7. prednáška (11.4.2011)

Distribuované algoritmy

Synchrónny distribuovaný výpočet

Krok synchrónneho výpočtu obsahuje:

- sekvenčný výpočet (na základe prijatých správ), príprava ďalších správ a ich pripojenie ku príslušným komunikačným kanálom
- odoslanie správ a prijatie nových správ (odstránenie všetkých správ z kanálov)

Časová zložitosť – počet týchto krokov, potrebných na dosiahnutie koncového stavu

Komunikačná zložitosť – počet odoslaných správ (počet bitov v správach)

Odosielanie a prijímanie správ

send (data, dest, tag) väčšinou asynchrónne

receive (data, src, tag, status)

src - špecifikuje identifikáciu procesu (rozhrania), z ktorého chceme prijať správu

tag – akú značku má mať prijatá správa

receive (data, src, tag) prijíma z ľubovoľného rozhrania s ľubovoľnou značkou

src, tag – identifikácia rozhrania, z ktorého prišla správa a prijatá značka

Leader election in a general network

Voľba koordinátora v sieti procesov

- každý proces vie identifikovať svojich susedov (zoznam S)
- výsledkom je dosiahnutie špeciálneho stavu *leader* pre jediný procesor – s najvyšším **myid**

orientácia ? obojsmerný prenos ? známy počet procesov ? priemer siete ?

Voľba koordinátora záplavou

procedure FloodMax

{vstup: S – susedné procesy, diam – priemer siete; výstup: status – leader or not leader}

status := **not** leader; maxid := **myid;**

for rounds := 1 **to** diam **do**

forall v **in** S **do** **send**(maxid, v, 0) **endforall;**

forall v **in** S **do** **receive**(id, v, 0); **if** id > maxid **then** maxid := id **endforall;**

endfor

if maxid = **myid then** status := leader

endif

end FloodMax

{veľká komunikačná zložitosť – diam*m ~ $O(n^3)$, nutnosť poznať maximálnu vzdialenosť procesov, alebo aspoň počet vrcholov}

Voľba koordinátora záplavou - zlepšenie

```
procedure FloodMax2
{vstup: S – susedné procesy, diam – priemer grafu; výstup: status – leader}
new := true; status := not leader; maxid := myid;
for rounds := 1 to diam do
    if new then new := false; forall v in S do send(maxid, v, 0) endforall
    endif;
    forall v in S do
        receive(id, v, 0); if id > maxid then maxid := id; new := true endif
    endforall
endfor;
if maxid = myid then status := leader endif;
end FloodMax2
{ďalšia vlna sa posiela len ak došlo k zmene}
```

Záplavové algoritmy (flood algorithms)

- úplne distribuované (neexistuje koordinátor)
- možno vyvolať v pravidelných intervaloch, resp. pri zmene topológie siete – on-line informácie – možnosť eliminácie chýb
- veľký počet správ

Algoritmy vlny (wave algorithms)

- princíp vlny – od koordinátora, resp. od listových uzlov
- výhodné vopred stanoviť priebeh vlny – orientovaná kostra s koreňom – koordinátorom
- možno očakávať odpoveď (echo) v opačnom smere ako smer pôvodne šírenej vlny

Prehľadávanie siete do šírky

```
procedure TreeBFS
{vstup: S – susedné procesy, p – rodičovský proces; výstup: }
if myid <> p then receive(join, p, 0) endif;
forall v in S - {p} do send(join, v, 0) endforall;
end TreeBFS
```

šírenie správ vlnou, pokiaľ je vytvorená usmernená sieť (resp. obojsmerná)

zložitosť – čas úmerný diametru, komunikácia N-1

možno použiť na broadcast

Prehľadávanie siete do šírky so spätnou väzbou

procedure TreeBFSecho

{vstup: S – susedné procesy, p – rodičovský proces; výstup: }

if myid <> p **then receive**(join, p, 0) **endif;**

forall v **in** S - {p} **do send**(join, v, 0) **endforall;**

if S <> {p} **then forall** v **in** S - {p} **do receive**(echo, v, 1) **endforall endif;**

if myid <> p **then send**(echo, p, 1) **endif**

end TreeBFSecho

pri echu je možné získať požadované údaje cestou od listov ku koreňu (koreň má ako rodiča samého seba)

- výpočty zo stavov jednotlivých procesov
- voľba koordinátora
- výpočet priemeru (diametra)

Konštrukcia BFS stromu vo všeobecnej sieti

```
procedure BFS  {prehľadávanie siete do šírky}
{vstup: S – susedné procesy, init ; výstup: p – rodičovský proces}
status := not tree; nrep := 0;
if myid = init then
     p:=myid; status:= tree; forall u in S do send(join, u, 0) endforall endif;
{pokiaľ počet prijatých správ ys a nys je menej, ako počet susedov}
while nrep < #S  do receive(mess, v, 0); nrep := nrep + 1 ;
     if status = tree then if mess = join then send(nys, v, 0); nrep := nrep - 1 endif
     else status := tree; p := v; forall u in S - {p} do send(join,u,0) endforall
     endif
endwhile;
if myid <> init then send(ys, p, 0) endif
endwhile
end BFS
```

Traverzovanie (DFS Depth-first search)

```
procedure DFS
{vstup: S – susedné procesy, init – začiatok prechodu; výstup: p – rodičovský proces}
status := not tree; forall u in S do used[u] :=false endforall;
if myid = init then
    p:=myid; choose u in S do send(tok,u,0) endchoose; used[u] := true;
endif
forall v in S do receive(tok, v, 0) ;
    if p = undef then p := v endif;
    if (#S - #used) > 1 then choose u in S - {p} and not used[u] do used[u]:=true;
            send(tok,u,0) endchoose
    else used[p] := true; send(tok,p,0)
    endif
endforall
end DFS
```

Bellman - Ford - najkratšie cesty

```
procedure BFdistrib
{vstup: S – susedné procesy, w – váhy prepojení k susedom, init – koreň; výstup: p –
    rodič, d – vzdialenosť ku koreňu}
if myid = init then d := 0; p := init else d:= inf endif;
for round := 1 to n-1 do
    forall v in S do  send(d,v,0) endforall;
    forall v in S do
        receive(x,v,0); if d > (x+w(v)) then d := x+w(v); p := v endif
    endforall
endfor
end BFdistrib
```

n-1 krokov a (n-1)*m správ

Floyd - Warshal - najkratšie cesty medzi všetkými dvojicami uzlov siete

```
procedure FW  {sekvenčný}
{vstup: E – spojenia uzlov, w – váhy prepojení uzlov; výstup: D[v,u] – najkratšia cesta
    medzi v a u}
forall v, u do
if v = u then D[v,u] := 0 else if (v,u) in E then D[v,u] := h(v,u)
    else  D[v,u] := inf
endforall;
forall x do
    forall v, u do D[v,u] := min ( D[v,u], D[v,x] + D[x,u] ) endforall;
endforall;
end FW
```

{ $O(n^3)$ }

Floyd - Warshal - najkratšie cesty - jednoduchý distribuovaný variant

```
procedure FWdist
{vstup: S – susedné procesy, w – váhy prepojení k susedom; výstup: d[u] – vzdialenosť k
    uzlu u, p[u] – prvý uzol na najkratšej ceste k u }
forall u do d[u] :=inf endforall; d[myid]:=0;
forall u in S do d[u]:=w(u); p[u]:=u endforall;
forall x do {všetky procesy musia vyberať rovnaké x}
    if myid = x then broadcast d[] else receive dx[] endif;
    forall v do if d[x] + dx[v] < d[v] then d[v] := d[x] + dx[v]; p[v] := p[x] endif;
    endforall
endforall
end FWdist
```

Floyd - Warshal - najkratšie cesty - Touegov algoritmus

```
procedure FWdistrib
{vstup: S – susedné procesy, w – váhy prepojení k susedom; výstup: d[u] – vzdialenosť k
    uzlu u, p[u] – prvý uzol na ceste k u }
forall u do d[u]:=inf endforall; forall u in S do d[u]:=w(u); p[u]:=u endforall; d[myid]:=0;
forall x do
    forall y in S do if p[x] = y then send(x,y,ys) else send(x,y,nys) endif endforall;
    forall y in S do receive(x,y,tag) {tag = ys or nys}; t[y] := (tag = ys) endforall;
    if d[x] < inf then
        if myid <> x then receive(dx, p[x], tab) endif;
        forall y in S do if t[y] then send(dx, y, dtab) endif endforall;
        forall v do if d[x]+dx[v]<d[v] then d[v]:=d[x]+dx[v]; p[v]:=p[x] endif endforall
    endif
endforall
end FWdistrib
```

Master - Worker

- Master – Worker paradigm

embarrassingly parallel

shared-task

backtracking

- distribúcia úloh pre pracovné uzly – výhoda paralelizácie s virtualizáciou spoločnej pamäte
- komunikačná zložitosť – množstvo vzájomných prístupov do virtualizovanej zdieľanej pamäti

Ďakujem za pozornosť !