurile algoritmilor paraleli, operațiile de comunicație reprezintă o mare parte din timpul total de execuție.

3. Modelul LogP

Modelul **LogP** [1] pleacă de la o serie de observații asupra tendințelor de evoluție a arhitecturilor paralele. Autorii modelului constată că, în domeniul calculatoarelor paralele, arhitectura dominantă este cea cu memorie distribuită. Un calculator paralel tinde să fie alcătuit din noduri de calcul, similare cu o stație de lucru, noduri dotate cu procesoare comerciale, care pot susține rate de sute de Mflops (milioane de operații în virgulă mobilă pe secundă) şi memorii locale mari, de ordinul sutelor de MB. Numărul nodurilor de calcul va fi restricționat la ordinul miilor, deci, în proiectarea algoritmilor paraleli va trebui să presupunem că un număr mare de elemente de date vor fi prelucrate de un singur procesor.

Tehnologia rețelelor de interconectare se dezvoltă şi ea, dar se constată că nu poate urma aceeaşi rată de creştere a performanței cu cea a microprocesoarelor sau a memoriilor. În prezent, lărgimea de bandă a rețelelor este mult mai mică decât lărgimea de bandă procesor-memorie. Cu toate că interfața procesor-rețea se îmbunătățeşte, creşterea performanțelor microprocesoarelor este mult mai rapidă. De aceea, va trebui să plecăm de la ipoteza că orice operație de comunicație va implica un interval de timp mare precum şi că lărgimea de bandă a rețelei este limitată.

Esența acestor observații poate fi rezumată în modul următor: dezvoltarea tehnologiilor va conduce la calculatoare formate din cel mult câteva mii de noduri, fiecare conținând procesoare puternice, memorie locală suficient de mare, interconectate prin rețele cu lărgime de bandă limitată şi latență semnificativă.

Plecând de la aceste observații, un grup de cercetători de la Universitatea Berkeley a construit un nou model pentru calculul paralel, pe un calculator cu memorie distribuită, model numit **LogP**. Acest model specifică parametrii de performanță ai rețelei de interconectare, dar nu face nici o supoziție asupra topologiei rețelei, păstrând în acest fel un caracter de generalitate. Parametrii modelului sunt:

- **L**: limita superioară a latenței ce apare în operația de transmitere a unui mesaj conținând un cuvânt de la un procesor sursă la un procesor destinație;
- **o**: *overhead*-ul, definit ca fiind intervalul de timp în care un procesor este ocupat cu transmiterea sau recepționarea unui mesaj, şi nu poate efectua calcule;
- **g**: *gap*, adică intervalul minim de timp între două transmisii sau recepționări succesive de mesaje. Inversul lui g reprezintă lărgimea de bandă a rețelei de interconectare;
- **P**: numărul de procesoare/module de memorie din sistem. În analiza algoritmilor se va presupune că o operație de calcul va dura o unitate de timp.

De asemenea, se presupune că rețeaua de interconetare are o capacitate limitată, astfel că numărul mesajelor care se pot afla în tranzit de la orice procesor la orice procesor nu poate depăşi L/g. Dacă un procesor încearcă să transmită un mesaj care ar duce la depăşirea acestei limite, atunci acel procesor va fi blocat până când mesajul se poate transmite fără depăşirea capacității rețelei.

Alegerea acestor parametri reprezintă un compromis între descrierea cât mai exactă a caracteristicilor unui calculator real şi un cadru rezonabil pentru analiza şi proiectarea algoritmilor. Numărul mic de parametri aleşi nu poate descrie complet o maşină reală, dar, dacă se alege un număr mai mare de parametri, atunci analiza algoritmilor ar deveni mult mai dificilă.

Aceşti parametri nu au o importanță egală în toate situațiile. De exemplu, la algoritmii în care operațiile de comunicație sunt rare, se poate neglija capacitatea rețelei sau lărgimea de bandă. Dacă, în schimb, algoritmul presupune trimiterea unui număr mare de mesaje, acestea pot fi înlănțuite astfel încât timpul total de comunicație va fi dominat de g, iar L poate fi ignorat.

Conform modelului LogP, trimiterea unui mesaj necesită $L+2o$ unități de timp: un *overhead* o la procesorul sursă, apoi L unități de timp pentru ca mesajul să ajungă la destinație şi iar un *overhead* o la procesorul destinație, necesar recepționării mesajului. O secvență de n mesaje necesită un timp $2o+(n-1)g+L$. Procesorului

sursă îi vor trebui o unități de timp pentru a injecta primul mesaj în rețea, apoi, la fiecare g intervale de timp, va trimite celelalte n-1 mesaje. Am făcut presupunerea că $o < g$ pentru a se putea utiliza întreaga lărgime de bandă a rețelei. Ultimul mesaj injectat în rețea va necesita L unități de timp pentru a ajunge la destinație unde va mai fi nevoie de o unități de timp pentru recepționare. În acest mod, calculând timpul necesar comunicației precum și cel necesar calculelor, se poate prezice timpul total de execuție al unui algoritm care poate fi apoi comparat cu cel obținut experimental.

Vom exemplifica concret două situații de analiză a algoritmilor utilizând modelul LogP: distribuția unui mesaj de la un procesor către toate celelalte procesoare și înmulțirea a două matrici.

Ideea de bază la implementarea distribuirii unui mesaj către toate procesoarele din sistem este simplă: îndată ce un procesor recepționează un mesaj, îl va transmite mai departe, cu precauția ca un procesor să nu primească mai mult de un mesaj. Procesorul care inițiază distribuția trimite primul mesaj la momentul *0*. Acest mesaj intră în rețea la momentul *0* și va fi recepționat la destinație la momentul $2o+L$. Între timp, procesorul sursă a mai inițiat transmiterea de mesaje la momentele g, $2g$, ...(presupunând $o \leq g$). Toate procesoarele destinație vor deveni, acum, la rândul lor, rădăcina unui arbore de distribuție mai mic.

În figura 2, este reprezentată situația unui sistem cu 8 procesoare, P_0 fiind nodul inițial al distribuției, iar valorile parametrilor modelului fiind $L=8$, $g=4$, $o=2$. În partea stângă a figurii, este prezentat arborele de distribuție, valorile din interiorul nodurilor reprezentând momentul de timp la care procesorul respectiv a recepționat mesajul. În partea dreaptă a figurii, este prezentată analiza în timp a distribuirii mesajului.

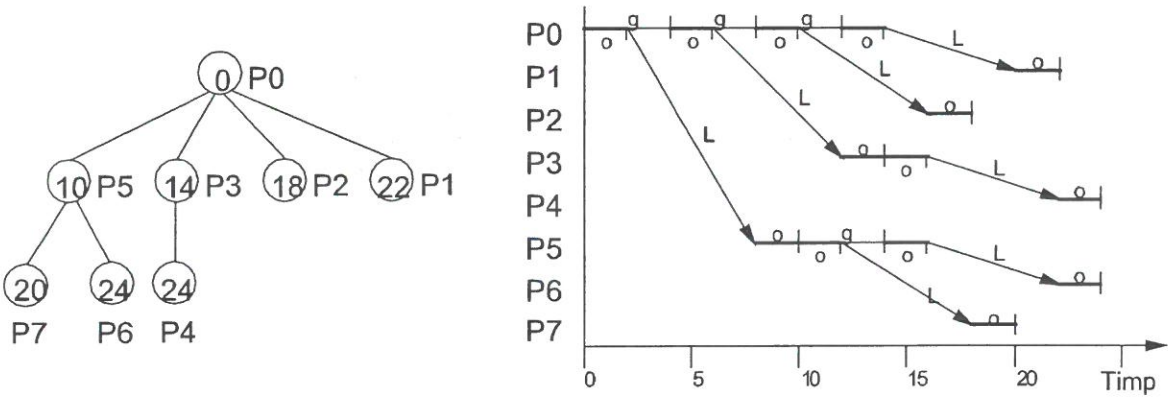

Figura 2. Distribuția unui mesaj și analiza timpului de execuție folosind modelul LogP

Al doilea exemplu pe care îl vom analiza este reprezentat de înmulțirea matricelor. Fie două matrice, A și B de dimensiuni $n \times n$ care sunt partiționate în blocuri $A_{i,j}$ și $B_{i,j}$ cu $0 \leq i, j < \sqrt{p}$. Fiecare bloc de matrice are dimensiunea $(\frac{n}{\sqrt{p}} \times \frac{n}{\sqrt{p}})$ și este atribuit unuia din cele p procesoare care alcătuiesc o grilă bidimensională $\sqrt{p} \times \sqrt{p}$. Procesoarele sunt indexate începând cu $P_{0,0}$ până la $P_{\sqrt{p},\sqrt{p}}$. Procesorul $P_{i,j}$ memorează, inițial, blocurile $A_{i,j}$ și $B_{i,j}$ și are sarcina de a calcula blocul $C_{i,j}$ din matricea rezultat. Calculul submatricei $C_{i,j}$ necesită toate blocurile $A_{i,k}$ și $B_{k,j}$ pentru $0 \leq k < \sqrt{p}$. Pentru a obține toate aceste blocuri de matrice, are loc, mai întâi, o distribuție n-la-n a blocurilor $A_{i,k}$ pe rânduri de procesoare și o operație similară cu blocurile $B_{k,j}$ pe coloanele de procesoare. După ce procesorul $P_{i,j}$ a obținut $A_{i,0}$, $A_{i,1}$, ..., $A_{i,\sqrt{p}-1}$ și $B_{0,j}$, $B_{1,j}$, ..., $B_{\sqrt{p}-1,j}$ poate efectua local înmulțirea între respective l blocuri de matrice pentru a obține $C_{i,j}$.

Pentru a estima timpul de execuție al acestui algoritm, vom presupune că operațiile de adunare sau înmulțire în virgulă mobilă durează t_c unități de timp. La obținerea blocului $C_{i,j}$, fiecare procesor înmulțește $\sqrt{p}$

blocuri de matrice de dimensiuni $(\frac{n}{\sqrt{p}} \times \frac{n}{\sqrt{p}})$. Această operație necesită un timp egal cu $\sqrt{p} \times (n/\sqrt{p})^3 t_c = n^3 t_c / p$.

Algoritmul necesită două operații de distribuție n-la-n pe rânduri și coloane între cele $\sqrt{p}$ procesoare ce formează un rând/coloană. Fiecare mesaj constă în n^2 / p elemente ale matricei, deci, timpul necesar comunicației va fi $2\left(L \log p + g\frac{n^2}{\sqrt{p}}\right)$. În concluzie, timpul total pentru înmulțirea celor două matrice va fi:

$$T_p = \frac{n^3}{p} t_c + 2\left(L \log p + g\frac{n^2}{\sqrt{p}}\right) \quad (1)$$

Acest timp prezis de model poate fi comparat cu timpul obținut experimental pentru a verifica corectitudinea modelului LogP.

Cu toate că prezintă multe avantaje față de alte modele, există și în cazul modelului LogP câteva puncte sensibile. În primul rând, s-ar putea spune că folosind prea mulți parametri, vor fi foarte greu de analizat algoritmii complicați. Totuși, în situații particulare, anumiți parametri ai modelului pot fi ignorați, simplificând analiza: în mulți algoritmi, un număr mare de mesaje sunt trimise grupat, unul după altul, astfel că timpul de comunicație va fi dominat de g, iar L poate fi ignorat. Algoritmii care presupun schimburi de date foarte rare pot fi analizați ignorând g și o.

4. Modelul BSP

Modelul **BSP** [9] (Bulk Synchronous Parallel) propus de Valiant se dorește a fi o generalizare a modelului PRAM. Un calculator BSP este compus din:

- un număr de perechi procesor-memorie;
- rețea de interconectare, care face posibilă comunicarea punct-la-punct;
- un mecanism de sincronizare o tuturor proceselor sau a unui subset de procese.

În categoria calculatoarelor BSP, se pot include cele mai variate tipuri de sisteme de calcul: un sistem uniprocesor cu memorie cache și memorie principală, un cluster de stații de lucru, pe care rulează PVM [3] sau MPI [4], calculatoare paralele, cu memorie distribuită, cum ar fi IBM SP2 sau Meiko CS2, chiar și calculatoare cu memorie partajată (de exemplu, Silicon Graphics Origin sau Power Challenge etc).

Pentru a caracteriza performanțele unui astfel de sistem, se utilizează o serie de parametri. Doi dintre acești parametri sunt numărul de procesoare și viteza unui procesor. Dacă definim un *pas* ca fiind o unitate de bază în calcul (de obicei, această unitate se alege ca fiind o operație în virgulă mobilă), viteza procesorului o putem exprima ca v pași/sec.

Un alt aspect important, care trebuie luat în considerare, este legat de performanțele rețelei de interconectare. Pentru a simplifica lucrurile, vom măsura performanța rețelei în unități de viteză ale procesorului. Alegând astfel unitățile de măsură, putem compara un sistem pentru care o operație de sincronizare între procesoare necesită doar un număr mic de pași în care s-ar fi putut executa calcule, cu unul în care această operație consumă mult timp.

De asemenea, la evaluarea performanțelor rețelei trebuie să ținem seama și de viteza cu care aceasta transmite un cuvânt - vom măsura această viteză tot în unitățile de măsură ale vitezei de calcul a procesorului. În acest fel, vom obține rata dintre puterea de calcul și cea de comunicare a sistemului.

Am obținut, astfel, patru parametri cu ajutorul cărora se caracterizează un calculator BSP:

- p - numărul de procesoare;
- v - viteza procesoarelor;
- l - timpul exprimat în pași elementari de calcul al unei operații de sincronizare la barieră;
- g - timpul exprimat în pași elementari de calcul pe cuvânt, necesar livrării mesajelor;

Deoarece viteza procesoarelor v este considerată drept unitate de măsură pentru ceilalți parametri, putem considera că avem doar trei parametri independenți: p, l și g.

Valorile acestora se obțin prin măsurători directe, efectuate asupra sistemelor de calcul. Viteza procesorului, care este factorul de normalizare a tuturor celorlalți parametri, este cea determinată prin măsurare directă, și nu viteza maximă pe care o indică producătorul. De asemenea, g este costul mediu al transferării unui cuvânt măsurat în condițiile unor aplicații reale, și nu este bazat doar pe valoarea lărgimii de bisecție, pe care o indică producătorul sistemului de calcul. Valoarea lui g poate fi estimată ca fiind raportul dintre numărul total de operații locale, efectuate de toate procesoarele pe secundă, și numărul total de cuvinte transferate într-o secundă de sistemul de comunicație.

Cu cât valoarea lui g este mai apropiată de valoarea parametrului l și acesta este mai mic, cu atât sistemul va fi mai scalabil. Mărind numărul de procesoare din sistem, fără a crește și dimensiunea problemei, volumul de calcule ce revine unui procesor va descreşte în mod corespunzător. Această descreştere va atinge, la un moment dat, limita în care costul comunicației, exprimat prin l și g, vor domina timpul total. Deci, cu cât l și g au valori mai mici, cu atât această limită va fi atinsă mai greu.

Din punct de vedere al programării, modelul BSP impune o anumită structurare a programelor. Punctul cheie al modelului BSP este noțiunea de *superpas* în care comunicarea și sincronizarea sunt complet decuplate. Un program BSP poate fi privit ca fiind alcătuit din faze între care au loc operații de comunicare globală. Acest model de organizare reprezintă un cadru general de dezvoltare a programelor portabile.

Anterior am definit un *pas* ca fiind o operație efectuată local asupra unor date. Programul BSP va fi constituit dintr-o secvență de superpași executați de fiecare procesor în paralel. Fiecare superpas este alcătuit dintr-un număr de pași efectuați cu date locale, urmați de o sincronizare la barieră a tuturor procesoarelor, moment în care devin efective toate accesele la date nelocale. Cererile de acces la date care nu sunt locale unui procesor pot fi efectuate oricând, în interiorul unui superpas, dar nu există garanția că acestea vor fi avut loc efectiv până în momentul operației de sincronizare, de la finalul superpasului.

Vom descrie o metodă simplă de estimare a complexității unui algoritm în modelul BSP. Conform acestei metode, un algoritm este alcătuit dintr-o secvență de superpași iar în interiorul unui superpas fiecare procesor efectuează calcule sau participă la o relație h. Acești superpași sunt urmați de o etapă de sincronizare globală. Relația h este o procedură de comunicare, în care fiecare procesor trimite cel mult h mesaje și primește cel mult h mesaje de un cuvânt fiecare. Prin ipoteza că în interiorul unui superpas procesorul fie efectuează calcule, fie comunică, se pierde din generalitatea modelului, dar se simplifică foarte mult analiza algoritmului.

Cheia de bază în estimarea complexității unui algoritm este funcția de cost, adică timpul necesar executării unui program paralel. Costul unei relații h, incluzând și sincronizarea de la sfârșitul superpasului, este:

$$T_{com} = \max_{0 \le i < p}(h_i) \cdot g + l = h \cdot g + l \qquad (2)$$

unde h este maximul dintre numărul de mesaje trimise sau primite de procesorul i. Convenim să exprimăm acest timp în pași elementari de calcul - timpul necesar unei operații în virgulă mobilă. Se observă că T_{com} crește liniar cu h, iar pentru valori mari ale lui h avem $T_{com} \approx h \cdot g$. Termenul l apare aici ca un cost fix, asociat operației de comunicare.

Timpul necesar calculelor propriu-zise dintr-un superpas poate fi exprimat în felul următor:

$$T_{calc} = \max_{0 \le i < p}(w_i) + l = w + l \qquad (3)$$

unde w_i este numărul de operații în virgulă mobilă, efectuate într-un superpas de procesorul i. Unitatea de măsură am considerat-o a fi egală cu timpul necesar unei operații în virgulă mobilă. Însumând cei doi timpi, se obține:

$$T_p = a + b \cdot g + c \cdot l \qquad (4)$$

unde a, b și c sunt constante pentru o dimensiune dată a problemei și pentru un număr dat de procesoare.

Bibliografie

1. **CULLER, D., R. KARP, D. PATTERSON, A. SAHAY, K. ERIK SCHAUSER, E. SANTOS, R. SUBRAMONIAN, T.VON EICKEN**: LogP: Towards a Realistic Model of Parallel Computation. Proc. of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, May 1993.

2. **DODESCU, GH., B. OANCEA, M. RĂCEANU**: Procesare Paralelă, Editura Economică, Bucureşti, 2003.

3. **GEIST, AL., A. BEGUELIN, J. DONGARRA, W. JIANG, R. MANCHECK, V. SUNDERAM**: PVM: Parallel Virtual Machine - A Users's Guide and Tutorial for Networked Parallel Computing, The MIT Press, Cambridge, Massachusets, 1994.

4. **GROPP, W., E. LUSK, A. SKJELLUM**: Using MPI: Portable Parallel Programming with the Message-Passing Interface, The MIT Press, Cambridge, Massachusets, 1994.

5. **KUMAR, V., A. GRAMA, A. GUPTA, G. KARYPIS**: Introduction to Parallel Computing, The Benjamin/Cummings Publishing Company, 1994.

6. **LAKSHMIVARAHAN, S., S. K. DHALL**: Analysis and Design of Parallel algorithms, McGraw-Hill, 1990.

7. **OANCEA, B.**: Metode numerice de calcul paralel pentru modelele matematice din economie, Teză de Doctorat, ASE, Bucureşti, decembrie, 2002.

8. **SABOT, G.W.**: High Performance Computing, Addison-Wesley, 1995.

9. **VALIANT, L.G.**: A Bridging Model for Parallel Computation. În: Communication of the ACM, 33/8, 1990.