Parallelism Implementation Mechanisms

Asist. Felician ALECU
Catedra de Informatică Economică, A.S.E. Bucureşti

The parallelism at application level also needs some software mechanisms (like degree, type and level of parallelism) together with the hardware ones used to make possible the usage of multiple resources and to implement the communication and synchronization mechanisms required by the parallel programs.
Keywords: *parallelization, parallel applications, parallel processing, granularity, communication, synchronization.*

Dezvoltarea sistemelor paralele a fost impusă de necesitatea existenței unor calculatoare performante care să fie capabile să rezolve aplicații cu cerințe intensive de calcul în condițiile în care aceste performanțe nu puteau fi atinse de către calculatoarele secvențiale clasice [Jor02]. Astfel, interconectarea mai multor procesoare în cadrul unei configurații paralele reprezintă o soluție eficientă din punct de vedere al costurilor ce permite satisfacerea cerințelor mereu în creştere ale puterii de calcul.

Calculatoarele de mare performanță sunt din ce în ce mai folosite în domenii cum ar fi analiza structurală, meteorologie, explorare petrolieră, cercetări privind fuziunea nucleară, medicină, simulări de aerodinamică, inteligență artificială, sisteme expert, detectarea imaginilor din satelit, siguranța reactoarelor nucleare, automatizări industriale, armată, inginerie genetică, socio-economie, etc. Fără ajutorul supercalculatoarelor, progresele în multe dintre aceste domenii nu s-ar fi putut realiza într-o perioadă de timp rezonabilă. Atingerea performanțelor deosebite a fost posibilă nu numai datorită progreselor înregistrate în domeniul hardware-ului, dar şi datorită elementelor noi şi inovatoare care au apărut în legătură cu arhitecturile paralele şi cu tehnicile de procesare utilizate.

Calculatoare paralele constând din sute sau mii de procesoare sunt acum disponibile comercial. Aceste calculatoare oferă o putere de calcul cu câteva ordine de mărime mai mare decât a supercalculatoarelor tradiționale, la un preț mult mai scăzut. În plus, sisteme distribuite constând din rețele de stații permit programarea paralelă a aplicațiilor cu cerințe mari de calcul.

Direcții de dezvoltare a programelor paralele

Dezvoltarea calculatoarelor paralele a atras după sine progrese importante şi în domeniul algoritmilor paraleli ce au ca principală caracteristică faptul că sunt capabili să execute simultan mai multe operații de calcul. Inițial, calculatoarele paralele nu puteau fi întâlnite decât în laboratoarele de cercetare. Pe aceste sisteme se executau exclusiv aplicații ştiințifice care necesitau calcule intensive. Exemple in acest sens ar fi programele paralele destinate simulării numerice a sistemelor complexe.

Astăzi însă, dezvoltarea calculatoarelor paralele este impusă cu preponderență de către aplicațiile comerciale care sunt capabile să manevreze şi să prelucreze largi colecții de date. Câteva dintre domeniile în care aceste aplicații şi-au găsit o largă aplicabilitate sunt: grafica şi realitatea virtuală, baze de date paralele, sisteme expert de diagnosticare şi suport decizional. Se poate afirma fără urmă de îndoială că direcțiile de dezvoltare a acestor două categorii de aplicații se apropie de un numitor comun din moment ce aplicațiile comerciale tind să efectueze calcule din ce in ce mai complexe iar cele ştiințifice să utilizeze colecții tot mai mari de date [Wyr04].

Una dintre cele mai întâlnite tehnici de paralelizare o reprezintă divizarea unei probleme în mai multe subprobleme (secvențe de in-

strucțiuni) şi rezolvarea acestora în paralel cu ajutorul proceselor şi a firelor de execuție. Tehnica *divide and conquer* s-a dovedit a fi una dintre cele mai performante tehnici de paralelizare a algoritmilor. Procesele şi firele de execuție trebuie să dispună de mecanisme de comunicare şi sincronizare. Se obține astfel modelul *von Neumann extins* care a făcut posibilă apariția arhitecturilor paralele.

Din perspectiva aplicațiilor, principalele utilizări ale calculatoarelor au evoluat după cum urmează:

- procesarea datelor;
- procesarea informației;
- procesarea cunoştințelor;
- procesarea inteligenței.

Procesarea datelor este şi astăzi sarcina majoră a calculatoarelor. Odată cu dezvoltarea structurilor de date, s-a trecut de la utilizarea calculatoarelor pentru procesarea datelor la utilizarea acestora pentru procesarea informației. Multe din calculatoarele zilelor noastre sunt limitate la aceste două nivele de prelucrare. La aceste nivele s-a putut atinge un înalt grad de paralelism. Cum în ultimii ani s-au dezvoltat bazele de cunoştințe, aceasta a avut ca efect utilizarea calculatoarelor pentru prelucrarea cunoştințelor. Există foarte multe sisteme expert pentru rezolvarea unor probleme din domenii specifice în care ele pot atinge un nivel de performanță comparabil cu acela al experților umani.

Cu toate acestea, calculatoarele sunt departe de a fi considerate inteligente. Inteligența este foarte greu de creat, iar procesarea acesteia este şi mai dificilă. Calculatoarele sunt încă incapabile să comunice cu oamenii în limbaj natural (vorbit, scris, imagini, documente, ilustrații), să fie satisfăcătoare în demonstrarea teoremelor, în inferența logică şi în gândirea creativă.

Paralelismul la nivelul aplicațiilor paralele implică necesitatea existenței unui suport software alături de cel hardware [Lad04]. Acesta din urmă face posibilă existența resurselor de calcul multiple şi implementează mecanismele de comunicare şi sincronizare între acestea.

Gradul de paralelism

Gradul de paralelism al unui program paralel reprezintă numărul de procesoare care sunt utilizate pentru execuția acestuia. Gradul de paralelism poate varia în timpul execuției programului. Fiecare procesor va prelucra doar o submulțime a datelor de intrare. Dimensiunea acestor unități de calcul atribuite procesoarelor reprezintă granularitatea de calcul.

Granularitatea cea mai fină este reprezentată de cazul în care numărul procesoarelor din sistem este egal cu dimensiunea datelor de intrare. Un astfel de caz nu reprezintă însă o soluție foarte eficientă din punct de vedere al costurilor. Din acest motiv, în practică, fiecare procesor va primi o submulțime a datelor de intrare spre a fi procesate. Datorită acestui lucru, granularitatea de calcul a programului va creşte (granularitate medie sau granularitate grosieră) în condițiile în care numărul de procesoare necesare pentru execuția algoritmului se va diminua.

Granularitatea de calcul are un impact foarte ridicat asupra performanțelor programelor paralele. În general se consideră că paralelismul de granulă fină conduce la rezultate mai bune însă în implementarea acestui tip de granularitate trebuie să se țină seama şi de caracteristicile algoritmului. Astfel, dacă acesta nu permite obținerea unei granule fine, atunci performanțele vor fi afectate de către timpii suplimentari cauzați de comunicația şi sincronizarea dintre procesoare. O granularitate mai mare conduce la reducerea timpului consumat pentru comunicația şi sincronizarea dintre procese dar atrage după sine alte probleme cum ar fi încărcarea neechilibrată a procesoarelor datorită faptului că sarcinile de calcul inegale vor genera timpi de inactivitate la nivelul unora dintre elementele de procesare.

Tipul de paralelism

La nivelul unei aplicații paralele se pot identifica două tipuri de paralelism:

1. *paralelismul structural* – implică efectuarea unor operații identice asupra unor seturi de date distincte. Datele de intrare sunt parti-

ționate iar fiecare submulțime este asociată unui procesor distinct care va executa asupra datelor același set de operații. Din acest motiv paralelismul structural mai poartă și numele de *paralelism de date*.

2. *paralelismul funcțional* – implică divizarea sarcinii totale de calcul în taskuri independente care sunt executate în paralel pe procesoarele disponibile în sistem. Paralelismul funcțional se mai numește și *paralelism de control* datorită faptului că fiecare procesor execută operații diferite asupra unor seturi distincte de date și comunică cu celelalte elemente de procesare pentru obținerea rezultatului final al programului. Cea mai simplă modalitate de realizare a paralelismului de control o reprezintă implementarea pipeline a algoritmilor. Astfel, algoritmul este divizat într-un număr de etape de execuție iar fiecare etapă va fi asociată unui anumit procesor din sistem. Datele de ieșire ale unui procesor vor reprezenta date de intrare pentru procesorul care va executa următoarea etapă. Uzual, paralelismul de control este implementat prin existența unui planificator care centralizează taskurile din sistem și le alocă, pe baza unui algoritm propriu, procesoarelor disponibile în funcție de momentul de execuție.

Trebuie evidențiat faptul că cele două tipuri de paralelism nu sunt complet independente. Astfel, în paralelismul structural, partiționarea datelor atrage după sine și o divizare a programului în sarcini de calcul independente care sunt alocate pe procesoarele din sistem. Mai mult, obținerea rezultatului final al problemei implică existența unui proces coordonator care să centralizeze rezultatele parțiale de la nivelul procesoarelor individuale și pe baza acestora să calculeze soluția finală a problemei.

Nivelul de paralelism

Paralelismul poate apărea pe diferite nivele de procesare [Dod02]. În ordinea descrescătoare a importanței, acestea ar fi:

- paralelism la nivel de program;
- paralelism la nivel de proces;
- paralelism la nivel de fir de execuție;
- paralelism la nivel de instrucțiune.

Cel mai adesea, nivelul superior este implementat algoritmic, în timp ce nivelul cel mai scăzut este implementat hardware. Rolul hardware-ului crește de la nivelele superioare către cele inferioare, în timp ce implementările software au o evoluție inversă. Din moment ce costul hardware-ului scade continuu iar cel al software-ului crește, din ce în ce mai multe metode hardware înlocuiesc abordările software convenționale.

Comunicația și sincronizarea între procese

În cazul calculatoarelor cu memorie distribuită, fiecare procesor are asociată o memorie locală pe care o poate accesa în mod direct. Pentru a avea acces la memoria locală a unui alt procesor se folosește mecanismul numit schimb de mesaje (*message-passing*). Procesoarele pot interacționa între ele doar prin schimb de mesaje. Nu există locații comune de memorie care să fie folosite la schimbul de date dintre două procesoare.

Transferul de mesaje presupune însă schimburi de date care pot aglomera rețeaua de interconectare. Transferul de date între două procesoare este o operație mare consumatoare de timp. Din acest motiv, performanțele rețelei de interconectare pot influența în mod hotărâtor eficiența cu care este utilizat sistemul paralel. În plus, este foarte greu ca sarcina de calcul distribuită procesoarelor să fie echilibrată.

Pe de altă parte, calculatoarele cu memorie partajată dispun de un spațiu comun de memorie care poate fi accesat de către toate procesoarele din sistem. Comunicația între procesoare se realizează prin date memorate în spațiul unic de memorie partajată, toate procesoarele având acces la întreaga memorie.

Utilizarea unui spațiu unic de memorie poate conduce la conflicte de acces la memorie atunci când mai multe procesoare încearcă să utilizeze aceeași zonă de memorie sau când doresc să utilizeze o variabilă partajată la care un alt procesor are acces exclusiv.

Una dintre principalele cauze generatoare de erori în execuția unui program paralel o reprezintă inconsistența temporară dintre o variabilă locală și copia ei din memoria globală.

Inconsistența temporară este un fenomen larg întâlnit în execuția programelor secvențiale însă efectele negative ale acestuia sunt nule datorită faptului că variabilele nu sunt partajate de mai multe procese. Dacă variabila este accesată concurent de către mai multe procese, atunci rezultatele obținute vor fi eronate datorită faptului că valoarea variabilei globale nu a fost actualizată în concordanță cu modificările operate asupra copiei locale. Această situație se manifestă atât în cadrul calculatoarelor cu procesoare multiple cât şi în cazul sistemelor secvențiale multitasking în care se execută concurent mai multe procese.

Pentru a preveni astfel de situații este necesar ca actualizarea valorii unei variabile partajate să se realizeze ca o operație atomică care nu poate fi întreruptă de către un alt proces care doreşte să acceseze aceeaşi zonă de memorie. Cel de-al doilea proces va putea actualiza variabila globală doar după ce primul proces a eliberat zona respectivă de memorie. Serializarea accesului la variabilele partajate se obține prin implementarea mecanismelor de excludere mutuală.

Prin eliminarea inconsistenței temporare a unei variabile globale față de copia sa locală se obțin valori consistente care conduc la execuția corectă a proceselor concurente. Pentru actualizarea în siguranță a unei variabile globale este necesar ca operația respectivă să se execute într-o secțiune critică. Procesul care intră în secțiunea critică are dreptul exclusiv de a accesa variabila partajată asociată secțiunii respective. Atât timp cât procesul execută secțiunea critică, toate celelalte procese concurente nu vor avea dreptul să acceseze variabila partajată respectivă. Acest mecanism poartă numele de excludere mutuală deoarece un proces exclude temporar accesul celorlalte procese la variabila partajată.

În marea majoritate a sistemelor de calcul, implementarea acestui mecanism este realizată în trei faze:

- mai întâi se alege unul dintre procesele concurente care doresc să acceseze variabila partajată.
- procesul execută secțiunea critică în timp ce toate celelalte procese concurente vor aştepta completarea acestei operații.
- după încheierea secțiunii critice resursa este eliberată iar un alt proces poate începe execuția secțiunii critice.

Excluderea mutuală nu poate fi implementată prin utilizarea unei variabile globale care să permită sau să interzică accesul unui proces la secțiunea critică. Pentru rezolvarea corectă a excluderii mutuale se folosesc mecanisme hardware speciale cum ar fi invalidarea întreruperilor şi instrucțiunile de tip *TestAndSet*. Aceste mecanisme sunt utilizate pentru obținerea unor obiecte de sincronizare de nivel ridicat cum ar fi semafoarele, mutex-urile, barierele, variabilele de condiție.

Bibliografie

[Wyr04] R. Wyrzykowski, *Parallel Processing And Applied Mathematics*, Springer, 2004

[Lad04] S. Ladd, *Guide to Parallel Programming*, Springer-Verlag, 2004

[Dod02] Gh. Dodescu, B. Oancea, M. Raceanu, *Procesare paralelă*, Editura Economica, Bucureşti, 2002

[Jor02] H. F. Jordan, H. E. Jordan, *Fundamentals of Parallel Computing*, Prentice Hall, 2002