APP – teorie

1. Excludere mutuala bazat pe XCHG

Instrucţiunea XCHG(r,m) interschimbă, ca o operaţie atomică, conţinutul registrului r cu celula de memorie m (semafor). ❑ deci ri este o variabilă locală a sarcinii Si şi este iniţializată cu 0, iar ❑ m este o variabilă globală, comună tuturor sarcinilor care este iniţializată cu 1.

Ambele variabile pot lua valorile 0, 1. Algoritmul pentru Si va fi:

<inceput sarcina>;

Repetă

XCHG(ri ,m);

până_când ri;

<sectiune critica>;

XCHG(ri ,m);

<rest sarcina> ;

Numai procesul care găseşte m=1 va intra în secţiunea critică şi va trece în zero. La ieşire din secţiunea critică va returna pe m la valoarea 1 pentru a permite celorlalte sarcini să intre în secţiunea critică. (m=1 este o conventie, in unele sisteme se considera m=0) . Când o sarcina Si intră în secţiunea critică, ri=1.

2. algoritmul optim de partiție a unui microprogram descries prin graf de dependențe de date vs algoritmul euristic de partiție

In cadrul algoritmului optim de partitie a unui microprogram, descris prin graf de dependente de date, se va realiza partitia optima prin generarea din setul de microoperatii uODk a tuturor uIC posibile, pentru fiecare dintre ele se va analiza influenta pe care o are asupra generarii setului de microoperat ii disponibile urmator, uODk + 1. In cazul algoritmului euristic de partitie, el este bazat pe ordonare a microoperatiilor din setul disponibil uODk in functie de succesorii in graful de dependente de date. In cadrul

algoritmului euristic nu se va genera insa o partitie optima, dar va fi mult mai rapida in unele situatii. partitie euristica = bazata pe ordonarea operatiilor in functie de succesori -> complexitate liniara algoritm optim de partitie = NP completa

3. Descrieţi principiul de funcţionare a sistemelor de tip Data Flow . Comentaţi componentele principale ale arhitecturii.

   Sistemele de tip Data Flow au un model ce este bazat pe fluxul de date, in cadrul caruia executia unei instructiuni de prelucrare depinde de disponibilitatea operandului. In cadrul acestor sisteme nu avem Program Counter. Specifica efectuarea unor operatii imediat ce vom avea operanzii disponibili si din aceasta cauza nu avem nevoie de contorul de program. De asemenea, nu este nevoie de adrese pentru memorarea variabilelor iar operatia se efectueaza numai atunci cand token-ul este prezent pe ambele intrari ale operandului. Datele se stocheaza in cadrul instructiunilor, adica vom mentine un token cu valoare binara in cadrul instructiunii pentru fiecare data. O unittate specializata va verifica disponibilitatea datelor. Ca avantaje avem faptul ca avem un paralelism ridicat si o viteza de executie mai mare. De asemenea este usor de implementat partea de comunicatie si sincronizare, insa vom avea overhead mare pentru control, iar manipularea structurilor de date va fi dificila. Avem Token = {Data, Tag Destinatie, marker} si Match = {Tag, Destinatie}. Avem Df1, Df2, …, Dfn ce reprezinta retea de masini de baza ce sunt interconectate printr-o retea de comutare; Token Queue cu care pe masura ce vom avea token-uri, ele vor fi adaugate in coada Matching Unit - elementul de prelucrare are un token pe fiecare ramura; Instruction store - ia elementele de prelucrare si le pune pe o unitate de procesare

4. Exemple de calculatoare paralele

   (aici la ce se referă? Taxonomia Flynn cu extinderea pe tipuri de arhitecturi paralele?)

5. Teorema de suficienta + demonstratie
   Sistemele de sarcini formate din sarcini mutual neinterferente sunt determinate

Justificare (prin inducţie)

- **Pentru un sistem de sarcini format dintr-o singura sarcina este evident.**
- **Presupunem că afirmaţia este adevărată pentru sisteme cu n-1 sarcini, (α'1 şi α'2 sunt secvenţe de execuţie pentru un sistem cu n-1 sarcini)**
- **Presupunem sistemul C = (S, <' fiind obţinută din < eliminând relaţia de precedenţă ce implică pe S. V(Mi , α'1) = V(Mi , α'2) din ipoteza de inducţie.**
- **Deci se poate presupune că valorile din DS sunt aceleaşi, atât pentru α'1 cât şi pentru α'2 ,**
- **respectiv F(Mi , α'1) = F(Mi , α'2) pentru Mi DS .**

- **Rezultă astfel că pentru α'1 şi α'2 , sarcina S scrie aceeaşi valoare v pentru orice celulă Mi RS . Pentru Mi RS : V(Mi , α1) = V(Mi , α'1 SI SF) Lemă = V(Mi , α'1) Mi RS si ip. de inducţie = V(Mi , α'2) Mi RS = V(Mi , α'2 SI SF) Lemă = V(Mi , α2) Pentru Mi RS V(Mi , α1) = V(Mi , α2) Pentru Mi RS : V(Mi , α1) = V(Mi , α'1 SI SF) Lemă = (V(Mi , α'1),v) ip. de inducţie = (V(Mi , α'2),v) = V(Mi , α'2 SI SF) Lemă = V(Mi , α2) Pentru Mi RS V(Mi, α1) = V(Mi, α2)**
- **Sistemul C este determinat dacă sarcinile sunt mutual neinterferente.**

6. Limita inferioara

Limita inferioară = μ**, limita inferioară pentru situaţia în care resursele sunt distincte.**

μ **=** max **{|**μoPi**|} 1 ≤** i **≤** P

Fiind o singur resursa, se va fi controlata secvenţial, deci avem |μoPi**|, fiind |**μoPi**| microoperatii.**

Când resursele nu ar fi fost diferite, am putea considera că

μ **=** max **{[|**μoPi**| / |**Pi**|]}**

În calculul acestei limite, s-a presupus că nu există dependenţă de date între microoperatiile care controleaza diferite resurse.

Eu cred ca aici nu vrea asta, ci mai degraba sa ii explici la ce se refera limita asta inferioara, eventual pe scurt cazurile respective. Si sa ii mai explici si de ce vrei sa afli limita asta inferioara. Chiar nu cred ca vrea formulele si explicatii la formule.

7. Sun-Ni

Introduce modelul bazat pe limita de memoriei

- **Ts = fseq + g(p)(1- fseq)**
- **unde G(n) este cresterea de memorie a unui procesor**
- **Tp = fseq + g(p)(1- fseq) /p**
- **Vp=Ts/Tp= (fseq + g(p)(1- fseq)) /(fseq + g(p)(1- fseq) /p)**
- **Caz 1 : g(n)=1**
- **Vp= (fseq + 1- fseq) /(fseq + (1- fseq) /p)= 1 /(fseq + (1- fseq) /p) ~ Amdahl**
- **Caz 2 : g(n)=p**
- **Vp=Ts/Tp= (fseq + p(1- fseq)) /(fseq + p(1- fseq) /p)=(fseq + p(1- fseq)) /1 = fseq + p(1- fseq) ~ Gustafson**
- **Caz 3: g(p)= m > p**
- **Vp=Ts/Tp= (fseq + m(1- fseq)) /(fseq + m(1- fseq) /p)=**
- **(fseq + m(1- fseq))/(m/p+ (1-m/p)fseq)**

8. Sisteme de clasa determinate, secventa de executie partiala

9. Clase compatibile. Clase incompatibile

10. Modalitati de codificare microinstructiuni. Avantaje si dezavantaje pentru fiecare

11. Metode de calcul paralel: enumerare + comparatie intre memoria partajata si procesare sistolica

12. Amdhal + Worldon + Gustafson

13. Inidicatori paraleli.
    Viteza de prelucrare – Vp = T1 / Tp

Eficienta - Ep = Vp / p

Redundanta - Rp = Op / O1

Utilizarea - Up = Op / (p * Tp) <= 1

14. Sistem de sarcini determinat. Secvente de executie. Secventa partiala de executie.

    Secventa de executie: Alfa = a1a2a3…ai, ai+1, … a_2n. ai = eveniment ce declanșează sau termina o sarcina, cu respectarea ordinii de precedenta < (< relatie de ordine partiala).

    Secventa de executie partiala: orice prefix dintr-o secventa de executie valida.

    Sistemul de sarcini este determinat daca indiferent de ordinea de executie a sarcinilor independente (adica pentru orice secventa alfa valida) si indiferent de timpul necesar unei sarcini independente, se produce acelasi rezultat si, in plus, daca acest lucru se respecta pentru orice interpretare f_s a sistemului. (f_s = ceea ce executa fiecare sarcina).

** E corect si ce scrie mai jos, dar e dintr-un curs de mai tarziu. Deci le-as zice pe amandoua.

Un sistem de sarcini C este determinat dacă pentru orice stare iniţială s0 dată, V(Mi ,α) = V(Mi ,α'), 1 i m, pentru toate secvenţele de execuţie α ,….α' din C.

15. Tipuri de microinstructiuni

microinstrucţiuni operaţionale care controlează primitivele funcţionale ale unităţii de execuţie a sistemului numeric, asigurând fluxul de informaţie şi acţiunile asupra resurselor;

microinstrucţiune de ramificaţie (de salt) care inspectează starea primitivelor funcţionale şi asigură ramificaţia în algoritmul de control, constituind suportul pentru implementarea deciziilor.

16. Sisteme microprogramate vs. microprogramabile

17. Algoritmul de partitionare (in 8 pasi).

P1. Initializare

Se initializeaza partitia $\mu PT = \Phi$ **,** $\mu PTA = \mu B(\mu O)$ **- tot microsublocul (deci, daca le luam si le facem sevcential)**

μOD **=** $\{\mu Oj \mid \forall \mu Oj \in \mu B(\mu O)$ $nu\ exista\ \mu Oi < \mu Oj\}$ **- deci pentru care nu exista predecesori**

$\mu OND = \{\mu Oi \mid \mu Oi \in \mu B(\mu O) \setminus \mu OD\}$ **- operatii nedisponibile**

P2. Daca $\mu OND = \Phi\ si\ \mu OD \leq 3$ **atunci salt la pas 5**

P3. Se genereaza {IC} – multimea de microoinstructiuni complete

$\{\mu IC\} = \{\mu ICj , . . . , \mu ICk\}$ **- din OD curent daca j ≠ k (s-au generat mai multe IC) atunci salvare PTj = PT ODj = setul curent OD {IC}j = {IC} \ ICj**

P4. $\mu PT = \mu PT \cup \mu ICj\ \mu OD = \mu OD \setminus \mu ICj \cup \{\mu Od\}j\ \mu OND = \mu OND \setminus \{\mu Od\}j$

daca $\mu OD \neq 0\ si\ \mu PT \leq \mu PTj - 1$ **atunci salt pas 2**

P5. daca OD = atunci salt pas 7

P6. Se genereaza µIC din µOD curent. µPT = µPT U µIC; µOD = µOD \ µIC;

P7. daca μOD ≠ Φ si |μPT|< | μPTA| atunci μPTA = μPT altfel daca |μPTA| ≤ nmμI atunci μPTA este optima STOP.

P8. daca {μIC}j ≠ 0 (cel care a fost salvat la pasul 3) atunci reface μPT = μPTj ; μOD = μODj ; $j \leftarrow j$ + 1; (trec la urmatorul, selecteaza urmatoarea IC din setul generat) μIC = μICj μOND = {μOj |$\mu Oj \in \mu B(\mu O) \setminus (\mu PT \cup \mu OD)$} salt pas 4 altfel PTA este optima .

20. Fie un procesor care implementează o structura paralela pipeline de citire interpretare executie pentru procesare suprascalară , cu patru unitati paralele. Presupunând un ciclu de instrucțiuni contine:

-citire care necesita o singura perioada de ceas

-decodificarea instructiunii necesita două perioade de ceas

-executia instructiunii necesita doua perioade de ceas Avem o secvență de 3000000 de instrucțiuni. Considerand freventa ceasului in Ghz de 1. Calculati durata de executie a secventei de program in milisecunde

(1 + 2 + 3*10^6 / 4 * 2) * 10^(-9) ~= 1.5ms

Considerând că în urma analizei microoperațiilor dintr-un subloc au rezultat microintrucțiunile complete,mIC1,…mIC5 care conțin micro[1]operațiile ca în tabelul alăturat:

mIC1 = mo1 mo2 mo3 mo4 mo5 mo6

mIC2 = mo3 mo7 mo8 mo9

mIC3 = mo1 mo2 mo8 mo9 mo10

mIC4 = mo4 mo8 mo11

mIC5 = mo6 mo8

Care este organizarea optimă a câmpurilor din formatul general al microinstrucțiunilor:

a. Câmp 1: (mo1)

Câmp 2: (mo2)

Câmp 3: (mo3)

Câmp 4: (mo4 mo7 mo9 mo10 mo11)

Câmp 5: (mo5)

Câmp 6: (mo6)

Câmp 7: (mo8)

b. Câmp 1: (mo1)

Câmp 2: (mo2)

Câmp 3: (mo3)

Câmp 4: (mo4 mo9 mo10)

Câmp 5: (mo5 mo11)

Câmp 6: (mo6 mo7)

Câmp 7: (mo8)

c. Câmp 1: (mo1)

Câmp 2: (mo2)

Câmp 3: (mo3)

Câmp 4: (mo4 mo9)

Câmp 5: (mo5 mo11)

Câmp 6: (mo6 mo7 mo10)

Câmp 7: (mo8)

Ar fi D, dar dc? -- n-am idee, din ce văd și C este validă, nu văd ce ar fi greșit la ea **-> se calculeaza si costul**: pt fiecare camp, numeri cate instructiuni are, adaugi 1 (o instructiune nop imaginara, la fiecare camp), si obtii N -> te gandesti de cati biti ai nevoie pt a reprezenta N numere (ex: N = 4 => 2 biti, N = 5 => 3 biti); aduni toti bitii necesari pt toate campurile si obtii **costul**. Se alege varianta cu **costul minim** (exemple: campul 1 de mai jos are 1+1 instructiuni, e suficient 1 bit; campul 5 de mai jos are 3+1 instructiuni, sunt suficienti 2 biti - se va observa ca varianta **c** are costul **10**, iar **d** are costul **9**)

d. Câmp 1: (mo1)

Câmp 2: (mo2)

Câmp 3: (mo3)

Câmp 4: (mo4)

Câmp 5: (mo5 mo7 mo10)

Câmp 6: (mo6 mo9 mo11)

Câmp 7: (mo8)