Planificarea programelor paralele

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

In this paper we will focus on task allocation methods used to improve performances of parallel programs executed on parallel systems. The parallel program has to be divided into several independent tasks that will be executed concurrently. These tasks have to be assigned to the system processing units using a static or dynamic fault tolerant task-scheduling algorithm. Using the static approach, the schedule is computed at compile time. Dynamic scheduling means computing at runtime either by a central master or in a distributed manner.
Keywords: *Parallel program, task allocation, task scheduling problem, static task scheduling, dynamic task scheduling, machine configuration graph, program graph, scheduling algorithms and fault tolerance.*

Introducere

În ultimii ani sistemele paralele si distribuite au devenit din ce în ce mai atractive pentru aplicatii cu cerinte de calcul intensive cum ar fi cele destinate simularii unor sisteme complexe (aerodinamica, meteorologie). Principalul avantaj al unor astfel de sisteme este raportul mai mult decât atractiv dintre pret si performantele care pot fi obtinute.

Prin planificarea unui program se întelege alocarea taskurilor componente pe procesoarele existente în sistem în functie de relatiile de dependenta dintre acestea.

Planificarea poate fi realizata static sau dinamic. În planificarea statica, se cunoaste, înca din faza de compilare, procesorul pe care va fi rulat fiecare task component al programului. Datorita faptului ca problema planificarii este o problema NP-completa, solutii exacte pot fi obtinute doar pentru probleme de dimensiuni reduse (vezi [Cof72], în celelalte cazuri fiind necesara aplicarea unor algoritmi euristici. În toate cazurile însa este necesara cunoasterea în avans a timpilor de executie a taskurilor. Pentru marea majoritate a aplicatiilor practice, estimarea, la momentul compilarii programului, a timpilor de executie este aproape imposibila.

În planificarea dinamica alocarea taskurilor pe procesoarele sistemului se face la momentul executiei. Sarcina planificarii poate reveni unui singur procesor (numit master) sau poate fi distribuita la nivelul unitatilor de procesare prezente în sistem.

Planificarea este strâns legata de modalitatile de divizare a programelor paralele în unitati de executie. Pentru ca aceste unitati de executie sa poata fi executate concurent este necesar ca ele sa fie independente. Independenta a doua taskuri înseamna obtinerea aceluiasi rezultat indiferent daca cele doua taskuri sunt executate secvential în orice ordine sau în paralel. Pe lânga restrictiile legate de ordinea de executie, conflictele de acces la resursele partajate limiteaza independenta între doua sau mai multe taskuri. Pentru a analiza dependentele dintre taskurile unui program, uzual se foloseste un graf directionat numit graf de dependenta al programului. Nodurile grafului sunt unitatile de executie ale programului iar arcele reprezinta relatiile de dependenta între acestea. Din analiza grafului de dependenta reies în mod clar posibilitatile de paralelizare a programului respectiv.

Cele mai multe limbaje paralele de programare pun la dispozitia utilizatorilor instrumente ce permit definirea taskurilor componente (procese, threaduri) si a mecanismelor de comunicatie (transfer de mesaje, variabile partajate) si sincronizare (semafoare, lockuri, bariere, monitoare, evenimente) dintre acestea.

Exista o serie de limbaje de programare, numite limbaje de programare cu paralelizare implicita, care, plecând de la descrierea secventiala a algoritmului, realizeaza în mod automat paralelizarea acestuia. Rolul principal îl are compilatorul care analizeaza codul sur-

sa al programului si cauta sa identifice segmente de cod care sa fie executate în paralel. Aceste segmente de cod iau forma proceselor si threadurilor. Tot compilatorul este cel care specifica modul de comunicatie si sincronizare dintre elementele de executie identificate. Cu toate acestea, misiunea identificarii segmentelor independente din cadrul unui program nu este una facila si este limitata de complexitatea programului. Cu cât acesta are o structura mai neregulata, cu multiple ramificatii, bucle, apeluri, cu atât sansele de paralelizare automata scad.

Formularea problemei de planificare

Sistemul paralel poate fi descris cu ajutorul uni graf neorientat numit graful de configuratie al sistemului. Nodurile grafului reprezinta procesoarele sistemului împreuna cu memoria asociata. În dreptul unui nod se poate trece viteza procesorului si dimensiunea memoriei asociata acestuia. Arcele semnifica legaturile de comunicatie (conexiunile). În dreptul unui nod se poate trece capacitatea legaturii dintre cele doua noduri.

În marea majoritate a cazurilor se considera ca un procesor poate comunica cu oricare altul din sistem. Din acest motiv graful de configuratie al sistemului este un graf complet, între oricare doua noduri existând o muchie care le uneste.

Comunicatia între procesoare se realizeaza asincron si este gestionata de catre un subsistem de comunicatie care are si sarcina de a trata eventualele erori de comunicatie care pot aparea. Chiar daca legatura directa dintre doua noduri a picat, acestea se considera totusi conectate daca exista cel putin un drum în graf care sa le uneasca.

Erorile ce pot aparea la nivelul elementelor de procesare sunt gestionate de catre algoritmul de planificare. Atunci când un procesor devine izolat (nu mai exista conexiune cu nici un alt nod), se transmite un mesaj de informare tuturor nodurilor conectate din sistem. Erorile la nivelul elementelor de procesare pot fi tranziente sau permanente. În cazul unei erori tranziente, elementul de procesare poate participa din nou la executia programului dupa ce eroarea înregistrata a fost remediata.

Programul paralel ce se doreste a fi executat poate fi si el descris cu ajutorul unui graf orientat numit graful programului. Nodurile grafului pot fi de doua feluri: taskuri (reprezentate sub forma de cercuri) si date (reprezentate sub forma de patrate). Un nod în care se gaseste un task poate fi etichetat cu durata de executie a acestuia (costul de calcul). Datele de iesire ale unui task ce sunt utilizate ca date de intrare de catre un alt task apar ca nod al grafului. Nodurile semnifica fluxurile de date din sistem generate de executia programului si introduc constrângeri de precedenta în executia taskurilor. Astfel, un task nu poate fi executat pâna când toate datele de care are nevoie sunt disponibile. Datele necesare taskului devin disponibile atunci când nodurile parinte au încheiat executia si pot comunica datele rezultate. Exemplu de graf al programului este prezentat în figura 1.

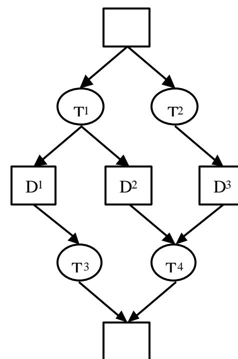

Fig. 1. Exemplu de graf al programului

Metode de planificare

În cazul în care graful programului este complet cunoscut înainte de momentul executiei programului (taskuri componente, timpi de executie, flux de date), se poate utiliza o planificare statica a taskurilor. În schimb, daca forma grafului programului nu poate fi complet determinata decât la momentul executiei, atunci se va folosi planificarea dinamica.

Datorita faptului ca se realizeaza la momentul compilarii programului (deci înainte de executia efectiva a acestuia), planificarea statica nu este consumatoare de timpi de executie. Planificarea dinamica implica executia

algoritmului de planificare concomitent cu rularea programului paralel.
Scopul principal al planificarii este reducerea timpului de executie paralela a programului. Din acest motiv, timpul consumat pentru fundamentarea deciziei de planificare trebuie sa fie mai mic decât timpul câstigat prin aplicarea respectivei planificari. Nu se poate afirma ca un anumit tip de planificare este cel mai bun deoarece alegerea unei metode de planificare pentru un anumit program se face în functie de caracteristicile acestuia. Planificarea statica este atractiva datorita faptului ca se executa o singura data si nu este consumatoare de timpi de executie. Se poate afirma însa cu certitudine faptul ca planificarea dinamica ofera posibilitatea tratarii într-o maniera eficienta a eventualelor erori permanente sau temporare aparute la nivelul unuia sau mai multor elemente de procesare din sistem.
Planificarea se poate reprezenta grafic cu ajutorul diagramelor Gantt. Dintr-o astfel de diagrama reiese în mod clar procesorul pe care este executat un anumit task precum si timpul de executie al acestuia.
Clusterizarea este o metoda frecvent folosita în cazul planificarii statice. Ideea ce sta la baza acestei metode este aceea conform careia costul de comunicatie a doua taskuri este diferit de zero doar daca taskurile sunt planificate pe procesoare diferite. Prin aceasta metoda se încearca reducerea multimii taskurilor prin gruparea acestora. Grupele astfel obtinute poarta numele de clustere si vor fi planificate pe acelasi procesor. Astfel, daca pentru doua taskuri avem o valoare mare a costului de comunicatie, atunci acestea vor fi incluse în cadrul aceleiasi grupe fiind planificate pe acelasi procesor. Planificarea pe acelasi procesor a celor doua taskuri anuleaza costul de comunicatie. Dupa definirea clusterelor, acestea vor fi planificate pe procesoarele sistemului în mod static, la momentul compilarii programului.
În cazul planificarii dinamice, sarcina planificarii poate reveni unui singur procesor sau poate fi distribuita elementelor de procesare din sistem.
Prin folosirea unui singur procesor (numit procesor *master*), acesta va fi responsabil pentru alocarea taskurilor pe procesoarele sistemului datorita faptului ca pe respectivul procesor va fi rulat algoritmul de planificare.
Folosirea unei metode descentralizate de planificare presupune delegarea sarcinii planificarii catre nodurile care au terminat executia taskurilor alocate[Liy95] [Yue97]. Sa consideram cazul în care taskul T_1 (figura 1) si-a încheiat executia. Procesorul care a executat taskul respectiv devine liber si verifica daca sunt taskuri a caror executie depinde de rezultatele generate de T_1. Astfel de taskuri vor fi numite succesori. Un task succesor este pregatit pentru executie atunci când îi sunt disponibile toate datele de intrare. În cazul grafului din figura 1, taskurile T_3 si T_4 pot deveni succesori gata de executie. Dupa identificarea succesorilor gata de executie, procesorul încearca sa gaseasca procesoare libere în sistem carora sa le fie alocate taskurile respective (unul dintre succesori poate fi executat chiar pe procesorul respectiv). Dupa executarea succesorilor, procesoarele carora le-au fost alocate taskurile respective preiau sarcina planificarii noilor succesori. Acest procedeu se continua pâna în momentul în care toate taskurile componente ale programului au fost executate si s-a obtinut setul final de date. În cazul în care un task are mai multi predecesori (cazul taskului T_4 din figura 1), trebuie sa existe implementata o politica cu ajutorul careia sa se poata stabili în mod clar care dintre predecesori este responsabil pentru alocarea respectivului task.
Tratarea erorilor într-o maniera eficienta poate fi realizata cu succes în planificarea dinamica deoarece algoritmul de planificare, fiind rulat la momentul executiei programului paralel, se poate adapta la evenimentele care pot aparea în sistem.
În cazul în care un element de procesare devine indisponibil, algoritmul de planificare va trebui sa se limiteze la folosirea exclusiva a procesoarelor valide [Alv98]. Daca planificarea se realizeaza la nivel centralizat, atunci procesorul master nu va mai aloca taskuri procesorului indisponibil, pâna în momentul în care acesta va fi raportat ca fiind din nou valid de catre rutinele software specializate în diagnosticarea elementelor hardware din sistem. În cazul în care

sistem. În cazul în care planificarea se face la nivel local, atunci este suficient ca toate procesoarele valide din sistem sa fie informate, prin intermediul unui mesaj distribuit în cadrul retelei de comunicatie, asupra faptului ca un procesor a devenit indisponibil. În acest fel el nu va mai fi folosit în cadrul planificarii pâna în momentul în care va deveni din nou disponibil, moment la care procesoarele sistemului vor fi informate de acest lucru tot prin intermediul unui mesaj.

O situatie aparte este aceea în care taskul este lansat din nou în executie pe procesorul în memoria caruia este stocata copia datelor de intrare. Într-un astfel de caz se realizeaza o noua copie a datelor respective iar aceasta copie va fi stocata în memoria locala a unui alt element de procesare. Aceasta noua copie va putea relansa taskul în cazul în care executia acestuia ar esua din nou. Dupa ce taskul a fost executat cu succes, datele lui de intrare nu mai sunt necesare. Din acest motiv datele originale cât si eventualele copii ale acestora sunt eliminate din memoriile locale ale elementelor de procesare din sistem. Datele de iesire reprezinta date de intrare pentru succesorii taskului si din acest motiv acestea vor fi la rândul lor duplicate.

Problema planificarii, în forma ei cea mai generala, pleaca de la existenta unui graf al programului cu n noduri si a unui sistem de calcul paralel cu p procesoare. Solutia optima se afla într-un spatiu de p^n planificari posibile.

Pentru a masura calitatea si eficienta diferitelor algoritmi si metode de planificare ar trebui sa se compare performantele obtinute cu cele ale planificarii optimale. Planificarea optimala este aceea care asigura un timp minim de executie a programului paralel.

Atunci când solutia optimala nu poate fi atinsa, se încearca rezolvarea planificarii într-o maniera eficienta prin utilizarea unor algoritmi euristici de planificare care sa conduca la gasirea unor solutii sub-optimale în timpi de executie rezonabili.

Pentru marea majoritate a problemelor de planificare însa, solutia optima nu poate fi obtinuta. Din acest motiv, concluzii asupra eficientei uni algoritm de planificare se pot trage doar comparând performantele acestuia cu performantelor altor algoritmi de planificare cunoscuti în conditiile în care testele se efectueaza asupra acelorasi seturi de date [Kwo98].

Bibliografie

[Alv98] G.A. Alverson, W.G. Griswold, C. Lin, D. Notkin, D.P. Agrawal, *Optimal Scheduling Algorithm for Distributed-Memory Machines*, IEEE Transactions on Parallel and Distributed Systems, Vol. 9, Nr. 1, January 1998

[Cof72] E. Coffman, R. Graham, *Optimal Scheduling on Two Processor System*, Acta Informatica, Vol. 1, 1972

[Kwo98] Y.K. Kwok, I. Ahmad, *Benchmarking the Task Graph Scheduling Algorithm*, IEEE Transactions on Parallel and Distributed Systems, Vol. 9, Nr. 3, March 1998

[Liy95] Y. Li, F.J. Markus, J. Rost, *Agent Scheduling – A Model for Dynamic Task Scheduling*, EURO-PAR, Lectures Notes in Computer Science, Springer-Verlag, 1995

[Yue97] K.K. Yue, D.J. Lilja, *An Efficient Processor Allocation Strategy for Multiprogrammed Shared-Memory Multiprocessors*, IEEE Transactions on Parallel and Distributed Systems, Vol. 8, Nr. 12, December 1997